Universite de Nice Sophia-Antipolis - UFR Sciences
Ecole Doctorale STIC

Institut Eurecom
Département de Communication d'Entreprise

THESE - THESIS

Présentée pour obtenir le titre de

Docteur en SCIENCES
De l'Université de Nice Sophia Antipolis
Spécialité : Réseaux Informatiques

Par

Alain Pannetrat

La Sécurité des Communications en Multicast
-
Secure Multicast Communications

Soutenue publiquement le 17 mai 2002, à l'Institut Eurécom, devant le jury

composé de :

M. Jean-Claude Bermond	*Directeur de Recherche CNRS,*	*Examinateur.*
M. André-Luc Beylot	*Professeur ENSHEEIT, INP Toulouse,*	*Rapporteur.*
M. Claude Castelluccia	*Chargé de Recherche I, INRIA,*	*Examinateur.*
M. Refik Molva	*Professeur, Institut Eurécom,*	*Directeur de thèse.*
M. Michel Riveill	*Professeur, UNSA/ESSI,*	*Examinateur.*
M. Gene Tsudik	*Associate Professor, U.C. Irvine,*	*Rapporteur, invité.*

Résumé

IP-Multicast[Dee89] est un mécanisme qui permet à une source de distribuer des données à un ensemble presque illimité de clients sur Internet. Ce mécanisme semble particulièrement bien adapté pour des applications de distribution commerciale de contenu à grande échelle comme, par exemple, des chaînes à péage, la diffusion de valeurs boursières ou la mise à jour de logiciels... Force est de constater que ces applications n'ont pas vu le jour comme on aurait pourtant pu l'imaginer. Un frein majeur au développement de ces applications est le manque de sécurité des communications en multicast. Il est clair qu'une distribution de contenu commercial nécessite de fournir des mécanismes qui restreignent l'accès au contenu distribué en multicast aux seuls clients légitimes de l'application. D'autre part, dans de nombreux scénarios le client doit pouvoir s'assurer de l'origine des données reçues et le producteur de contenu veut garantir cette origine afin de se protéger contre les risques d'une usurpation d'identité. Dans le cadre d'un réseau ouvert comme Internet, on fait appel à des techniques cryptographiques éprouvées pour résoudre ces problèmes entre deux entités communicantes. Mais pour des raisons de facteur d'échelle et parfois même de sécurité, ces techniques ne peuvent pas s'étendre au multicast. Cette thèse se focalise donc sur la fourniture de services de confidentialité et d'authentification spécifiquement pour les applications multicast à grande échelle.

Cette dissertation se divise donc en deux volets orthogonaux mais complémentaires : l'authentification et la confidentialité. Dans chacun de ces volets nous proposons d'abord une analyse détaillée de ces problèmes et nous mettons en valeur certains aspects nouveaux ou négligés qui sont spécifiques au multicast. Ensuite nous présentons les principales solutions existantes, en analysant leurs avantages et leurs limites. Nous terminons par nos propres solutions originales, en mettant en valeur les avantages qu'elles offrent par rapport aux solutions précédentes.

Abstract

IP-Multicast[Dee89] is a mechanism that allows a source to transmit packets to an almost unlimited number of recipients over the Internet. This mechanism would seem to be particularly well suited for large scale commercial content distribution, such as, for example, pay-TV, stock quote distribution, or software updates. However, a large scale deployment of any of these applications remains to be seen. One of the major reasons that has hindered the deployment of such applications is the lack of security protocols for multicast communications. Clearly, in many cases, the distribution of content with a commercial value requires the use of mechanisms that restricts access to the content solely to the legitimate recipients. Moreover, in many scenarios the recipient needs to ascertain the origin of the multicast content he receives and the content provider will also want to provide such a guaranty to protect himself from the potentially devastating effect of being impersonated by a third party. In an open network such as the Internet, well studied and reliable cryptographic techniques are used to provide this type of security in two party protocols. However, for scalability and sometimes even for security reasons, these techniques cannot easily be extended to the multicast setting.

The goal of this thesis is thus to study and provide basic security services designed specifically for large scale multicast applications. This dissertation is divided in two orthogonal but complementary themes: authentication and confidentiality. For each theme, we start with a detailed analysis of the problem, while highlighting new or neglected aspects of security that are specific to multicast. Then, we review existing solutions, analyzing their advantages and their limitations. Finally, we provide our own original solutions, highlighting the advantages they offer over previous proposals.

Contents

List of Figures

List of Tables

Acknowledgments

More than anybody else, I am indebted to my thesis supervisor, professor Refik Molva, who was always ready to provide help, pragmatic advice and thorough discussions. I believe that I have learned a lot from him.

I would naturally like to thank the other members of my thesis jury for their feedback and discussion: Jean-Claude Bermond, Claude Castelluccia, Michel Riveill and in particular the reviewers Andre-Luc Beylot and Gene Tsudik for their work.

I am grateful to the staff of Eurecom for providing such a great working environment and also a place where I found many friends. I would like to acknowledge in particular the wonderful friendship and support that I received from Cristina Russo during those years. My thoughts also go to many others who provided me with loads of advice and fun: David Mary, Camille Dumez, Sergio Loureiro, Pietro Michiardi, Jan Exner, Yves Roudier, Mathieu Vissac and Luis Garcés... Additionally, I would like to thank Despina Saparilla for the fruitful discussions we shared on erasure codes and Greek music.

I would like to acknowledge the unlimited support I received from many people outside Eurecom including my family back in Niort and my good old friends from University: Djuge, Finf & Anita, Hugues, Jams, Ladlou, Manelz & Caro, Mathias, Nicolas, PJ & Caro, Raf and all the gang.

This thesis is dedicated to my father.

A Dissertation Overview in French.

0.1 Introduction

Ce chapitre présente un résumé étendu de cette thèse en Français. Un certain nombre de points sont donc naturellement ici simplifiés ou omis par soucis de concision. Pour plus de détails, nous ferons référence aux chapitres appropriés de cette thèse.

0.1.1 IP-Multicast.

IP-Multicast[1] est un mécanisme qui permet à une source d'envoyer des données à un groupe de récepteurs sur le réseau Internet. La source envoie une seule copie des données et c'est le réseau qui se charge de les dupliquer quand cela est nécessaire. Au lieu d'identifier les membres du groupe par la liste de leurs adresses IP individuelles, on utilise une unique adresse IP pour identifier l'ensemble du groupe de façon implicite. Les récepteurs dans un réseau (domaine) local qui désirent faire partie d'un groupe doivent se manifester auprès de leur routeur multicast local par le protocole IGMP. Un mécanisme de routage inter-domaine spécifique au multicast est utilisé pour transmettre les paquets destinés à un certain groupe, à l'ensemble des réseaux locaux multicast qui le composent. Ce mécanisme de routage inter-domaine fait aussi intervenir des routeurs multicast qui ne sont pas nécessairement eux-mêmes des routeurs IGMP ayant des récepteurs du groupe, mais qui sont simplement des entités faisant office d'intermédiaires dans le routage multicast.

Beaucoup d'applications Multicast utilisent UDP comme couche de transport, avec parfois l'addition d'un protocole supplémentaire comme, par exemple, RTP[SCFJ96]. Ceci implique que l'on a pratiquement aucune garantie de fiabilité sur le transport des données. Certaines applications multimédias sont d'ailleurs construites pour tolérer des pertes ou des délais comme ceux que l'on peut observer dans un réseau non fiable. Ces applications adaptent la qualité de leur reproduction audio/vidéo de façon intelligente aux conditions réseaux. Il existe un certain nombre de propositions pour le multicast fiable (voir [Obr98]), cependant elles souffrent toutes d'un certain nombre de difficultés de passage à l'échelle. Dans cette thèse nous souhaitons nous intéresser à la sécurité multicast pour des applications de très grande échelle, et nous supposerons que le mécanisme de transport est peu fiable afin de couvrir le

[1] Dans le texte, nous garderons les mots anglais multicast et unicast au sigulier.

plus grand nombre de scénarios.

L'ensemble des points précédents est décrit de façon détaillée au début du chapitre 1 de cette thèse. Cependant, nous pouvons résumer les caractéristiques principales du multicast qui sont pertinentes pour la conception de protocoles de sécurité en 3 points :

- Le groupe n'est pas identifié explicitement par la liste des récepteurs mais de façon implicite par une adresse de groupe : un protocole de sécurité doit lui aussi pouvoir traiter le groupe de façon globale.
- Il existe un arbre de routage qui transmet les données multicast entre la source et les différents récepteurs. Cet arbre fait intervenir un certain nombre d'éléments intermédiaires de routage : pourquoi ne pas les faire intervenir dans la sécurité ?
- Le transport des données multicast n'est pas fiable : un protocole de sécurité doit pouvoir tolérer les pertes.

0.1.2 La Sécurité en Multicast.

Le multicast semble être un mécanisme particulièrement intéressant pour des applications commerciales de grande échelle, comme par exemple:

- La diffusion de télévision à péage.
- La diffusion de contenu audio de haute qualité.
- La distribution de cotations boursières ou d'informations temps réel en général.
- La mise à jour automatique de logiciels, etc.

Force est de constater pourtant que ce genre d'application n'est pas déployé sur Internet. Un des freins majeurs à ce déploiement se trouve sans doute dans le manque de mécanismes de sécurité adaptés au multicast. En effet, pour pouvoir exploiter ces applications de manière commerciale, il est nécessaire de garantir au moins deux besoins de sécurité fondamentaux : la confidentialité et l'authentification. Des mécanismes de confidentialité sont nécessaires pour restreindre la diffusion des données aux seuls récepteurs légitimes de l'application (ceux qui, par exemple, en ont payé l'accès.) L'authentification est nécessaire pour permettre aux récepteurs de s'assurer de l'origine des données multicast qu'ils reçoivent et pour empêcher une source de contenu d'usurper l'identité commerciale d'une autre.

Pour les communications unicast, il existe un certain nombre de protocoles qui fournissent ces deux services de façon souple et puissante. Comme nous le détaillons au début de la thèse, ces protocoles unicast ne peuvent pas s'étendre au multicast principalement pour des raisons de facteur d'échelle mais aussi pour des raisons de sécurité.

0.1.3 Présentation de la Thèse.

L'objectif de cette thèse est d'analyser, de discuter et de proposer des solutions aux deux besoins principaux auxquels les applications multicast de grande échelle peuvent être confrontées:

la confidentialité et l'authentification

De façon naturelle, nous avons donc divisé cette thèse en deux parties complémentaires qui traitent chacune d'un de ces besoins.

La première partie de cette thèse que nous résumons dans les sections 0.2 et 0.3 traite de l'authentification des flux de donnés sur un grand groupe IP-Multicast non fiable. Cette première partie se divise elle-même en trois chapitres. Nous commençons dans le chapitre 2 avec une analyse des besoins et des contraintes, en mettant l'accent sur la nécessité de tolérer les pertes de paquets. Suite à cela, dans le chapitre 3, nous établissons un état de l'art détaillé des propositions actuelles en matière d'authentification multicast. Finalement, dans le chapitre 4 nous proposons notre propre solution qui présente en particulier l'avantage d'avoir un coût en espace par paquet plus faible que toute autre solution actuelle.

La deuxième partie de cette thèse que nous résumons dans les sections 0.4 et 0.5 se focalise sur la confidentialité multicast. Comme nous l'avons fait pour l'authentification, nous débutons avec une étude utile et détaillée du problème et nous définissons un certain nombre de besoins centraux pour la confidentialité des applications multicast de grande échelle. La suite de cette deuxième partie de thèse est dédiée aux principaux schémas de confidentialité multicast. Le chapitre 6 propose un état de l'art critique du domaine, tandis que les chapitres 7 et 8 décrivent nos solutions qui sont basées sur l'utilisation des éléments intermédiaires du réseau sans engagement de confiance.

0.1.4 Contributions.

Les contributions de cette thèse se situent à deux niveaux. Nous proposons tout d'abord une analyse des problèmes et des besoins liés à l'authentification et à la confidentialité multicast qui est plus approfondie que celle que nous pouvons trouver dans d'autres travaux. En particulier, dans le chapitre 5, nous proposons un nouveau besoin pour la confidentialité: l'endiguement (a. *containement.*) Deuxièmement, nous fournissons nos propres solutions originales aux problèmes décrits. Notre algorithme d'authentification multicast génère un coût en terme d'espace par paquets qui est, dans la majorité des cas, plus bas que dans les autres solutions proposées à ce jour. Les deux protocoles de confidentialité multicast que nous décrivons répondent à un nombre plus important de besoins que les autres schémas proposés actuellement. En particulier, ce sont les seuls protocoles de confidentialité multicast qui offrent une propriété d'endiguement sans engagement de confiance envers les éléments intermédiaires du réseau.

0.2 Authentification Multicast

0.2.1 Besoins et Contraintes

Dans le chapitre 2 nous rappelons tout d'abord dans quel cadre exact se situe le problème que nous traitons : Nous ne cherchons pas à authentifier un document dans sa totalité, une fois qu'il a été reçu par le récepteur. Au contraire, nous considérons les données envoyées par une source comme une suite potentiellement illimitée de paquets qui doivent être authentifiés au fur et a mesure de leur réception et cela malgré la perte de certain d'entre eux. Nous rappelons aussi les deux principaux types d'authentification : l'authentification de l'origine et la non répudiation.

Nous établissons un certain nombre de critères pour étudier les applications multicast:

Play-back ou Direct.

Il est important de différencier les schémas d'authentifications multicast en fonction du type de données qu'ils traitent. En effet, pour des données préenregistrées destinées au play-back, il est possible de calculer les informations d'authentifications à l'avance et sans contrainte de temps. On peut ensuite les insérer dans les données et les envoyer de façon normale. Pour des données temps réel, comme la retransmission d'un événement sportif en direct, il faut pouvoir faire les calculs cryptographiques nécessaires pour authentifier les données en temps réel et insérer l'information dans le flux immédiatement. Naturellement, un schéma qui fonctionne pour le direct, fonctionne de facto pour le play-back. Il est donc intéressant de fournir des solutions efficaces pour le direct.

La robustesse.

La tolérance aux pertes est un souci majeur pour l'authentification des données en multicast. Ceci tient au fait que la plupart des solutions pour l'authentification créent une dépendance logique entre les paquets de données.

Cette dépendance permet de réduire le coût de l'authentification mais en contrepartie, la perte d'un paquet peut alors avoir un impact sur notre capacité à authentifier d'autres paquets qui en dépendent. Dans cette thèse, nous suivons une approche récente dans le domaine de l'authentification multicast qui consiste à adapter le mécanisme d'authentification à des schémas de perte typique de l'Internet plutôt que d'employer un schéma qui tolère des pertes arbitraires, ce qui est naturellement plus coûteux. Dans cette thèse, nous employons un modèle maintes fois suggéré pour modéliser les pertes dans le réseau Internet : le modèle de Gilbert, ou chaîne de Markov à deux états.

La modularité

Comme un flux de données peut être particulièrement long, il n'est pas toujours possible ou souhaitable, en terme de souplesse, d'obliger le récepteur à être présent depuis le début du flux. Idéalement, on souhaite permettre à un récepteur de pouvoir authentifier les données à partir d'un point arbitraire dans le flux, ou tout du moins sur la base d'un bloc de plusieurs paquets suffisamment petit. Cette propriété, nous l'appelons la modularité (a. *joinability*).

Le délai.

Dans un cas idéal, on souhaite bien sûr pouvoir authentifier les paquets individuellement dès leur réception. Dans la pratique, l'authentification des paquets peut dépendre de d'autres paquets qui ne sont pas encore reçus, ce qui peut introduire un délai supplémentaire d'authentification. Ce délai d'authentification (a. *latency*), nous le quantifions souvent en terme de paquets. Notre but est naturellement de limiter ce délai à une valeur raisonnable.

Le tampon (coté serveur).

Côté serveur, le pendant du délai d'authentification est la taille du tampon de paquets. En effet, si certaines informations d'authentification concernent plusieurs paquets, il est parfois nécessaire de les conserver temporairement sur le serveur pour faire des calculs avant de les envoyer. Naturellement, plus le tampon est grand, moins le schéma convient pour du direct, puisque par définition, on souhaite envoyer les données immédiatement pour du direct.

Coût

Le critère final qui permet de départager les schémas d'authentification entre eux est leur coût à la fois en terme de ressources de calcul et en terme d'espace par paquet. En effet un schéma qui nécessite des temps de calculs important par paquet va nécessairement limiter le débit de paquets que le serveur peut produire ou que le récepteur peut vérifier. Pour authentifier les paquets, il est nécessaire de leur ajouter une certaine information d'authentification. Si celle-ci est coûteuse en terme d'espace, elle réduit d'autant la bande passante disponible.

La difficulté principale dans la construction d'un schéma d'authentification est donc de trouver un compromis entre les différents critères cités précédemment. Dans cette thèse, nous avons choisit de nous focaliser sur des mécanismes d'authentification utilisables pour le direct puisqu'ils sont plus généraux.

0.2.2 Etude de l'état de l'art.

Dans le chapitre 3 de cette thèse, nous passons en revue les différents systèmes d'authentification multicast proposés actuellement :

1. Les MMAC ou Code d'Authentification de Message Multicast de Canetti et al. [CGI+99].

2. Les signatures rapides de Rohatgi [Roh99] et de Wong et Lam [WL99].

3. TESLA : l'authentification à engagement temporel de Perrig et al. [Riz97].

4. Les techniques hybrides qui amortissent une signature sur plusieurs paquets avec des techniques synthétiques (hachage et MAC):

   (a) Les arbres de hachage de Wong et Lam [WL99].

   (b) Chaînes de Hachage de Golle et Modadugu [GM01].

   (c) EMSS de Perrig et al. [PCTS00].

Nous établissons que les solutions 1, 2 et 4(a) ne sont pas utilisable en pratique dans un certain nombre de cas pour des raisons de coût en espace par paquet. La solution 1 possède aussi une limitation théorique en terme de sécurité. La solution 3 est très intéressante mais nécessite une synchronisation temporelle sécurisée entre la source et l'ensemble des récepteurs, ce qui peut être complexe à mettre en oeuvre. En outre, la solution 3 n'offre pas de non répudiation et doit être modifiée pour fournir la propriété de modularité. Les solutions 4(b) et 4(c) sont voisines et répondent à la plupart des besoins évoqués. Ces solutions sont appelées hybrides car elles amortissent une signature digitale sur un bloc de plusieurs paquets en utilisant des techniques symétriques comme les hachages et les MAC. La solution 4(c) offre une meilleure gestion de la signature de bloc par rapport à 4(b).

0.3 Un Protocole d'Authentification pour le Direct

Le protocole d'authentification multicast que nous proposons dans cette thèse fait donc partie de la famille des techniques hybrides. Les paquets sont groupés en bloc de taille b. Pour chaque paquet P_i d'un bloc nous calculons une somme de contrôle cryptographique h_i par une fonction de hachage comme MD5 ou SHA. Ensuite, nous calculons une signature digitale σ sur l'ensemble de ces sommes de contrôle $\{h_1, ..., h_b\}$. Pour garantir un certain degré de robustesse, nous utilisons des codes de reconstruction (a. *Erasure Codes*) d'une façon originale afin de permettre au récepteur de reconstituer la suite $\{h_1, ..., h_b\}$ et sa signature. Cette méthode nous permet d'atteindre des coûts d'authentification en terme d'espace par paquet qui sont plus faible que toutes les méthodes d'authentification actuelles dans la plupart des scénarios. D'autre part dans notre schéma, nous proposons une méthode concrète pour transmettre la signature de chaque bloc sans faire de coûteuses retransmissions multiples, comme c'est le cas dans les autres solutions hybrides.

Dans la section 4.2 nous introduisons des codes reconstructeurs[Riz97]. La générateur de code $C_{k,r}$ prend un ensemble X de k paquets sources et produit $(k+r)$ paquets, dont les r derniers représente la redondance Y. Nous noterons donc $\{X, Y\} \leftarrow C_{k,r}(X)$. L'algorithme D_k de reconstruction de données qui prend un ensemble Z de k paquets recus et produits les k paquets sources d'origine sera noté $X \leftarrow D_k(Z)$.

Le coeur de notre protocole est basé sur un algorithme de génération d'étiquettes : $\mathcal{T}_{[b,p]}$ de paramètres b, le nombre de packet dans un bloc, et p, le taux de perte maximal d'un bloc. Cet algorithme accepte en entrée une suite de b sommes de contrôle $\{h_1, ..., h_b\}$ et génère en sortie une suite $\{\tau_1, ..., \tau_b\}$ de b étiquettes.

```
Génération d'étiquettes: $\mathcal{T}_{[b,p]}$
ENTREE: $\{h_1, ..., h_b\}$
SORTIE: $\{\tau_1, ..., \tau_b\}$
$\{X;\overline{X}\} \leftarrow C_{b,\lceil pb\rceil}(X)$
$\sigma \leftarrow \mathcal{S}(H(h_1||...||h_b))$
$\{Y;\overline{Y}\} \leftarrow C_{\lfloor b(1-p)\rfloor,\lceil pb\rceil}(\overline{X}||\sigma)$ où $Y = \overline{X}||\sigma$.
Partager $\{Y;\overline{Y}\}$ en $b$ étiquettes de taille égale $\{\tau_1, ..., \tau_b\}$.
```

La principale propriété de cet algorithme est la suivante :

Soit $\Pi = [P_1, ..., P_b]$ un bloc de b paquets et $\{h_1, ..., h_b | h_i \leftarrow H(P_i)\}$ l'ensemble des haschages de ceux-ci. Si nous calculons $A = \{\tau_1, ..., \tau_b\} \leftarrow \mathcal{T}_{[b,p]}(\{h_1, ..., h_b\})$ alors tout sous ensemble d'au moins $\lfloor b(1-p)\rfloor$ paquets dans Π peut etre authentifié par tout sous ensemble d'au moins $\lfloor b(1-p)\rfloor$ étiquettes dans A.

Un flux multicast se subdivise donc en une suite $B_1...B_\infty$ de blocs. Pour chaque bloc B_i nous calculons une suite d'étiquettes $\{\tau_1, ..., \tau_b\}$. Ces étiquettes nous les ajoutons soit au bloc suivant $B_{(i+1)}$, soit au bloc précèdant $B_{(i-1)}$ ou tout simplement au bloc courrant B_i. Chacune de ces configurations a des avantages et des inconvénients que nous analysons dans la section 4.3.2.

Reste à déterminer b et p en fonction des schémas de pertes observés dans le réseau et des besoins de l'application. La valeur b dépend essentiellement du délai d'authentification maximal permis, car naturellement plus b est grand plus le délai augmente. Si b est petit, cela demande plus de ressource de calcul. Nous préconisons de choisir b le plus large possible en fonction des contraintes de l'application en terme de délai d'authentification. Pour déterminer la valeur de p, nous modélisons les pertes du réseau par une chaîne de Markov à deux états, comme suggéré dans de nombreux travaux [BFPT99, YMKT99]. Le choix de p se fait par simulation de cette chaîne de Markov sur un grand nombre de paquets de façon à garantir une proportion de bloc authentifié supérieur à 99%, comme décrit en 4.3.3.

Dans la section 4.4, nous comparons notre protocole avec les autres approches voisines et nous montrons que pour des paramètres de perte du réseau raisonnables, notre approche est moins coûteuse en terme d'espace par paquet. En contrepartie, notre approche a un coût en terme de calcul supérieur aux autres approches, ceci étant dû aux codes de reconstruction. Comme nous le soulignons en 4.4, ce coût reste cependant très raisonnable.

Nous discutons aussi des autres aspects de notre protocole en suivant la liste de critères établis plus haut et nous montrons que notre protocole constitue dans de nombreux cas une alternative intéressante aux autres solutions qui ont été proposées jusqu'à présent.

0.4 Confidentialité

0.4.1 Définitions et Besoins

Dans un environnement ouvert comme Internet, la confidentialité des données transmises en Multicast s'obtient en les protégeant par des moyens cryptographiques de façon à restreindre leur diffusion aux seules entités qui ont légitimement le droit d'y accéder. L'exemple le plus typique d'une application ayant ce besoin est celle de la diffusion d'une chaîne à péage. Dans cette thèse nous définissons un certain nombre d'éléments qui participent à un protocole de confidentialité multicast. Ces éléments sont :

Les récepteurs : on appelle ici récepteur toute entité capable de recevoir des données venant d'un certain groupe multicast, même si elle ne peut pas les déchiffrer.

Les membres : on appelle membre tout récepteur possédant des moyens cryptographiques légitimes pour accéder au contenu transmis en multicast.

La source : on appelle source, l'entité qui génère un contenu et le transmet aux membres de façon sécurisée. Dans cette thèse, nous nous focalisons sur des scénarios ayant une ou quelques sources et un ensemble très important de membres. Comme l'ont déjà remarqué certains auteurs [HC99], cette approche couvre la plupart des applications commerciales de grande échelle.

Les gestionnaires de membres : Un gestionnaire de membre est une entité qui gère la liste des membres. Lorsqu'un récepteur souhaite devenir membre il contacte un gestionnaire. Si le récepteur est autorisé à devenir membre, le gestionnaire lui transmet les paramètres cryptographiques lui permettant d'accéder au contenu des paquets Multicast. On dit alors que le récepteur est ajouté au groupe. Réciproquement, le gestionnaire de membre peut décider de supprimer la possibilité pour un membre d'accéder au contenu des paquets multicast. On dit alors que le membre est supprimé du groupe.

Les intermédiaires : dans certains protocoles multicast, les routeurs ou proxies qui participent à la transmission des données Multicast sont aussi mis à contribution pour assurer la confidentialité des données. Nous appelons ces éléments actifs, des intermédiaires.

Un protocole de confidentialité Multicast permet donc à une source de transmettre des données de façon restreinte à un ensemble de membres choisis. Le premier but d'un tel protocole est donc d'interdire aux non-membres d'accéder aux données Multicast. L'ensemble des membres est, la plupart du temps, dynamique car il évolue avec les ajouts et les suppressions de certains membres. Le deuxième but d'un protocole de confidentialité Multicast est donc de garantir ce que nous appelons le secret postérieur et antérieur. Lorsqu'un membre est supprimé du groupe il ne doit plus avoir accès aux données Multicast, c'est ce que nous appelons le *secret postérieur*. Symétriquement, un membre ne doit pas pouvoir déchiffrer les données qu'il a peu enregistrer et qui ont été transmises avant qu'il ne soit ajouté au groupe, c'est ce que nous appelons le *secret antérieur*.

Un aspect particulièrement important des communications Multicast est qu'elles impliquent un nombre potentiellement immense de participants. Dans une telle configuration, il est pratiquement certain que les paramètres d'accès de certains membres vont être compromis volontairement ou non. Un membre peut être tenté de divulguer ses paramètres d'accès à un récepteur pour lui permettre lui aussi d'accéder au groupe. Pire : un pirate informatique peut publier des paramètres d'accès sur un site internet. Il est donc primordial de limiter l'impact de la compromission des paramètres d'accès détenus par un membre. C'est ce que nous appelons l'*endiguement*.

Dans la même ligne d'idées, il est important que le protocole résiste à la collusion de certains de ses membres. Un ensemble de membres qui échangent entre eux leurs paramètres d'accès ne doivent pas être en mesure d'élever leurs privilèges au delà de la somme des privilèges correspondants aux paramètres échangés.

Dans un certain nombre de protocoles de confidentialité Multicast, on utilise les intermédiaires comme participants actifs dans la sécurisation des données. Ces éléments sont potentiellement nombreux et il est donc logique d'être aussi prudent avec les intermédiaires qu'avec les membres. Nous chercherons donc à limiter la confiance que nous plaçons dans les éléments intermédiaires.

A ces besoins de sécurité viennent s'ajouter des besoins purement algorithmiques :

La scalabilité de traitement : Le coût supporté par une entité individuelle, que ce soit la source, les intermédiaires ou les membres, doit pouvoir rester raisonnable quelle que soit la taille du groupe et quelle que soit la durée de participation d'un membre dans le groupe.

La robustesse de participation des membre : Les membres doivent pouvoir accéder au contenu des paquets multicast dès leur réception.

Ce dernier point peut paraître évident mais, en pratique, il peut être difficile à garantir. En effet, des problèmes de synchronisation entre les paramètres d'accès détenus par un membre et ceux effectivement nécessaire pour accéder aux données peuvent apparaître dans certains protocoles. Ces problèmes de synchronisation peuvent induire des problème de robustesse de participation de certains membres.

Nous avons donc établi les objectifs suivants pour un protocole de confidentialité multicast :

- Confidentialité des données.
- Secret antérieur et postérieur.
- Scalabilité de traitement
- La robustesse de participation des membres
- L'endiguement

- Résistance à la collusion
- Une confiance limitée aux éléments intermédiaires

Le chapitre 6 de cette thèse propose une étude de l'état de l'art des protocoles de confidentialité multicast. Nous commençons cette étude en traitant des hiérarchies de clés logiques qui furent introduites par Wong, Gouda et Lam[WGL98] et simultanément par Wallner et al. [WHA98]. Nous étudions les différentes propositions qui ont suivi cette approche en l'enrichissant : [CEK+99, CGI+99, CVSP98, CWSP98, PST01, WL00, MS98]. Nous concluons que malgré un certain nombre de progrès, certains problèmes de robustesse de participation des membres demeurent. D'autre part, nous soulignons que les hiérarchies de clés logiques n'offrent pas d'endiguement.

Ensuite, nous nous intéressons aux arbres de rechiffrement qui ont été proposés par Suivo Mittra dans le système IOLUS [Mit97]. Dans cette approche on utilise un certain nombre d'éléments intermédiaires comme participants actifs dans la sécurité des données. Ces éléments intermédiaires déchiffrent et rechiffrent les données, c'est pour cela que nous appelons cette famille d'approches qui est aussi suggérée dans [HM97], des arbres de rechiffrement. Cette approche permet d'avoir un bon facteur d'échelle et offre la propriété d'endiguement. Malheureusement cette approche néçessite de faire une confiance totale aux intermédiaires.

Nous présentons aussi le système MARKS[Bri99] qui n'est pas tout à fait un schéma de confidentialité multicast, mais plutôt un schéma d'abonnement non révocable. En effet, dans MARKS le récepteur s'abonne au groupe pour une certaine durée, et il est ensuite impossible de lui retirer l'accès aux données avant le terme de son abonnement. Cependant ce système présente de nombreux avantages en terme de robustesse et de scalabilité. Par contre, il n'offre pas d'endiguement.

0.5 Des solutions pour la confidentialité multicast.

Nos propres protocoles que nous présentons dans les chapitres 7 et 8 sont des arbres de rechiffrement. Notre idée est de proposer une approche qui offre tous les avantages des arbres de rechiffrement sans hériter de leur faiblesse majeure qui est de faire une confiance totale aux éléments intermédiaires.

0.5.1 La confidentialité par suites de chiffre.

Nous résumons ici brièvement les points principaux du chapitre 7.

Dans notre schéma, nous définissons tout d'abord des transformations cryptographiques ayant une structure particulière :

Soit $\mathcal{G} = \{g_i : \mathcal{N} \mapsto \mathcal{N},\ i \in \mathcal{A}\}$ un ensemble de permutation sur un ensemble $\mathcal{N}$, indexé dans $\mathcal{A}$. On appelle $\mathcal{G}$ un *groupe de chiffrement* s'il vérifie les propriétés suivantes :

$(\mathcal{G}, \circ)$ forme un groupe par la loi de composition des fonctions $\circ$

$(\mathcal{G}, \circ)$ est peut être indexé par composition: Il existe un algorithme en temps polynomial $Comp : \mathcal{A}^2 \mapsto \mathcal{A}$ qui à partir d'une pair d' index $(i, j) \in \mathcal{A}^2$, calcule $k = Comp(i, j) \in \mathcal{A}$ tel que $g_{(k)} = g_{(j)} \circ g_{(i)}$.

Si on a une suite $F = \{f_1, ..., f_N\}$ d'éléments de $\mathcal{G}$ il existe une fonction $g \in \mathcal{G}$ tel que $g = (f_1 \circ ... \circ f_N)^{-1}$. Cette fonction, nous l'appelons la fonction d'inversion de la suite F.

Notre approche consiste à choisir un groupe de chiffre $\mathcal{G}$ et à distribuer des éléments aléatoires $f_0, ..., f_N$ de ce groupe à l'ensemble des éléments intermédiaires de l'arbre de transmission multicast ainsi qu'à la source. Chaque feuille (ou sous-groupe de membres) de l'arbre reçoit la fonction d'inversion correspondant à l'ensemble des transformations effectuées sur la branche reliant la source à la feuille. La source souhaitant transmettre une information M à tous les membres lui applique une première transformation f_0, puis envoie les données au réseau. Chaque élément intermédiaire i qui reçoit un paquet lui applique lui-même une transformation f_i et transmet les données au réseau. Quand les données arrivent dans une feuille, les membres contenus dans celle-ci lui appliquent la fonction inverse correspondant à l'ensemble des transformations effectuées sur la branche reliant la source à la feuille. Cette transformation inverse permet aux membres de la feuille d'obtenir M.

Pour garantir qu'un adversaire externe ne puisse pas obtenir des informations sur M en interceptant les messages transmis dans le réseau, il est nécessaire que les transformations utilisées dans G aient certaines propriétés de sécurité qui permettent d'assurer la confidentialité des données transmises. Dans cette thèse, nous envisageons deux type de transformation ayant à la fois les propriétés de groupe de chiffrement et de sécurité nécessaires :

- l'exponentiation dans un corps premier 7.5.
- l'exponentiation modulo un composite (extension de RSA) 7.6.

Notre approche par arbre de chiffrement permet d'avoir une forme d'endiguement similaire à celle qu'on trouve dans IOLUS mais avec un avantage de taille : Nous ne faisons pas confiance aux éléments intermédiaires. En effet, ceux-ci se contentent d'effectuer certaines transformations sans accéder directement aux données. Il existe des scénarios d'attaques complexes possibles mais ceux-ci nécessitent la compromission de plusieurs éléments, comme nous le décrivons de façon plus détaillée en 7.3.4.

Le problème de cette approche réside dans son coût. En effet, les seules transformations vérifiant les propriétés nécessaires à notre schéma sont des mécanismes de cryptographie asymétrique. Il n'est pas possible d'utiliser notre schéma de façon réaliste pour chiffrer des flux entiers de données. Nous ne pouvons que l'utiliser pour la distribution de clés.

0.5.2 La confidentialité par chiffrements multiples commutatifs.

Nous avons souhaité proposer une autre solution qui s'inspire dans sa philosophie de la précédente mais qui puisse être adaptée aux données.

Il existe un mode de chiffrement dit "counter based" qui permet de chiffrer des données en générant une suite pseudo aléatoire d'octets qui sont combinés avec le texte clair par un ou exclusif binaire. Ce mode de chiffrement est aussi celui que l'on utilise de façon typique avec les "stream ciphers". Soit f_a une primitive de chiffrement comme DES ou AES, opérant sur k bits dont la clé est a, le chiffrement d'un message M se fait comme suit :

- La fonction $\mathcal{E}_a(\sigma, x)$ est definie comme suit:

```
soit $x = x_1 x_2 ... x_n$
pour $i = 1, ..., n$ faire $y_i = f_a(\sigma + i) \oplus x_i$.
retourner $(\sigma, y_1 y_2 ... y_n)$.
$\sigma \leftarrow \sigma + n$
```

Le compteur σ est conservé et maintenu par la source. De façon presque similaire nous obtenons l'algorithme de déchiffrement :

- La fonction $\mathcal{D}_a(\sigma, y)$ est definie comme suit:

```
soit $y = y_1 y_2 ... y_n$
pour $i = 1, ..., n$ faire $x_i = f_a(\sigma + i) \oplus y_i$
retourner $x = x_1 x_2 ... x_n$
```

La sécurité de cet algorithme à été justifié dans [BDJR97]. Dans cette thèse, nous proposons une extension XORC$^l = \{\mathcal{E}^{(l)}, \mathcal{D}^{(l)}\}$ à l clés comme suit :

- $\mathcal{E}^{(l)}_{a_1, a_2, .., a_l}(\sigma_1, ..., \sigma_l, x)$:

```
soit $x = x_1 x_2 ... x_n$
pour $i = 1, ..., n$ faire $y_i = f_{a_1}(\sigma_1 + i) \oplus ... \oplus f_{a_l}(\sigma_l + i) \oplus x_i$.
retourner $(\sigma_1 ... \sigma_l, y_1 y_2 ... y_l)$.
pour $j = 1, ..., m$ faire $\sigma_j \leftarrow \sigma_j + n$
```

- $\mathcal{D}^{(l)}_{a_1, ..., a_l}(\sigma_1, ..., \sigma_l, y)$:

```
soit $y = y_1 y_2 ... y_n$
for $i = 1, ..., n$ do $x_i = f_{a_1}(\sigma_1 + i) \oplus ... \oplus f_{a_l}(\sigma_l + i) \oplus y_i$
retourner $x = x_1 x_2 ... x_n$.
```

Les propriétés de cet algorithme XORC^l sont décrites dans 8.2.2. Nous pouvons les résumer de la façon suivante :

- L'ordre de chiffrement et de déchiffrement de l couches sont indépendant.
- Si on connaît la clé correspondant à une couche i, $0 < i \leq l$, il est possible de substituer la couche i par une autre couche générée par une clé arbitraire.

Nous utilisons le chiffrement en l couches de la façon suivante :

- La source chiffre les données avec l couches.
- Chaque élément intermédiaire substitue une couche par une autre. Ce qui implique un seul chiffrement et un seul déchiffrement.
- Les récepteurs déchiffrent l couches pour accéder aux données.

La philosophie reste la même que dans notre précédent schéma : les éléments intermédiaires n'accèdent pas directement aux données, et se contente de faire une transformation de celles-ci.

La sécurité de ce schéma est évalué de façon précise dans la section 8.4.

Chapter 1

Introduction

The subject of this dissertation is the security of Multicast communications. Clearly, both *security* and *multicast* encompass a broad number of problems. In this work we have chosen to deal with the two most fundamental aspects of security in the context of multicast: *confidentiality* and *authentication* services for multicast applications. This chapter is designed to introduce and motivate the work presented in this thesis. We start with a overview of IP-Multicast and next we highlight the main security requirements of large scale applications which are build upon multicast mechanisms. Finally, we use these elements to draw the outline of this dissertation.

1.1 IP Multicast

IP-Multicast was introduced by Steve Deering [Dee89, Dee91] and describes a set of mechanisms which allow the delivery of a packet to a *set* of recipients rather than just one recipient as in *unicast* communications. The recipients that receive the same multicast packets are said to be part of the same *multicast group*. In the multicast setting, the sender sends a single copy of a packet and the network takes care of duplicating the packet at proper branching points in order for each recipient to receive a copy of the original packet. With this approach, only one copy of the original packet will be transmitted on most of the network links, which saves resources compared to an equivalent set of unicast communications, as illustrated in the example of figure 1.1.

While many local area network protocols such as Ethernet or Token-Ring already have broadcast or selective broadcast capabilities, they are usually interconnected through a set of heterogeneous networks in the Internet. In his work, Deering proposed an extension to IP to achieve multicast transmission both over the local area networks and the internetwork that connects them together.

Applications

Multicast is by definition a natural foundation for applications such as:

- Video/audio conferences (for example, [Tur94]).

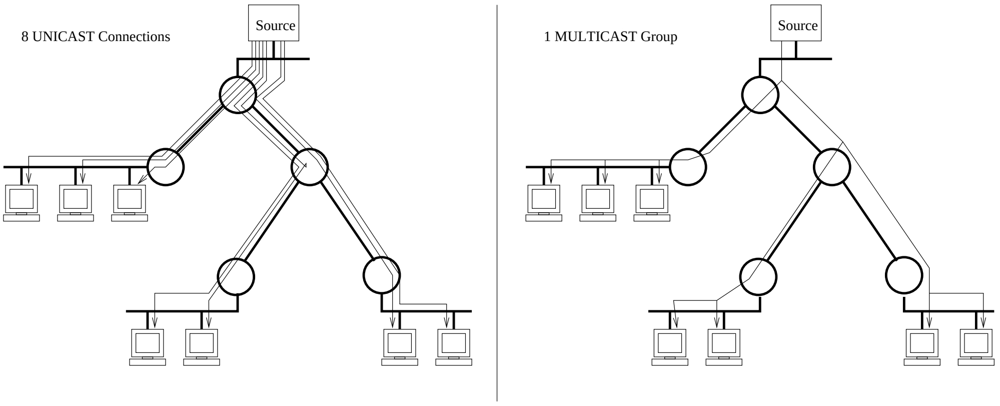

Figure 1.1: Unicast or Multicast

- Commercial television broadcastings and pay-per-view.
- The distribution of software updates (such as anti-virus database updates).
- News feeds and stock quote distribution.
- Cooperative applications, shared white boards (for example The LBL Shared White Board `http://www-nrg.ee.lbl.gov/wb/`).
- Interactive distributed video games[1].

Today a few of these applications have been implemented over the Internet, some over IP-Multicast while others use multiple unicast connections despite the scalability benefits of multicast. As we will see in the next section, security issues are one of the reasons that have limited the deployment of IP-Multicast.

1.1.1 Primary Multicast Mechanisms

To allow packets to be distributed in a scalable manner to a potentially unlimited number of receivers, IP-Multicast does not specify the individual IP addresses of all receivers, but instead uses a single group address to identify the group of recipients. In practice IP-Multicast packets are quite similar to unicast IP packets, except that the destination IP address is chosen in the range 224.0.0.0 to 239.255.255.255, also called class D in IPv4 (there are further assignment restrictions, see [Aut]). Receivers that are interested in receiving packets from a certain group "subscribe" to the corresponding group address and the multicast routing protocols take care of forwarding the packets to these receivers. To achieve this, multicast relies on two primary mechanisms: first, *a local membership protocol* which allows recipients to signify to a subnet router their interest in receiving multicast packets, and second, *an inter-subnet multicast routing protocol* that creates a multicast delivery tree between the subnet routers.

[1] There are distributed games which have a big success such as Ultima Online `http://www.uo.com/`, but these are build over unicast.

A local membership protocol

The local membership protocol runs between receivers and their closest designated multicast router. The protocol allows the recipients to specify to the local multicast router their interest in receiving packets from a specific multicast group. The local designated multicast router uses this membership information to decide which multicast packets it will need to forward from the Internet down to the local receivers. Currently, the protocol that performs this task is the Internet Group Management Protocol or IGMP[Fen97].

An inter-subnet multicast routing protocol

To interconnect the IGMP local multicast routers together over the Internet, multicast uses a specific internetwork routing mechanism. This routing mechanism constructs a delivery tree between the multicast subnets. In fact, quite a few of these routing algorithms have been proposed [Moy94, Bal97, WP98, EFH$^+$97, ...] each with their advantages and their drawbacks. These routing protocols are usually classified in two broad categories:

1. *Source based trees:* This type of algorithm creates one routing tree for each sender or source. This approach usually results in the construction of an efficient delivery tree but suffers from limitations due to its high cost in (sometimes unnecessary) network resources.

2. *Shared trees:* This type of algorithm creates a single routing tree that is shared between all the recipients of a specific group, regardless of the sender. This approach makes a much more efficient use of network resources but may create traffic bottlenecks and increased transmission delays.

The choice of a multicast routing protocol may further depend on the application constraints, the density and dissemination of the receiver group and other factors. A detailed study of multicast routing algorithms is beyond the scope of this work. Nevertheless, there is a characteristic that is common to all protocols: the use of a tree of network components to interconnect the multicast local routers. This tree involves other routers that are not necessarily IGMP enabled but still implement one or several inter-domain multicast routing protocols. In practice, since not all routers on the Internet are multicast enabled, tunneling mechanisms are used between multicast enabled routers to create parts of the multicast network structure. This creates a multicast specific virtual network on top of the Internet: the MBone (see The MBONED charter `http://www.ietf.org/html.charters/mboned-charter.html`).

Source Specific Multicast

IP-Multicast does not require the sender to be part of the multicast group, which means that the sender is not required to register to a local IGMP router to send packets to the group. The number of senders is potentially unlimited. Recently, however, an opposite approach has been taking some ground: SSM or Source Specific Multicast. This approach was introduced by Holbrook and Cheriton in the EXPRESS framework[HC99] and can be found today in commercial enterprise solutions (see for example, IPTV `http://www.cisco.com/iptv`).

In SSM, receivers subscribe to a (*source, group*) address pair instead of just a single group address. Multicast addresses ranging from 232.0.0.0 through 232.255.255.255 have been dedicated by the IANA[Aut] specifically for SSM protocols. The first advantage of using SSM is that since each multicast flow is identified by both a source and a group address there can be as many multicast groups as there are senders in the Internet. The number of multicast groups is not limited by the number of class D addresses available. Consequently, while traditional multicast mechanisms need to allocate dynamically a group address to multicast applications on demand, such an allocation scheme is completely unnecessary for SSM.

More importantly, as highlighted in EXPRESS[HC99], SSM corresponds to the needs of large scale multicast commercial applications where there is one of few sources and a very large number of recipients. As an illustrative example, consider the delivery of television broadcasts over the Internet. While this dissertation reviews and presents many multicast security protocols which work in the general setting, a particular emphasis is implicitly put on a *1-to-n* model since it seems to correspond to the most important applications in commercial terms.

The Transport Protocol

Multicast itself is not a connection oriented protocol and relies mostly on UDP to transmit packets in a best effort manner. This naturally has an impact on the design of multicast applications and also on security, particularly in the context of authentication as we will see. A few multimedia multicast applications rely on the Real-time Transport Protocol or RTP[SCFJ96], which adds specific information needed to identify and reconstruct the multicast data. RTP is often implemented over UDP and is coupled with a control protocol called RTCP[SCFJ96]. Some proposals have suggested multicast transport protocols to provide reliable multicast. A study of reliable multicast is beyond the scope of this thesis; see [Obr98] for a good overview. Nevertheless, it is usually admitted that these reliable protocols suffer from some scalability or efficiency drawbacks that make them unsuitable for most large scale applications in large multicast groups[Obr98]. Consequently, in this dissertation, we will always assume that the multicast transport protocol is unreliable, in order to remain as general as possible.

1.1.2 A Multicast Topology

In this work, particularly in the context of access control, we will often describe the multicast network as a tree with the following elements:

A Source: refers to the sender.

Intermediary elements: or simply *intermediaries*, refers to routers or proxies that implement a multicast routing protocol.

Leafs: refers to the set of recipients that are connected to a common IGMP local router.

These elements are common to all multicast protocols and, when we have only one source such as in SSM, it is convenient to view the source as the *root* of the tree from an algorithmic point of view.

Since a set of recipients that forms a leaf is attached to a common IGMP router, each leaf in the tree is the unique child of a single intermediary element represented by that IGMP router. In this work, we will refer to trees with this property as *singular leaf* trees.

1.2 The Security of Multicast Applications

In the previous section we outlined the basic mechanisms behind IP-Multicast. Clearly, the integrity and accuracy of multicast routing information is important for the proper functioning of IP-Multicast. Adversaries may inject bogus routing messages in the network to modify or even disrupt the routing mechanisms. However, this is true for any routing protocol the only difference being in multicast that the effects of an attack are perhaps multiplied by the multiparty nature of the protocol. There proposals have been made to secure the multicast routing protocols themselves [Moy94, Fen97] and we will not address these issues here. In this dissertation we have chosen to deal with the security of applications that are built on top of IP-Multicast, independently of any routing mechanism employed.

Many large scale commercial applications such as video/audio broadcasting or stock quote distribution could benefit from IP-Multicast mechanisms to reach many receivers in a scalable way. However, as we noted previously, these applications have been deployed on a very limited basis over the Internet. In fact most, prominent multicast applications have been confined to the enterprise level or for non-commercial uses such as IETF or NASA broadcastings[MS97]. Though some application specific issues remain, one of the biggest curb on the deployment of commercial applications may well be security. Indeed, securing multicast applications turns out to be a challenge in many situations, mainly because existing solutions that secure unicast applications cannot be extended to the multicast setting.

1.2.1 An application scenario

One of the best ways to illustrate the difficulty of securing multicast applications is to use a simple application scenario as an illustration.

Scenario Description. Consider for example a large financial news broadcaster who wants to create a business by providing financial data and market analysis to a large set of customers on the Internet through multicast. These customers are expected to pay for access to the content for variable lengths of time, ranging from less than an hour to several days, depending on their needs.

Security Issues. This financial news broadcaster is faced with two main security issues related to the distribution of its commercial content over the Internet:

Confidentiality: the content should only be accessible to the clients who payed for the service, and only for the duration corresponding to the payment.

Authentication: the content provider wants to guard itself against the risk of being impersonated by another entity who could try to generate content on its behalf, possibly harming its image and credibility.

Adapting Unicast Solutions

In unicast communications, two efficient techniques are frequently used to address these issues: symmetric encryption[BDJR97] and MACs (Message Authentications Codes)[BCK96]. To refer to these mechanisms informally, we will denote the symmetric encryption of a plaintext message M with a key K as $C \leftarrow \mathcal{E}_K(M)$ and the corresponding decryption as $M \leftarrow \mathcal{D}_K(C)$. We will write $\sigma \leftarrow \mathcal{T}_{K^{mac}}(M)$ the algorithm that takes a message M and a MAC key K^{mac} to produce a MAC tag σ and we denote $\{1, 0\} \leftarrow \mathcal{V}_{K^{mac}}(M, \sigma)$ the corresponding verification algorithm which returns 1 if $\sigma = \mathcal{T}_{K^{mac}}(M)$ and 0 otherwise. Informally speaking, it should be computationally infeasible for an adversary who sees m pairs $\{M_i, \sigma_i\}$ to generate a new pair $\{M', \sigma'\}$ which verifies $\mathcal{V}_{K^{mac}}(M', \sigma')$ such that $\{M', \sigma'\} \neq \{M_i, \sigma_i\}$ for $i = 1, ..., m$ without the knowledge of K^{mac}.

Confidentiality. If the multicast data is encrypted with a single key K common to all clients who paid for the service then it should prevent other recipients to access the data. However, when one of the clients R_j ends its subscription then we need to change K to a new value K' in order to prevent the client R_j to access the data beyond the limit of its subscription. Without loss of generality, we can assume in our scenario that each client R_i shares a private encryption key K_i with the source. If K is the only secret shared between the source and the clients, in our setting there is no simple way to exclude a single client from the client group other than sending: $\mathcal{E}_{K_i}(K')$ for all $i \neq j$. This approach is clearly unscalable in anything but relatively small groups.

Restricting access to the multicast data through the use of encryption poses another more subtle problem: containing security exposures. Indeed, in a large group there is a non negligible chance that one recipient will be compromised and that its keys will be exposed. Let us suppose for example that the key K we used above to encrypt data is exposed on a web page or in a public newsgroup; this allows anyone within the scope of the multicast group to access the data. Unlike in unicast communications, the adversary does not need to be on the link or to corrupt the routing protocol between the communication endpoints, since multicast data is forwarded automatically to him if he requests to join the group. Consequently, we need to provide mechanisms that limit the impact of such exposures.

Authentication. In the multicast setting, if we construct a Message Authentication Code or MAC[BCK96] for each packet with a common MAC key K^{mac} shared between the recipient clients and the source, it will disallow non-clients who do not possess K^{mac} to forge packets that would appear to originate from the source. Since a

MAC is computed efficiently using symmetric cryptographic techniques, each packet can be verified independently, thus, lost packets will not affect the ability to authenticate others. But this approach has one big drawback: any client can impersonate the source, by generating a message M and a valid tag $\sigma = \mathcal{T}_{K^{mac}}(M)$ from the key K^{mac}. Since the MAC key K^{mac} is common to all clients and the source, there is no way to distinguish a packet generated by the source from a packet generated by a client.

An alternative is to have a different key K_i^{mac} for each recipient R_i and append a list of the corresponding MACs to each packet as:

$$M, \sigma_1 = \mathcal{T}_{K_1^{mac}}(M), \sigma_2 = \mathcal{T}_{K_2^{mac}}(M), ..., \sigma_n = \mathcal{T}_{K_n^{mac}}(M)$$

This approach is clearly impractical in a large group since the overhead per packet will increase linearly with the group size.

A third alternative is to replace the MAC with a digital signature (Digital signatures also provide non-repudiation of origin which was not a requirement in our scenario). However, this solution also has its drawbacks because digital signatures are based on asymmetric techniques that introduce a significantly higher computational cost and a non negligible communication overhead, making this alternative impractical in most scenarios.

A Clash Between Security and Scalability

Though the previous example is somewhat simplified, it highlights some of the main difficulties associated with the design of security protocols for multicast in large groups. There seems to be an inherent clash between multicast and security. In the case of confidentiality, there is a conflict between multicast scalability mechanisms and security. Indeed, multicast relies on a single group address to identify the set of recipient rather than explicitly listing them. This anonymous identification is the primary mechanism that allows multicast to scale to virtually any group size. On the other hand, confidentiality requires us to identify explicitly the entities that send or receive data in order to provide them with the right keys to access the encrypted multicast data. In the case of authentication, the problem is not related to the group size explicitly but rather to the requirement of an efficient asymmetric mechanism to disallow receivers from impersonating the sender. Additionally, since most multicast protocols are implemented over a best effort channel, these authentication mechanisms need to tolerate losses.

1.3 Outline of this Thesis

The goal of this dissertation is precisely to analyze, discuss and suggest solutions to the two cornerstone security issues present in large scale multicast applications that our example highlighted in the previous section:

confidentiality and *authentication.*

Clearly, some applications require exclusively authentication or confidentiality, while others require a combination of both as illustrated in the example of the previous section, nonetheless from a formal point of view these two problems are mostly independent. Consequently, we have chosen to separate this dissertation in two distinct parts, which reflect these two independent but complementary issues.

The first part of the thesis deals with authentication of lossy digital streams over a large IP-Multicast group. This part is further subdivided in 3 chapters. We start in Chapter 2 with an analysis of multicast stream authentication issues and requirements dealing, in particular, with loss tolerance. We will see that solutions to this problem need to provide a subtle balance between many competing requirements that are not necessarily apparent at first. Next, in Chapter 3, we provide a general overview of existing multicast authentication proposals. Finally, in Chapter 4 we propose our own solution, which features in particular an improvement in terms of overhead per packet in comparison to previous proposals.

The second part of the thesis focuses on multicast confidentiality. As we did for multicast authentication, we start with a useful and detailed analysis of the problem and define the main requirements for multicast confidentiality in large groups in terms of both security and scalability. The remainder of this second part of the thesis is dedicated to multicast confidentiality proposals. Chapter 6 reviews major proposals for multicast confidentiality while Chapters 7 and 8 put an emphasis on our own approaches, which use untrusted intermediary elements in the network.

The final chapter of this dissertation serves as conclusion and provides a perspective for future work in multicast security.

1.4 Contributions

The contribution of our work is twofold. First, we provide a deeper analysis of the issues and requirements related to multicast authentication and confidentiality than previously found in related work. In particular, we propose a new security requirement for multicast confidentiality in Chapter 5: *containment*. Second, we provide our own original solutions to the described multicast security issues. The multicast authentication algorithm we propose has a lower overhead than any previously proposed scheme we know of, and our two multicast confidentiality algorithms are designed to answer a larger set of requirements than previously proposed schemes.

Part I

Authentication

Chapter 2

Definitions and Requirements

2.1 Background

Authentication protocols provide mechanisms that allow an entity to ascertain the origin of a piece of information it received. In this dissertation we are interested in authentication issues related specifically to multicast streams and consequently to multicast applications, such as the continuous delivery of stock quotes or video/audio streams for example, to a large group of recipients. In this context, the stream is not viewed as a single string of bits that we wish to authenticate once it is fully received but rather as a potentially infinite sequence of consecutive chunks of data that we wish to authenticate individually as soon as they are received, and in most situations before the application processes them. Consequently multicast authentication deals with the continuous authentication of broadcasted data by a large group of recipients.

A lot of requirements further affect how we deal with this problem. Many multicast streams are delivered through an unreliable protocol such as RTP/UDP[SCFJ96], and many multicast multimedia applications are designed to tolerate partial data loss with a graceful degradation in playback quality. Consequently, one of the primary requirements of a stream authentication protocol is the ability to authenticate received packets amid losses in the network. In some cases, the data itself is prerecorded in advance, which allows the sender to compute authentication information in advance. But the data may need to be broadcasted in real time and in that case authentication information also needs to be computed in real time. The group of receivers may change or remain constant during the broadcast. The sender and receivers may have different computational powers or storage capacities. As we will show, all these elements influence the design of a stream authentication scheme.

We distinguish the *source* of the broadcast, which produces the authenticated packet stream, while we call *recipients* the entities which need to authenticate the received packets. We are interested in two types of authentication mechanisms:

- **Source authentication:** provides a mechanism to convince the recipients that the received data was generated by a specific source.
- **Nonrepudiation (of origin):** provides a mechanism that allows a recipient to prove to a third party that the received data was generated by a specific

	signatures/sec.	verifications/sec.
1024 bits	74	1991
2048 bits	12	715

Table 2.1: RSA signatures/verifications performance on 600 Mhz Pentium III

source.

Naturally nonrepudiation implies source authentication. Note that nonrepudiation is by definition a multiparty paradigm: only one entity can generate the authentication information but any third party can verify it.

While unicast source authentication can rely on efficient symmetric cryptographic techniques such as MACs (Message Authentication Codes), such mechanisms cannot be applied in a straightforward manner to the multicast setting. Indeed, as we highlighted in the introduction of this thesis, if both the source and the recipients share a common MAC key, any recipient can masquerade as the source, and if we use a different key for each source-recipient pair then we run immediately into scalability issues.

The inadequacy of simple unicast MACs in the multicast setting raises the question of the feasibility of constructing a multiparty version of a MAC or MMAC. This MMAC could be generated by only one entity but verified efficiently by an large or unlimited number of recipients. As we will see in the next chapter, there has been some attempts to provide such a mechanism in a limited way. In their work[CGI+99] Canetti et al. created a rather efficient MMAC that can be verified by all recipients but only resists the collusion of less than w recipients. The advantage of their algorithm decreases as w grows. Recently, Boneh et al. [BDF01] studied that question essentially from a theoretical point of view. They have shown that basically, if an efficient MMAC scheme can be produced, it can be turned into an efficient digital signature scheme. Digital signature are based on asymmetric cryptographic techniques, which are several orders of magnitude less efficient than symmetric techniques such as a MAC. Consequently the design of an efficient MMAC does not seem possible without a major advance in digital signatures.

Extending Asymmetric Cryptography.

Despite the fact that nonrepudiation is not needed in all scenarios the inadequacy of purely symmetric techniques seems to suggest that asymmetric cryptographic techniques are needed for multicast authentication even to achieve simple source authentication. Consequently, a straightforward idea to provide multicast authentication is to let the source sign individual packets in the stream. However, the cost of digital signatures make this scheme impractical. Table 2.1 shows the performance of an RSA signature and verification algorithm using the OpenSSL toolkit, `http://www.opensll.org/` on a 600 Mhz Pentium III. The reason why the verification is much faster than the generation is because a small practical public exponent e is used such as $e = 65537$ ($e = 3$ or $e = 17$ are usually not recommended [Bon99]).

In our example, these operations take 100% of CPU time but a recipient in a multimedia application is likely to dedicate most of its CPU power to data processing (for example the decoding and playback of the received video/audio). For live broadcastings which would require real time cryptographic operations on individual data packets of the stream, the signature rate is a true bottleneck. Also, for low computational power recipients, such as mobile or embedded devices, these figures are likely to be much lower, making this approach impractical without dedicated hardware.

The second and perhaps more critical problem of this approach is its overhead per packet, which amounts to at least 1024 bits (128 bytes) for a secure RSA signature. These bytes need to be added to every packet of data which is to be compared for example to the RTP/UDP/IP overhead per packet, which is only 20 bytes.

We conclude that it is often not possible to use digital signatures as a drop-in replacements for MACs to provide multicast authentication. However as we will see, multicast authentication can be achieved by using an asymmetric mechanism *in combination* with other more efficient techniques with reasonable tradeoffs. The general paradigm is to use a digital signature mechanism as a commitment and use other techniques to extend this commitment to as much data as possible. In that sense, we can consider multicast authentication techniques as the art of extending asymmetric cryptographic techniques.

Offline and Online Authentication.

An important distinction that influences the design and requirements that affect a stream authentication scheme is whether the authentication scheme is targeted for an online or an offline stream. *Offline* or *pre-recorded* streams refer to streams that are known in advance to the sender, while *online* or *live* streams refer to streams that are not known in advance to the sender.

Offline streams: Consider the case of a *pre-recorded* film stream to be multicasted to a group. In such a case cryptographic authentication information can be computed before the broadcast without real-time constraints, based on the whole set of packets in the stream. The cryptographic authentication information can then be inserted in the stream. During the broadcast to the group, the source does not need to make further cryptographic computations.

Online streams: On the other hand consider the delivery of *live* data, such as the broadcast of stock quotes or the retransmission of an important live sports event. In this case, authentication information needs to be computed "on the fly" with the least possible buffering of packets at the source level. Cryptographic information needs to be efficiently computed in real time by the source and embedded in the stream as it is produced.

Naturally, if we have an efficient algorithm that provides *online* or *live* authentication, it can also be used for *offline* or *pre-recorded* data. This motivates the search of a good online scheme as a general solution to stream authentication.

2.2 Robustness Requirements

In this dissertation we focus on stream authentication schemes that can authenticate received packets despite losses in the network. We believe that this covers most if not all multicast stream applications. For non-lossy stream authentication, see for example the work of Gennaro and Rohatgi[GR97].

2.2.1 Packet Losses in the Internet

One of the primary parameters of multicast stream authentication is its loss tolerance or *robustness*, which describes how well the scheme adapts to losses in the network.

Ideally we would like to have a scheme that offers perfect robustness: a scheme that tolerates an arbitrary number of losses while retaining the capability to authenticate all received packets. But in most realistic situations we do not need such a strong notion of robustness if we consider Internet loss patterns as opposed to purely random losses. This idea was first introduced in a stream authentication scheme by Golle and Modadugu[GM01] and later reused in other schemes [PCTS00, MS01]. Based on the work of Paxson [Pax99] who analyzed TCP/IP traffic on a large scale and showed that losses often occur in bursts, Golle and Modadugu designed a stream authentication algorithm that was targeted at bursty loss patterns. We will see that such an hypothesis allows us in many cases to design more efficient stream authentication schemes.

A Markovian Model For Internet Losses

There has been quite a few studies about Internet loss patterns for applications such as Audio Unicast/Multicast [BC93], Internet Telephony[BFPT99], Multicast [YKT96a, YKT96b], TCP[Pax99] TCP/UDP[BSUB98]. These studies differ on their analysis and in their scope, however there is a general consensus among most studies that:

1. Packet losses are not independent. When a packet is lost the probability that the next packet will be lost increases, which means that losses in the Internet are often *bursty*.

2. However the majority of bursts are small (often less than 6-7 packets).

3. There are some rare long bursts, lasting up to a few seconds (In [BSUB98] the authors suggest that these bursts could attributed to network disruption or maintenance).

Many of these studies have suggested that Internet loss patterns can be satisfactorily modeled with a k order Markov chain (a chain with 2^k states). In this thesis we propose to refer to a model often suggested to describe the bursty loss nature of Internet traffic which is a simple 2 state Markov chain [BFPT99, YMKT99] also called the Gilbert model. This model has also been used in the EMSS[PCTS00] stream authentication scheme proposed by Perrig et al. In this two state model, state 0 represents a packet received and state 1 a packet lost by the recipient. We

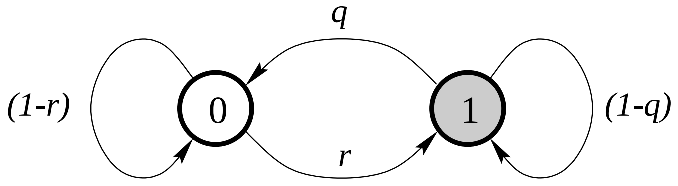

Figure 2.1: The Gilbert Model

denote $r \neq 0$ the probability of going from state 0 to state 1 and $q \neq 0$ the probability of going from state 1 to state 0, as illustrated on figure 2.1. Intuitively, we see that if q is small then the chain has a greater probability of staying in state 1 once it reaches that state, which corresponds to a burst. This model is conveniently represented[GS00] by its transition matrix $M = [m_{ij}]$ as:

$$M = \begin{bmatrix} (1-r) & r \\ q & (1-q) \end{bmatrix}$$

This type of discrete Markov chain is *ergodic* and we can compute the mean residence time w_i of each state i by solving the equation $(w_0, w_1).M = (w_0, w_1)$ and using the fact that $w_0 + w_1 = 1$. The mean residence time of state 1, w_1 corresponds to the *average packet loss rate*, and here we have:

$$w_0 = \tfrac{q}{r+q}, \quad w_1 = \tfrac{r}{r+q}$$

When the chain is in state 1 then the probability of leaving that state is q. Consequently, taking an analogy with a sequence of Bernoulli trials, the probability of leaving state 1 after being in that state for k consecutive steps is $q.(1-q)^{k-1}$, which describes a geometric distribution of mean

$$\mu_1 = \frac{1}{q}$$

and which is called the mean *sojourn time* of state 1, or in the context of packet loss modeling: *the average loss burst length.*

We have shown how to compute both w_1 the average packet loss rate and μ_1 the average loss burst length from the corresponding 2 state Markov model. Reciprocally, it is possible to compute the Markov chain parameters (r, q) from the average loss rate w_1 and the average loss burst length μ_1:

$$q = \tfrac{1}{\mu_1}, \quad r = \tfrac{w_1}{(1-w_1)\mu_1}$$

2.2.2 Case Studies

To illustrate the 2 state Markov chain model above, we propose to recall two case studies from EMSS, the stream authentication scheme proposed by PERRIG ET AL.[PCTS00]. While the EMSS scheme will be described more in detail in section 3.4.3, we will reproduce the two case studies used in their work here, both as an illustration of the model and as a useful reference for further comparisons between some stream authentication algorithms in Chapter 3:

CASE 1: The traffic information sensors.

A municipality wishes to provide traffic information from traffic sensors distributed over the streets. The system requirements are as follows:

- The data rate of the stream is about 10 Kbps, about 20 packets of 64 bytes each are sent every second.
- The packet drop rate is at most 5% for some recipients, where the average length of burst drops is 5 packets.
- The verification delay should be less than 10 seconds.

Given the drop rate of 0.05 and the average length of bursts of 5, we have a corresponding 2 state Markov chain with $r = 0.010526$, $q = 0.2$.

CASE 2: The real time video broadcast.

The second case study proposed by Perrig et al. is related to real-time video broadcasting, with the following requirements:

- The data rate of the stream is about 2Mbps, or 512 packets of 512 bytes each every second.
- The packet drop rate is at most 60% for some recipients, with an average length of bursts of 10 packets.
- The verification delay should be less than 1 second.

Here we have a much higher drop rate of 0.6 and the average length of bursts is 10, which gives us a corresponding 2 state Markov chain with $r = 0.15$ and $q = 0.1$.

2.3 Other Requirements

Though robustness is an important requirement of a multicast stream authentication scheme, there are quite a few other important additional requirements, which we define here:

- Joinability
- Low authentication latency
- Limited buffering
- Limited cost

Joinability

Since a stream is a potentially very long (or infinite) sequence of packets, we need to consider that in many situations, the recipient will only want to authenticate part of the stream. The group of recipients will often change during the broadcast. We define the *joinability* of an authentication scheme as its ability to allow recipients to start verifying packets at a random point in the stream. Ideally we would like the authentication scheme to be *joinable* on every packet of the stream, in practice we believe it is sufficient for an authentication scheme if it is *joinable* on a reasonable boundary, or every n packets.

Latency

Multicast streams need to be authenticated continuously. Ideally, we would like to authenticate packets or application data units individually as soon as they are received. In practice, a packet may be authenticated by a the next one or even by another future packet, which introduces a delay that we define as the authentication latency. In some cases we will refer to the *maximum authentication latency*, which defines the number of packets that need to be transmitted before we can authenticate a packet *in the worst case*.

For many applications a small authentication delay is not a problem. Consider for example, a live video broadcast, in such a situation it is conceivable to have an authentication delay of about a second, if that small tradeoff allows us to make the authentication more efficient. Note also that Internet streaming client applications (for example, realplayer `http://www.real.com/`) natively include a small client side buffering scheme to adapt to irregularities of Internet traffic.

(Server Side) Buffering

Some stream authentication algorithms introduce dependencies between the data packets that require the sender to buffer a few packets before sending them. This sender side buffering is mainly relevant to online authentication schemes. Symmetrically to latency, for quite a few applications it is acceptable to introduce a small buffering, however if the buffer gets too large then the scheme may not be considered truly "online" anymore.

Cost

The final criteria that distinguishes a multicast authentication scheme from another is its cost in terms of computational requirements and in terms of overhead in bytes per packet of application data. First, if a scheme is too computationally intensive then it may not be usable in practice because it consumes computer cycles that would otherwise be used by the application itself (such as video/audio codec processing). Second, if the scheme has too much overhead, then it may significantly reduce the network bandwidth available to the stream and create congestion problems that further downgrade the quality of the stream.

Conclusion

Authentication in the context of multicast can be summarized as double a problem:

- *A multiparty problem:* one or few entities can generate the data, many entities need to verity it. This aspects forces us to use asymmetric mechanisms.
- *A streaming problem:* a set of packets need to be authenticated sequentially on a lossy channel. This aspects defines a set of constrains that affect the asymmetric mechanism we use.

Ideally we seek a *robust online* stream authentication scheme that is *joinable* on every packet, has *no buffering* as well as no *latency* and all that for a *cost* of the same order magnitude as a MAC without any additional requirement. Unfortunately, as we said at the beginning of this chapter, such as scheme does not exist. Consequently, to provide an authentication scheme with a reasonable cost it is often necessary to find the best compromise between the list of requirements we defined above.

In the next chapter we review prior proposals for robust stream authentication, most of which are targeted for online authentication. We will highlight how most practical solutions use different techniques to *extend an asymmetric cryptographic commitment* from one to many packets, using hash chaining techniques as in section 3.4 or time constrains as in section 3.3. The following chapter is also meant to allow us to compare our own scheme, presented next in a separate chapter, with other related proposals.

Chapter 3

An Overview of Authentication Algorithms

3.1 Multicast MACs

Let $\sigma \leftarrow MAC_K(M)$ denote the computation of an authentication tag σ with a MAC (Message Authentication Code) algorithm such as HMAC[BCK96], keyed with K. The security of a MAC scheme is usually defined as its resistance against existential forgery in an adaptive chosen plaintext scenario, where an adversary A interacts with a legitimate party $\mathcal{O}$ who returns the MAC σ_i of any message M_i that A asks for. Naturally a query M_i may depends on previous queries and replies that were produced. The key K is only known to $\mathcal{O}$ and is chosen randomly before any interaction with A. An adversary A is considered successful if after querying $\mathcal{O}$ on n messages $\{M_1, ..., M_n\}$ and obtaining the corresponding tags $\{\sigma_1, ..., \sigma_n\}$, he is capable of producing a new pair $\{\sigma_{n+1}, M_{n+1}\}$ without interacting with $\mathcal{O}$ such that $\sigma_{n+1} = MAC_K(M_{n+1})$ where $M_{n+1} \notin \{M_1, ..., M_n\}$. A MAC is considered (n, t, q') secure[1] if the probability that any adversary, asking at most n queries to $\mathcal{O}$ and spending at most t units of time, succeeds with probability less than q'. In practice, a MAC is secure if q' remains negligible for computationally tractable parameters t and n.

3.1.1 Basic Scheme

As already briefly mentioned in the the previous chapter, Canetti et al. constructed a MMAC (Multicast Message Authentication Code) scheme that provides security for a recipient against a coalition of up to w recipients. Their scheme draws from ideas from *set intersection* schemes [DFFT95] and proceeds as follows:

Setup.

- The source knows a set of l MAC keys, $R = \{K_1, ..., K_l\}$.

[1] A weaker alternative sometimes considered is selective forgery, where the message M' is chosen *before* the game starts, for a more formal analysis see for example, [BDF01].

- Each recipient i gets a subset R_i of $\frac{l}{(w+1)}$ keys chosen randomly from R.

Authentication.

To authenticate a multicast message M we proceed as follows:

- The source multicasts $(M, \sigma_1, \sigma_2, ..., \sigma_l)$ where $\sigma_i = MAC_{K_i}(M)$.
- Each recipient i verifies all the MACs that correspond to the subset R_i of keys it holds. If at least one MAC is not good then the message is considered corrupt.

The notion of security that is used here is somewhat different from the one we described above for the MAC. The authors define a MAC scheme to be "q-secure per message" if an adversary who interacts with an oracle $\mathcal{O}$ as above has *a small but not negligible* probability q of guessing a good pair $\{\sigma_{n+1}, M_{n+1}\}$, but still has a negligible probability of knowing that the guess was correct. In this context, they suggest to choose $q = 10^{-3}$, which they argue is sufficient for many applications though not all.

Main Result.

Let q' define the probability of computing the output of a MAC without knowing the key. In their work, Canetti et al. show that, given a recipient u, if we have a total number of keys $l = \mathbf{e}.w.\ln(1/q)$, then the probability that a coalition of corrupt users can forge a message for u is at most $q + q'$.

3.1.2 Extensions

With a classical MAC algorithm, q' will be negligible compared to q. Consequently, it is possible to reduce the MAC output size in bits such that the security of the MAC decreases down to $q' = q$ without compromising the overall security of the algorithm by a significant factor.

Canetti et al. show that further increasing the number l of keys held by the source further increases the security of the algorithm. Reciprocally it allows to further decrease the size of the MAC output to 1 bit, trading in more MAC computations for a lower communication overhead. In this lower communication overhead scheme, the probability that a coalition of w recipients can forge a message for a recipient u is $2.q$, with a total number of keys $l = 4.\mathbf{e}.w.\ln(1/q)$ and still $l/(w+1)$ keys for each individual user.

3.1.3 Discussion

An important aspect of the security bounds that are given for the algorithms in [CGI+99] is that they apply to a coalition of w recipients who want to forge a message for *a chosen recipient* u. However, these results do not tell us what is the probability that the coalition can forge a message for *any recipient* in the group. Canetti et al. acknowledge (footnote 5 in [CGI+99]) that a scheme such that

	Auth. per sec.	Verif. per sec.	Overhead in bits	Source Key	Receiver Key
basic scheme, $w = 10$, $q = 10^{-3}$	2650	26 500	1900	190 (10 bit MAC) keys	19 (10 bit MAC) keys
1 bit scheme $w = 10$, $q = 10^{-3}$	660	6 600	760	760 (1 bit MAC) keys	76 (1 bit MAC) keys
any recipient security $N = 10^4$, $q' = 10^{-3}$	200	2 000	25 000	2500 (10 bit MAC) keys	250 (10 bit MAC) keys
RSA, 1024 bits	50	30 000	1024	2048 bits	1024 bits

Table 3.1: A performance comparison for MMAC schemes.

no coalition of w recipients can cover the set of keys of any user would be more expensive, with a total number of keys reaching $O(w^2 \ln(N))$ and an number of keys for individual recipients reaching $O(w \ln(N))$, where N denotes the number of recipients in the group.

To provide a more concrete evaluation of the cost and overhead of the scheme, we reproduce in table 3.1 a numeric comparison of concrete instantiations of their scheme along with an RSA signature also reproduced from [CGI+99]. The basic scheme is the first scheme we described above with MAC outputs truncated to 10 bits to match the security q' of the MAC with q. The one bit scheme is much more efficient in terms of communication overhead, with only 760 bits. The third line of this table illustrates the remark we made above about the security of the schemes: if we want to make sure that no coalition of w recipients can forge a MAC for any recipient than the scheme becomes clearly impractical.

3.1.4 Conclusion

The most important aspects of this scheme are summarized in the frame below. The MMAC scheme provides a Multicast MAC which offers satisfactory security for a chosen recipient against a coalition of at most w recipients. MMAC has the same advantage as a MAC in terms of robustness, joinability, latency and buffering.

The scheme is computationally much more efficient than a digital signature, however, for a coalition resistance factor $w = 10$ it still has an overhead equivalent to a signature even in the best scheme based one a one bit MAC. Since this overhead is proportional to w, higher values of w will make the scheme impractical in many scenarios. Yet, the overhead is not proportional to the group size but only to the coalition size.

However, the security of this scheme is defined in terms of a coalition of recipients trying to forge a message for a chosen recipient. If we want to achieve the same level of security against forgeries for any recipients, the overhead of the scheme increases to clearly impractical levels as shown in table 3.1, with an overhead which this time increases logarithmically with the recipient group size.

In his work on lower bounds for multicast message authentication[BDF01], Boneh et al. suggest that the results of [CGI$^+$99] are optimal, within classical definitions that are applied to a MAC. It is thus unlikely that a better and more practical MMAC scheme can be constructed without a major advance in cryptography.

MMAC scheme summary

Scheme type: Online source authentication of lossy streams.

Advantages: The MMAC is an online authentication scheme with all the good properties of a MAC scheme:

- **Robustness:** perfect robustness.
- **Joinability:** authentication can begin at any packet in the the stream.
- **Latency & Buffering:** none.

Drawbacks: The two main drawbacks are:

- **Overhead**: an overhead comparable to a signature even with the 1 bit MAC scheme.
- **Security:** limitations for the most practical schemes.

3.2 Faster Signatures

If digital signatures were faster and smaller they would provide an interesting solution for stream authentication, allowing us to sign packets individually, providing unlimited robustness and the lowest possible latency. So quite logically, one of the directions that has been explored to provide multicast authentication is to develop more efficient signatures.

3.2.1 Extended Fiege-Fiat-Shamir.

The Fiege-Fiat-Shamir signature scheme is a signature scheme derived from the Fiat-Shamir zero knowledge identification scheme (any zero knowledge identification scheme can be converted to a signature scheme [MvOV96]). Let .||. denote concatenation and let H denote a cryptographic hash function such as MD5[Riv92]. The Fiege-Fiat-Shamir scheme is defined by three algorithms $\{\mathcal{K}, \mathcal{S}, \mathcal{V}\}$, parameterized by (k, t) as follows:

- $\mathcal{K}(t, k)$: the key generation algorithm.
  - Generate 2 random distinct primes p, q and let $n = p.q$
  - Choose k integers in $\mathbb{Z}_n^*$: $\{s_1, ..., s_k\}$.
  - for $i = 0, ..., k$ do $v_i \leftarrow s_i^{-2} mod\, n$.

  - Output the public key $pk = \{v_1, ..., v_k; n\}$ and the secret key $sk = \{s_1, ..., s_k\}$

- $\mathcal{S}_{sk}(m)$: the signature algorithm.

  - Generate t random integers in $\mathbb{Z}_n^*$: $\{r_1, ..., r_t\}$
  - for $i = 1, ..., t$ do $x_i \leftarrow (r_i)^2 mod\, n$
  - let b define the $k.t$ first bits of $H(m||x_1||...||x_t)$
  - Split b as $b_{i,j} \in \{0, 1\}$ where $i = 1, ..., t$ and $j = 1, ..., k$
  - for $i = 1, ..., t$ do $y_i = r_i.(s_1^{b_{i,1}}.s_2^{b_{i,2}}...s_k^{b_{i,k}}) mod\, n$ (1)
  - The signature of m is $\sigma = \{\{y_1, ..., y_t\}, b\}$.

- $\mathcal{V}_{pk}(\sigma, m)$: the verification algorithm.

  - parse σ as $\{\{y_1, ..., y_t\}, b\}$
  - Split b as $b_{i,j} \in \{0, 1\}$ where $i = 1, ..., t$ and $j = 1, ..., k$
  - for $i = 1, ..., t$ do $z_i \leftarrow y_i^2.(v_1^{b_{i,1}}.v_2^{b_{i,2}}...v_k^{b_{i,k}}) mod\, n$
  - if the $k.t$ first bits of of $H(m||z_1||...||z_t)$ are equal to b then the signature is valid.

In their work on stream authentication with hash trees[WL99], Wong and Lam propose two types of extensions to the FSS scheme: speedups and adjustable verification and they call this extended scheme, *eFSS*. The Speedups try to make the FFS scheme more efficient comparatively to a traditional RSA signature, in terms of signature and verification rate per second. Adjustable verification allows receivers with different computational power to trade in security for speed if needed: receivers with more power will have greater security while lower power receivers will be able to cope with high packet rates with lower security.

Speedups

- A known improvement[MvOV96] to the FFS signature scheme is to select $v_1, ..., v_k$ as small primes that can be represented with two bytes. This shortens the public key and increases the efficiency of the verification scheme.

- As in efficient versions of RSA[RSA99], we can use the Chinese Remainder Theorem to improve the operations on line (1) above, by computing the y_i in $\mathbb{Z}_q^*$ and $\mathbb{Z}_p^*$ and combining the results to get y_i.

- If t is expected to remain low we can also do some pre-computations for the operations on line (1) above. Since $b_{i,j}$ only takes values in $\{0, 1\}$ then the product $(s_1^{b_{i,1}}.s_2^{b_{i,2}}...s_k^{b_{i,k}})$ can take all possible multiplicative combinations of elements in $\{s_1, ..., s_k\}$ and store them in memory for later use. For example, for $t = 4$ we need to store 480 values.

	signature size in bytes
RSA	128
eFFS(32,4)	912
eFSS(64,2)	400
eFSS(128,1)	144

Table 3.2: RSA 1024 and eFFS($k.t = 128$) overhead.

	signature rate	verification rate
RSA	193	5290
DSA	874	609
ElGamal	749	45
eFSS(128,1)	1610	5250

Table 3.3: Signing/verification rates (approx.) 1024 bit signature schemes in packets per second.

Adjustable Verification

The idea here is to use t greater than 1 to provide a selectable level of security for the signature. The scheme above is modified by appending to the original signature $\{\{y_1, ..., y_t\}, b\}$ the values $\{x_2, ..., x_t\}$ which gives us $\sigma = [\{y_1, ..., y_t\}, b, \{x_2, ..., x_t\}]$. The verifier can compute a verification of security level $l \in [1, ..., t]$ as follows:

- for $i = 1, ..., l$ do $z_i \leftarrow y_i^2.(v_1^{b_{i,1}}.v_2^{b_{i,2}}...v_k^{b_{i,k}}) mod\, n$
- if the $(k.t)$ first bits of of $H(m||z_1||x_2||...||x_t)$ are equal to b, and $(z_2, ..., z_l) = (x_2, ..., x_l)$ then the signature is valid.

The scheme can be further iterated for a receiver to increase the level of security from l to a higher value l' with a similar mechanism; see [WGL98] for details. The signature size for a (k, t) FFS signature and thus the overhead per packet is $kt + (2t - 1) \times |n|$ where $|n|$ denotes the size of n. In their work [WGL98], Wong and Lam use $k.t = 128$ and $n = 1024$ as a point of comparison with 1024 bit RSA[RSA78, RSA99]. We extend the tables presented in their work, with a few sample signature sizes in table 3.2. In particular we show values for $t > 1$ that were omitted in [WGL98]. This table is interesting because it shows that adjustable verification has a big cost in terms of overhead. Thus adapting the FFS scheme to low power devices costs much more bandwidth !

Table 3.3 reproduces some performance results form [WGL98] in terms of number of packets signed and verified per second on a Pentium II 300 PC running Linux. It shows much improved signature rates compared to other signature schemes.

Conclusion

First, we need to point out that the eFFS scheme was not designed to be used to sign each packet, but rather as a complement to the Hash tree construction will will review in 3.4. Clearly, despite certain improvements, it is not a practical solution for signing packets individually. Nevertheless, the eFFS scheme allows improved signature generation speed. It also offers adjustable security for low power devices but this comes at the cost of a strong increase in terms of overhead.

3.2.2 *k-time* Signatures

Confronted with the shortcomings in speed of traditional digital signatures, ROHATGI proposed in [Roh99] to use the offline/online digital signature paradigm[EGM96] in the multicast setting. The basic idea of this approach is to perform expensive computations with a traditional digital signature offline, even before the stream is known, to produce a certificate for a much more efficient one-time signature that will be computed online during the broadcast. Indeed, the online part of the signature is basically computed using hash functions which are several orders of magnitude faster than digital signatures. However, offline/online signatures have not been in widespread use because they tradeoff online speed for an significant increase in overhead, even greater than a traditional digital signature. Consequently to make these signatures more practical in the multicast setting ROHATGI uses several tricks to reduce this overhead (some of these ideas come from previous work by the same author and GENNARO on non-lossy streams[GR97]).

To outline the basic idea of this work consider a sender who wishes to sign an 80 bit string Y. Let H define a cryptographic hash function, where H^k denotes k successive applications of H, that is $H^k(x) = H(H^{k-1}(x))$ where $H^0(x) = x$. Also, let $\{\mathcal{S}, \mathcal{V}\}$ informally define a "traditional" digital signature and verification algorithm associated to the sender.

Sender task

- Offline, the sender will compute:
  - 23 random values : $r_1, ..., r_{23}$
  - let $h_1 \leftarrow H^{15}(r_1), h_2 \leftarrow H^{15}(r_2), ..., h_{22} \leftarrow H^{15}(r_{22})$ and $h_{23} \leftarrow H(r_{23})$
  - $\sigma \leftarrow \mathcal{S}(H(h_1, ..., h_{22}, h_{23}))$
- Then he publishes $\{\sigma, h_1, ..., h_{23}\}$.
- Online, to sign a 80 bit value Y, the sender proceeds as follows:
  - split Y in 20 chunks of 4 bits $y_1, ..., y_{20}$.
  - for $i = 1, ..., 20$ do $t_i \leftarrow H^{(15-y_i)}(r_i)$
  - let $X = \sum_{i=1}^{i=20} y_i$ that we split in 2 chunks of 4 bits $\{x_1, x_2\}$ and 1 chunk of one bit $\{x_3\}$, such that $X = x_1 + 16.x_2 + 64.x_3$.
  - compute $t_{21} \leftarrow H^{x_1}(r_{21}), t_{22} \leftarrow H^{x_2}(r_{22}), t_{23} \leftarrow H^{x_3}(r_{23})$.
  - Send $Y, \{t_1, ..., t_{23}\}$

Receiver task

- Offline, the receiver can verify the public key signature σ on $\{h_1, ..., h_{23}\}$.
- Online, when the receiver gets $Y, \{t_1, ..., t_2\}$ he proceeds as follows:
  - split Y in 20 chunks of 4 bits $y_1, ..., y_{20}$.
  - let $X = \sum_{i=1}^{i=20} y_i$ that we split in 2 chunks of 4 bits $\{x_1, x_2\}$ and 1 chunk of one bit $\{x_3\}$, such that $X = x_1 + 16.x_2 + 64.x_3$.
  - for $i = 0, ..., 20$: verify that $h_i = H^{y_i}(t_i)$.
  - verify that $h_{21} = H^{15-x_1}(t_{21})$, $h_{22} = H^{15-x_1}(t_{22})$, $h_{23} = H^{1-x_3}(t_{23})$.
  - If all previous verifications are successful then accept Y as valid.

Extensions

Of course the above two mechanisms are not very practical on their own. In his work, ROHATGI uses several other mechanisms to greatly extend the usability of the scheme, the two most important ones are:

- **TCR hash function:** Use keyed hash functions to reduce the security requirement to *target collision resistance* (TCR) instead of *strong collision resistance.* Birthday attacks do not apply to TCR hash function, consequently, the hash length can be reduced from 128/160 bits down to 80 bits. Of course some additional mechanism is provided to transmit the key of the hash function in an efficient way. The use of 80 bit TCR hash functions also allows the above scheme to be applied to a data packet, by using it to sign the corresponding 80 bit hash.

- **Sign n^2 k-time signatures:** In the above scheme there is one public key signature for one online signature. In the scheme proposed by ROHATGI, a public scheme is used to sign n^2 offline signatures using a MERKLE authentication tree [Mer89] of degree n. This only requires 1 public key operation for n^2 packets and the authentication tree is pre-computed offline.

According to some simulations performed in [Roh99], 500 to 1000 signature per second can be generated this way on a workstation class computer. The overhead includes the transmission of the offline public key certificate and the online signature for each packet and amounts to approximately 300 bytes per packet.

Conclusion

Despite the potential gain in speed in terms of signature generation and the obvious advantage of signing each packet individually in terms of robustness, buffering, latency and joinability, these signature still require a large overhead which makes them impractical in many situations.

3.2.3 Faster Signature Summary

The sections presented two improved signature schemes, yet none of them are satisfactory in terms of overhead to sign packets individually. Though perhaps elliptic curve cryptography and new signature scheme such as NTRU [HPS98] may open the path to new performance and overhead improvements, signing each packet seems impractical today.

Nevertheless, these signatures may be an interesting alternative to RSA in the hybrid approaches we review in section 3.4.

Faster Signature Summary

Scheme type: Online source authentication of lossy streams.

Advantages:

- **Robustness:** perfect robustness.
- **Joinability:** authentication can begin at any packet in the the stream.
- **Latency & Buffering:** none.

Main Drawback: the main drawback is a high communication **overhead**. In fact, both the adjustable version of the *eFFS* and the *k-time* signature scheme have an even bigger overhead than traditional signatures.

3.3 Time Committed Authentication

Time Committed Authentication refers to a specific scheme proposed by PERRIG ET AL. called TESLA[PCTS00]. Here, a signature scheme is used to sign a commitment and the asymmetry of the signature is propagated by using a time based verification condition. Note that TESLA does not aim to provide non-repudiation, only message authentication. This however sufficient for many applications.

3.3.1 Basic Scheme

Let F and F' define 2 distinct pseudo-random functions[GGM84], where F^k denotes k successive applications of F, that is $F^k(y) = F(F^{k-1}(y))$ and $F^0(y) = y$. From a random seed x, we derive a pseudo-random sequence of n keys $\{K_1, ..., K_n\}$ as follows:

- $K_n = x$
- $K_i = F^{n-i}(x)$ for $i < n$

The main property of this sequence is that given K_i and F alone it is computationally infeasible to compute K_{i+1} assuming F is pseudo-random. However, given K_j such that $j > i$ it is straightforward to compute K_i form K_j since $K_i = F^{(j-i)}(K_j)$.

Sender Tasks: Let $K_i' = F'(K_i)$. Consider now a sender who wants to send authenticated data packets $\{M_1, M_2, ...\}$ to a group of recipients. The sender produces packets $\{P_0, P_1, ..., P_n\}$ that are transmitted individually every unit of time t_i as follows:

- (*time* t_0): send $P_0 = Sign(F(K_1))$, where $Sign(F(K_1))$ is the sender' s digital signature on $F(K_1)$.
- (*time* t_i): send $P_i = \langle M_i || K_{(i-1)}, \tau_i \rangle$, where $\tau_i = MAC_{K_i'}(M_i || K_{(i-1)})$.

Receiver Tasks: The verification of a packet P_i follows three rules:

1. (*Chaining*) Check that $F(K_{(i-1)}) = K_{(i-2)}$ (or if $K_{(i-2)}$ is missing apply F until we reach a known value in the chain).
2. (*MAC*) Check that $\tau_i = MAC_{K_i'}(M_i || K_{(i-1)})$.
3. (*Time*) Check that P_i was received before time $t_{(i+1)}$.

The main originality of TESLA resides in the third rule, which is time based. Note that the second rule cannot not be applied before we receive $P_{(i+1)} = \langle M_{i+1} || K_i, \tau_{i+1} \rangle$, consequently the verification of a packet introduces a latency of one packet for the receiver.

We propose a brief illustration of the TESLA scheme on figure 3.1.

Security

Informally, consider an adversary who wants to forge packets in this scheme. Since the adversary is potentially one of the receivers we will assume that he knows both pseudo-random functions F and F'.

First we observe in the first rule that the receiver always checks that the key K_i is part of the one way hash chain, the first node of which is signed by the source. Consequently, unless there is a flaw in F or in the original digital signature scheme in packet P_0, the adversary cannot forge K_i. Neither can he change K_i' since it is also derived from K_i through a one way function.

Second, if an adversary wants to forge packet P_i, unless there is a flaw in the MAC, he needs to know K_i' to produce a valid tag τ_i that passes the second rule. To obtain K_i' he needs to wait for packet $P_{(i+1)}$ which reveals K_i needed to derive $K_i' = F'(K_i)$. Since packet $P_{(i+1)}$ is transmitted at time $t_{(i+1)}$, the adversary will only be able to produce a forged packet P_i at a time greater or equal to $t_{(i+1)}$. But this attack will fail because of the third rule which does not allow a packet to be accepted after its MAC key is revealed.

Consequently, these three rules seem sufficient to guaranty the security of this scheme. For a more formal security analysis, we refer the reader to [PCTS00].

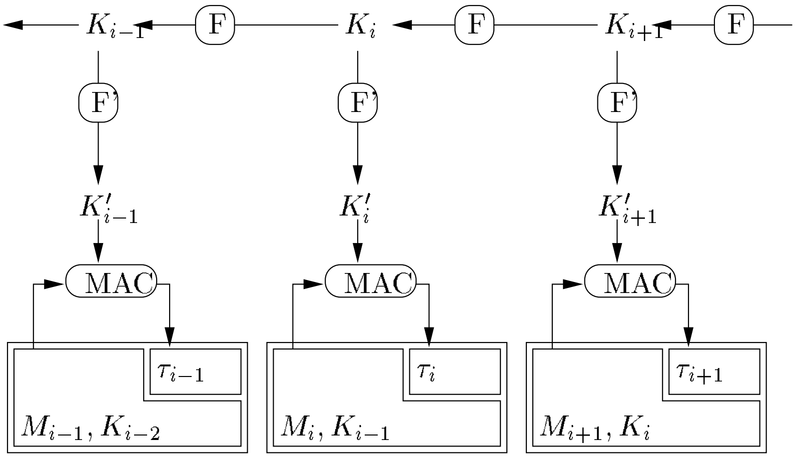

Figure 3.1: The simplified TESLA scheme.

Losses

Once the initial commitment is received, the TESLA scheme tolerates arbitrary packet loss. If k consecutive packets $\{P_{j+1}, ..., P_{j+k}\}$ are missing after P_j then we can still compute the corresponding keys in the chain once we receive P_{j+k+1} which reveals K_{j+k} and thus $K_{j+k-1} = F(K_{j+k})$ and so on until we get to $K_j = F^{k+1}(K_{j+k+1})$, which allows us to authenticate P_j.

3.3.2 Enhancements

Our above description of TESLA is partially simplified and relies on an implicit assumption: the sender and receiver have a common clock which allows them to share the same values for t_i. In practice, the sender and the receiver will need to synchronize with each other. However the sender and the receiver only need to have a *loose* synchronization. The receivers need to estimate the sender' s time t_i as well as a value δ_t which represents the maximum uncertainty on the senders time. The *third verification rule* is simply reformulated as follows:

3. (*Time*) A packet P_i arriving at time t_i^{AR} must verify $t_i^{AR} + \delta_t < t_{(i+1)}$.

This new version of the security condition has still some drawbacks. First, it limits the speed of the transfer rate since we must assume a time interval large enough between t_i and t_{i+1} to accommodate the maximum estimation error δ_t of most receivers, and it constrains the sender to output packets at a constant rate.

The authors of TESLA solve both problems. First, instead of revealing the MAC key $K'_i = F(K_i)$ of packet P_i in the next packet P_{i+1} they propose to reveal K_i in a further packet P_{i+d}. This allows to increase the transfer rate, but also increases the authentication latency. Additionally, the authors of TESLA propose to use a time based key selection instead of an index based key selection. All packets send around the same time use the same MAC key. More precisely, if the stream starts at time

t_0 then the index of a MAC key K_i' of a packet transmitted at time t is determined as $i = \left\lfloor \frac{t-t_0}{T_\Delta} \right\rfloor$ where T_Δ is the interval length. The third verification rule is then:

3. (*Time*) A packet arriving at time t^{AR} apparently send at interval i must verify $\left\lfloor \frac{t^{AR}+\delta_t-t_0}{T_\Delta} \right\rfloor < i + d$.

We refer the reader to [PCTS00] for methods relating to the selection of T_Δ and d.

Finally, the authors of TESLA propose to use multiple chains with different choices of (T_Δ, d) to accommodate between different receivers with different bandwidth and network delays.

3.3.3 Joinability

The scheme we described above clearly assumes that the receivers start authenticating from the beginning of the stream. To start authenticating a stream at packet $P_{(i+1)}$ the receiver needs to:

1. Perform time synchronization with the source.

2. Receive a commitment $Sign(F(K_i))$ to a key K_i in the chain.

If we implement these operations on a receiver request basis, they will introduce a two way communication between the source (or a dedicated server) and the recipient. In [PCTS00] the authors suggest to perform periodic broadcasts of commitments $Sign(F(K_i))$, whereas time synchronization can be performed with distributed trusted time servers in order to avoid scalability issues.

If we perform periodic commitment broadcasts every b packets, this increases the overhead per packet but allows the stream to be joinable every b packet.

3.3.4 Conclusion

TESLA is a very interesting approach for online stream source authentication where nonrepudiation is not needed. It performs with a cost almost similar to simple unicast MAC scheme. It has no sever side buffering with a very reasonable tradeoff in terms of latency. We summarize our analysis in the frame blow.

TESLA scheme summary

Scheme type: Online source authentication of lossy streams.

Advantages:

- **Robustness:** perfect robustness one the original
- **Cost**: low on average.
- **Overhead:** low, with only MAC key, a MAC for each packet only (and the amortization of the commitments if periodic commitment broadcasts are used).
- **Latency:** reasonable, depends on parameter d above.
- **Buffering:** none.

Main Drawback: Requires a loose but secure time synchronization between the source and the recipients, which is not always possible. Moreover each secure time server is a potential target for an adversary who wishes to subvert the authentication of packets.

Other characteristic: Requires periodic commitment transmission to allow **joinability**.

3.4 Hybrid Techniques

We refer to techniques which combine several hashes (or MACs) with a signature as *hybrid* techniques. A single signature is amortized over a block of b packets. While the previous scheme used time constrains to extend an initial commitment, here, different hash techniques are used to extend the asymmetric properties of the signature to all the packets in the block. These hash techniques are designed to tolerate certain loss patterns.

3.4.1 Hash Trees

Hash trees which where proposed by WONG AND LAM[WL99] are based on MERKLE authentication trees[Mer89]. Let H denote a keyless cryptographic hash function and let $.||.$ denote concatenation.

Background

Without loss of generality, we will only describe in detail the general binary tree construction. Consider a block of $b = 2^k$ data packets $\{D_1, ..., D_b\}$ from which we construct a fully balanced binary tree with n leafs as follows:

- Each leaf ordered from left to right represents the hash h_i of a packets P_i: $h_i \leftarrow H(P_i)$

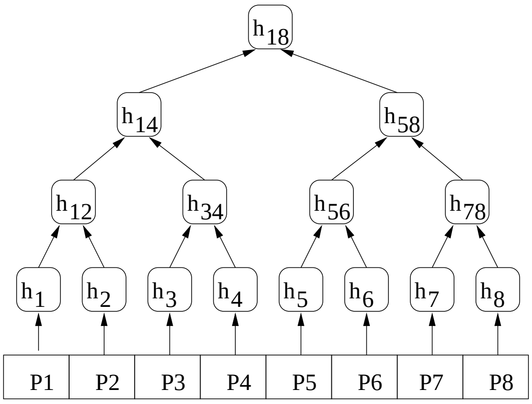

Figure 3.2: A 8 packet binary hash tree.

- Each non-leaf vertex in the tree is the hash of its two children ordered from left to right and concatenated.

An example of an hash tree for 8 packets is given on figure 3.2. For convenience we identify vertices in an authentication tree by the hashes they represent. We will let $Parent(h_i)$ denote the hash value of the parent of h_i. For example on figure 3.2 we have $h_{14} = Parent(h_{12})$ as well as $h_{14} = Parent(h_{34})$.

Let $Q_i = \{h_{i_1}, ..., h_{i_k}\}$ define a path from the root to the i^{th} leaf in the tree, the hash of P_i. For each packet P_i we define the *associated hash set* $\overline{Q}_i$ as the set of $(k-1)$ hashes $\{\overline{h}_{i_1}, ..., \overline{h}_{i_{(k-1)}}\}$ which verify $\left(Parent(\overline{h}_{i_j}) \in Q_i; \overline{h}_{i_j} \notin Q_i\right)$. For example on figure 3.2, the associated hash set of $Q_3 = \{h_{18}, h_{14}, h_{34}, h_3\}$ is $\overline{Q}_3 = \{h_{58}, h_{12}, h_4\}$.

Hash tree property: Given a data packet D_i and its associated hash set $\overline{Q}_i$ it is always possible to compute the hash of the root of the tree.

Indeed from a packet D_i we can compute $H(P_i)$ and combine it with further hash operations with the values in $\overline{Q}_i$ to recover the root of the tree. For example in figure 3.2, we can compute h_{18} form P_3 and $\overline{Q}_3 = \{h_{58}, h_{12}, h_4\}$ since we have $h_{18} = H(H(h_{12}||H(H(D_3)||h_4))||h_{58})$.

Sender tasks: To send a stream of packets the sender divides the stream in blocks of b packets and proceeds as follows:

1. Compute the hash tree associated to the packets of the block $\{D_1, ..., D_b\}$
2. Compute the signature σ_r of the root h_r of the hash tree: $\sigma_r \leftarrow \mathcal{S}(h_r)$.
3. **For** $i = 1, ..., b$ **do**
   send $P_i = \left(D_i||\overline{Q}_i||\sigma_r\right)$

The authors of the hash tree construction suggest to append the root signature to each packet as illustrated on line 3. This allows each packet to be completely independent from another so an individual packet can be verified even if all other packets in the block are lost.

Receiver tasks: The receiver who receives a first packet P_i in a block will parse P_i as $D_i||\overline{Q}_i||\sigma_r$, then it will use the hash tree property to recover the hash of the root of the tree h_r and next it will verify the signature. For the following packets in the block, the receiver will not need to verify the signature again, he only needs to compute enough hashes to verify that the packet is part of the hash tree.

Recalling figure 3.2, let's assume that P_1 and P_2 are lost and P_3 is received, form $P_3 = \left(D_3||\overline{Q}_3||\sigma_r\right)$ we can compute $h_{34} = H(H(D_3||h_4))$, then $h_{14} = H(h_{12}||h_{34})$ and $h_r = h_{18} = H(h_{14}||h_{58})$ and finally we can verify the signature with $\mathcal{V}(\sigma_r, h_{18})$. If the signature is valid and the receiver gets the next packet $P_4 = \left(D_4||\overline{Q}_4||\sigma_r\right)$ he will only need to verify that $h_4 = H(D_4)$ since he already authenticated h_4 with P_3.

Conclusion

Appending the signature of the hash tree to each packet in a block makes the scheme robust to arbitrary packet losses. The scheme buffers b packets, but has no authentication delay. The computational cost of the scheme is one signature per block and $(2.b - 1)$ hashes. The overhead per packet is $\log_2(b).h + s$ where h is the size of the hash and s the size of the signature.

In this scheme, the choice of b is a compromise between buffering and overhead versus the computational cost. If b is small then signatures will be verified and generated more frequently which increases the computational cost of the scheme. On the other hand, if b is large the server side buffering becomes important and the overhead will be increased by a logarithmic factor.

We will illustrate this scheme in the two case scenarios we proposed in section 2.2.2, assuming a signature size of 128 bytes and a hash length of 16 bytes.

- **CASE 1:** We could assume that the traffic sensors don't have any real buffering capacity so the hash tree construction might not be the best construction, however, we propose to assume that $b = 32$. This gives us a signature verification approx. every 1.5 seconds and an overhead of 208 bytes ! This is to be compared to the data itself which only represents 64 bytes. Clearly, if these sensors communicate through a low bandwidth link, this overhead may become an issue.

- **CASE 2:** In this case we can assume that both endpoints of the communication channel have better buffering and computational capacity than in CASE 1. If we choose $b = 512$, we have a signature generation and verification every second, a buffering of one second of video on the sender side and an overhead per packet of 272 bytes.

The main advantage of this scheme is its robustness, however this comes at a big cost in terms of overhead, even bigger than signing each block with a traditional signature. Additionally, in this scheme the sender is required to buffer b packets.

Wong and Lam scheme summary

Scheme type: Authentication of lossy streams with nonrepudiation.

Advantages:

- **Robustness:** perfect robustness.
- **Joinability:** on every packet.
- **Latency:** none.

Drawbacks:

- **Overhead:** Even more than a digital signature. Uses $ln_2(b).h+s$ bytes: a digital signature of s bytes is amortized over b packets using $ln_2(b)$ hashes of h bytes.

Other Characteristic:

- **Buffering:** requires the buffering of b packets at the server.
- **Cost:** cost 1 signature and $(2b-1)$ hashes.

3.4.2 Hash Chains

Based on the observations of PAXSON[Pax99] who conducted a large scale survey of TCP/IP Internet communications and who showed that losses often occur in bursts, GOLLE AND MODADUGU[GM01] proposed a stream authentication mechanism designed to tolerate the loss of packets in bursts of at most β packets in a block.

Background

GOLLE AND MODADUGU construct a directed acyclic graph between the packets of the block, by putting the cryptographic hash of a packet in one or several other packets. The hash h of a packet P is computed over the data in that packet as well as all hashes that have been appended to P. The value h is then appended to other new packets, and so on, forming an acyclic graph over the packets. If a packet P' is signed then any packets P for which there exists a path in the graph joining P to P' can be authenticated. In their work, GOLLE AND MODADUGU propose methods to design such acyclic graphs in an optimal way regarding bursty packet losses. Their simplest scheme is constructed as shown on the example of figure 3.3: the hash of a packet P_i is stored both as part of the following packet P_{i+1} and as part of $P_{i+1+\beta}$. Finally the hashes of the last $(\beta+1)$ packets are sent, along with a signature of these $(\beta+1)$ hashes for verification.

The same authors further refined their hash chain construction, to create "Augmented Chains", which require to buffer a few packets, but allows a smaller set of hashes to be signed at the end. The principle remains the same and we refer the

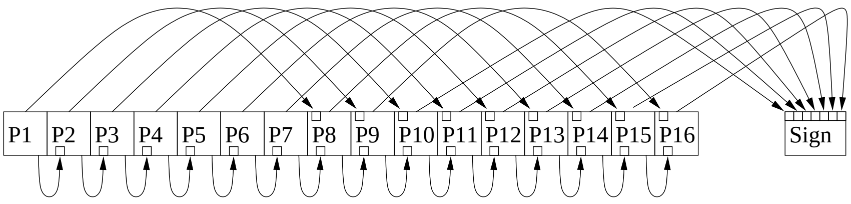

Figure 3.3: Augmented chain resisting to bursts of 6 packets in a 16 packet block.

reader to their work [GM01] for details. It is worth noting that their first scheme can tolerate several bursts in a block while the augmented chain construction may have difficulties in some situations if there are several bursts in the same block, consequently we will focus on their first scheme in this comparison.

Optimality. The authors of this scheme also proved that their constructions were optimal in terms of overhead in the above setting. More precisely, if we restrict ourself to hash graphs constructed by replicating the hashes of one packet in other packets as above then the overhead per packet in terms of number of hashes is optimal: each packet needs to carry at least 2 hashes in order to tolerate losses. This fact is based on the following observation: if we want to authenticate packets in the presence of losses than the hash of a packet needs to be duplicated in a at least two different locations. (This result does not cover the overhead generated by the signature.)

Analysis

Hash Chain Overhead: Let h denote the size of the output of the cryptographic hash function we use, and let s denote the size of the digital signature. The authors of [GM01] do not detail how to choose β nor do they provide a clear method to deal with signature loss except to suggest the transmission of several copies of the signature. If these signatures are transmitted far enough apart, we can consider that their loss probabilities are uncorrelated. If we assume that γ signatures are transmitted, we can approximate the cost of the hash chain construction as $\delta_{HC}(\gamma, b) = \gamma \frac{(\beta+1).h+s}{b} + 2h$ bytes per packets, with the notations already used throughout this work. The size of b is essentially constrained by the authentication delay, which here is at most the distance between the first packet of the block and the γ^{th} redundant signature that is transmitted for that block. Since the simple hash chain construction is not sender side buffered the γ signatures pertaining to a block are transmitted after the last packet of that block.

Recalling the Markov chain model of section 2.2.1 we estimated that the probability that a burst of k lost packets occurs is $q.(q-1)^{(k-1)}$ with an average length of $1/q$ packets in a burst. Consequently we will choose β in the hash chain such that the probability that a burst exceeds β is low, for example such that $1 - \sum_{k=(\beta+1)}^{\infty} q(1-q)^{k-1} \leq 99\%$. If we refer to the two case studies in section 2.2.2, we would have:

- **CASE 1:** We propose $b = 160$, $\gamma = 2$, $\beta = 21$ since $q = 0.2$. We would transmit the first signature at the end of the block and the second signature 20 packets later (1 second). The probability that one of the signature arrives is approximately $1 - 0.05^2 = 0.9975$ and the overhead per packet is $\delta_{HC}(\gamma, b) \approx 38$ bytes.

- **CASE 2:** This case is more problematic because the network is extremely lossy and the signature has a high probability of being lost. Indeed if we take $\gamma = 8$ the probability that one redundant signature at least arrives is $1 - 0.6^8 \approx 0.99$ (if we take $\gamma = 4$ the signature arrival probability is lowered to 0.87). But this means that each block is transmitted along with 4 to 8 signatures and it becomes difficult to define a reasonable size for $b < 512$. If we choose b small then we need to compute several signatures per second and we need to send several copies of each them during the same time (without a guaranty that lost signatures will still be independent). If we choose b larger then the probability of authenticating a packet within the authentication delay becomes lower. As a indication, if $b = 256$, $\gamma = 8$, $\beta = 43$ since $q = 0.1$, we have $\delta_{HC}(\gamma, b) \approx 58$ bytes.

No matter how good the network conditions are and no matter how long the block size is, the hash chains have at least an overhead of $2.h$ per packets (with an additional amortized overhead for the signature).

Conclusion

The hash chain construction is an interesting authentication scheme which would probably deserve greater analysis, such as the one we sketched here, regarding the choice of β. The main advantage is its simplicity and robustness to Internet like packet losses, with a reasonable overhead. On the other hand, in bad network conditions, signature loss issues may lower the probability of authentication or add some authentication delay, as illustrated in CASE 2.

Hash Chain scheme summary

Scheme type: Authentication of lossy streams with nonrepudiation.
This summary applies to the "simple hash chain" construction only.

Advantages:

- **Robustness:** tolerates several bursts of β packets in a block of b packets.
- **Joinability:** on every packet.
- **Overhead:** 2 hashes, *plus* the cost of the reliable delivery of the block signature (which may include multiple retransmissions).
- **Buffering:** none

Drawback: Signature transmission is not clearly addressed,

Other Characteristic:

- **Maximum Latency:** a block b packets, plus the time needed for the reliable delivery of the block signature.
- **Cost:** cost 1 signature and $b + 1$ hashes.

3.4.3 EMSS

Similarly to Golle and Mogadugu, Perrig et al. used hash chain techniques in their EMSS[PCTS00] scheme, using a different methodology.

Background

As opposed to the work of Golle and Modadugu, which uses a deterministic edge relationship pattern among the packets in the chain, the EMSS scheme uses randomly distributed edges.

The Basic Scheme.

In the first variation of the scheme, the current packet's hash h_i is replicated in e distinct future packets and the distance between the current packet P_i and the e packets carrying h_i is chosen uniformly in a finite interval $[1, ..., e_{max}]$. Consequently the hash h_i of packet P_i is carried in P_{i+d_1},...,P_{i+d_e} where $d_1, ..., d_e$ are distinct random values in $[1, ..., e_{max}]$. This is applied to all packets regardless of any block consideration.

Packets are only grouped in blocks of b packets for signature purpose: the last e_{sig} hashes of each block are grouped together and signed. The signature packet is thus composed of e_{sig} hashes and the actual signatures of these hashes. The signature packets is transmitted γ times to accommodate for potential loss. Note however that if no signature packets pertaining to a block B_i arrive, then the next

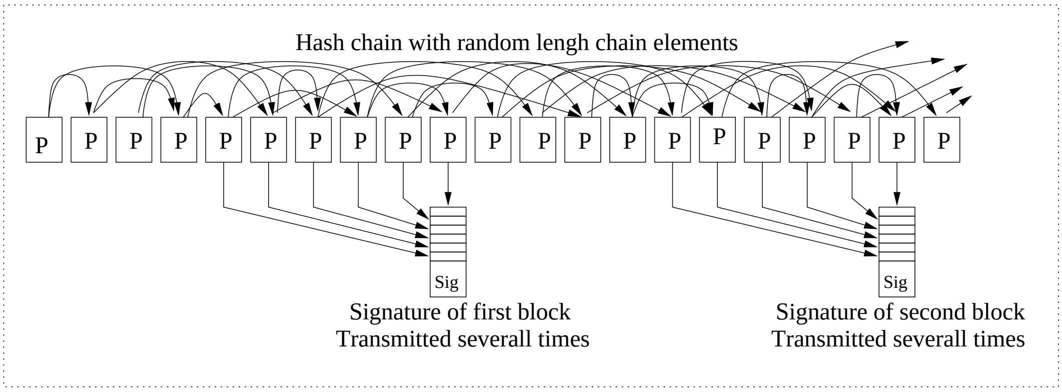
Hash chain with random lengh chain elements
P P P P P P P P P P P P P P P P P P P P P
Sig
Signature of first block
Transmitted severall times
Sig
Signature of second block
Transmitted severall times

Figure 3.4: The Basic EMSS Scheme

block $B_{(i+1)}$ may still be used to authenticate B_i because the hash chain runs *across* block boundaries. In that case however, the authentication latency of block B_i is increased.

We sketch a simplified illustration of the basic EMSS scheme on figure 3.4.

The EMSS scheme is parameterized by many variables:

- e : the number of replications of the hash of a packet.
- e_{max}: the maximum size of a single edge in the hash chain.
- e_{sig}: the number of hashes in the packet signature.
- b: the size of a block (equivalent here to the frequency of a new signature).
- γ : the number of copies of a block signature that are transmitted.

Because of the large number of parameters, the authors of EMSS chose to run simulations, combined with some heuristics in order to tune these parameters, thus we cannot provide any analytical results here. They first assumed independent packet losses, and refined their results by simulating the network losses with a 2 state Markov chain (see section 2.2.1).

A subtle point of the EMSS scheme is the reconstruction of the random graph by recipients. If this graph is truly random, how do the recipients know how to reconstruct it ? In their work, Perrig et al. (section 3.4, footnote 9 in [PCTS00]), suggest two possible approaches to deal with this issue: first since the probability of a collision in the output of the hash function is negligible, we can store the last e_{max} hashes of packets we received, if any packet carries one of the copies of that hash then the packet is considered verified. The second approach is to construct a deterministic computable random graph over the packets. Though no details are provided in [PCTS00], in that case, we believe that some additional information will need to be provided to the recipient when he begins authenticating the stream (*joinability*).

The Extended Scheme. The authors of EMSS noted that the scheme carries some redundancy since the hash of a packet is reproduced in e different locations. As an alternative approach they suggested to split the hash in several chunks and use a recovery scheme that allows to recover the hash if k' out k chunks are recovered. An example of such a recovery scheme is RABIN'S Information Dispersal Algorithm[Rab89]. In some situations, recovering k' blocks out of k, provides more robustness than recovering 1 packet out of e. The authors give the following example: a scheme with packet loss probability of $p = 60\%$ and 80 bit hashes. In the basic EMSS scheme they would use $e = 6$, and if we assume independent packet loss, then the probability that one packet arrives is $1 - q^6 \approx 0.95$. In the extended EMSS scheme they construct $k = 30$ chunk of 16 bits each such that $k' = 5$ chunks are enough to recover the original 80 bit hash of a packet. The probability that a receiver gets more than 4 packets is then $1 - \sum_{i=0}^{i=4}\left(\begin{array}{c}30\\ i\end{array}\right).q^{30-i}.(1-q)^i \approx 0.9999$, higher than in the basic scheme.

Analysis

Let h denote the size of the output of the cryptographic hash function and let s denote the size of the digital signature. The authors of EMSS do not provide a detailed algorithmic description of their parameter selection or simulations. Consequently, we can't provide a precise relationship between the different parameters of the scheme and the network loss patterns, or the authentication latency. What we can simply say is that in the basic scheme the overhead per packet is $\frac{\gamma.(s+e_{sig}.h)}{b} + e.h$, in the extended scheme the overhead will further depend on the choice of k and k' in the recovery scheme.

Since we borrowed the two case studies presented in EMSS in section 2.2.2, we can conveniently recall their results here:

- **CASE 1:** They chose $e = 2$, $e_{max} = 50$, $e_{sig} = 5$ and $b = 100$. The choice of $b = 100$ is to have an most ten second verification with high probability. The basic scheme is sufficient here and yields an average overhead per packet of 22 bytes and a verification probability of 98.7% according to their simulations. This result presented in[PCTS00] assumes a hash of 80 bits. Since this not a secretly keyed hash function we believe that this choice is a little weak. Also to make a fair comparison with other scheme, we compute their result with a 128 bit hash (such as MD5[Riv92]). This leads to a readjusted overhead of 34 bytes.

- **CASE 2:** Here they chose $e = 6$, $e_{sig} = 40$ and $b = 200$ (the choice of e_{max} is not clear in [PCTS00]). Here the authors use the extended scheme to split the hash in k =25 chunks. The signature is send twice for each block of 200 packets, thus each packet can be verified with approximately 5 signatures per second (there are 512 packets per second, if we assume independent signature packet loss, the probability that one of the 5 packets arrive is $1 - p^5 \approx 92\%$). The overhead per packet here, again with a 80 bit hash is about 55 bytes per packet or $\frac{(2*128+40*10)}{200} + 50$ with 25 chunks of 2 bytes each (their simulation predicts an average verification probability of 97% on the final 2000 packets with a minimum verification probability of 90%). If we adjust the hash length closer to 128 bytes (we use 120 bytes instead for convenience), each chunk is now 3 bytes and the overhead becomes $\frac{(2*128+40*15)}{200}+75$ or approximately 82 bytes.

Conclusion

With adjusted hash lengths the case studies gives a better perspective to compare the EMSS scheme with the previous Hash Chain proposed by Golle and Modadugu. In CASE 1, both schemes are equivalent in terms of overhead with a slight advantage for EMSS due to their better signature management. In CASE 2, EMSS has a greater overhead, but recall that the scheme of Golle and Modadugu requires more signatures packets to achieve similar robustness. Both of these hybrid schemes where published independently and we believe that a combination of both approaches could lead an interesting scheme: use the simple hash chaining technique from Golle and Modadugu, while extending it across block boundaries like EMSS as well as using the same approach as EMSS to distribute the signature packets.

EMSS scheme summary

Scheme type: Authentication of lossy streams with nonrepudiation.

Advantages:

- **Robustness:** tolerates several bursts of β packets in a block of b packets.
- **Joinability:** on every packet (if a method to reconstruct the random graph is provided a priori to the receiver).
- **Overhead:** e MACs and γ signatures amortized over b blocks.
- **Buffering:** none.

Drawback: This scheme has no serious drawbacks regarding the list of requirements we established in Chapter 2. However, in [PCTS00] no clear method is provided for the parameter selection, which would be beneficial to further comparison with other protocols.

Other Characteristic:

- **Maximum Latency:** a block b packets, plus the time needed for the reliable delivery of the block signature.
- **Cost:** cost 1 signature and $e.b+1$ hashes.

3.4.4 Conclusion.

The most interesting multicast schemes presented in this chapter are TESLA, Hash Chains and EMSS. In environments where secure time synchronization is possible and where non repudiation is not required, TESLA is the most interesting authentication scheme. In other situations a combination of Hash Chains with the signature chaining mechanisms of EMSS is probably an approach that would deserve further study. Moreover, in these schemes, traditional signature schemes can be replaced with faster alternatives such as those presented in the first part of this chapter to achieve even better performance.

There has been some attempts to provide a more formal framework to the construction of hash or MAC graphs by Miner and Standon[MS01]. These authors work with an offline approach, where the first packet of the stream is signed and the hashes or MACs are computed *before* the stream is broadcasted. Nonetheless some of their results confirm the good properties of random graphs as used in EMSS for independent random losses. They also propose methods to construct graphs with different authentication priority classes, in order to increase the authentication probability of some packets and reduce it for other. They argue that it is interesting for application level protocols such as MPEG where certain frames are more important than others (such as synchronization frames). We refer the reader to their work for details. Note that one of the main conclusions of their work is that “there is still much to be done towards forming a comprehensive theory of graph based authentication schemes”.

In the next chapter, we propose a nonrepudiation scheme that is in the same category as EMSS and Hash Chain but achieves even lower communication overhead, additionally providing a uniform mechanism to deal with both data and signature packet loss.

Chapter 4

An Integrated Online Stream Authentication Algorithm

4.1 Introduction

The scheme presented in this chapter, which is based on [PM02a], uses a combination of hash and signature techniques with FEC, or more precisely, erasure codes. The two most employed techniques to achieve reliable delivery of packets in computer communication protocols are ARQ (Automatic Repeat reQuest) techniques and FEC (Forward Error Correction). ARQ techniques are used every day in Internet protocols such as TCP, while FEC techniques have long been confined to the telecommunications world. However, there has been recently a surge of interest for FEC techniques in the Internet world, often in combination with more traditional ARQ approaches[NBT98, BLMR98]. While in the telecommunications world FEC techniques are used most often to detect and correct errors occurring in the transmission of a stream of bits, they are used in the Internet world to recover from the loss of packet sized objects. Indeed, in the Internet world a packet is either received or lost. A packet can be considered lost if it does not arrive after a certain delay or perhaps if it has a bad checksum. Considering the hybrid schemes in the previous chapter, a preliminary idea could be to use FEC to transmit the signature alone, but it turns out that FEC can also be used as a complete alternative to hash trees or chains (section 3.4) to transmit authentication information, with lower overhead per packet in most cases than any other scheme suitable for *live* broadcasts.

A brief overview of erasure codes will be presented in the next section. Our scheme is formalized in section 4.3 as well as its relationship with Internet loss patterns based on the Markov chain model of section 2.2.1. Section 4.4 discusses the cost and overhead of our scheme and presents its use in a few concrete scenarios. Finally, we compare other practical online lossy stream authentications schemes in section 4.5 with our approach.

4.2 Background

4.2.1 Erasure Codes

Erasure Code techniques are quite similar to secret sharing schemes[Sha79] that are used in some cryptographic protocols. An erasure code generation algorithm $C_{k,r}$ takes a set $X = \{x_1, ..., x_k\}$ of k source packets in a block and produces $(k+r)$ code packets:

$$\{y_1, ..., y_{(k+r)}\} \leftarrow C_{k,r}(X)$$

The main property of the set $Y = \{y_1, ..., y_{(k+r)}\}$ is that any subset of k elements of Y suffices to recover the source data X with the help of a decoding algorithm D_k. To be exact, the decoding algorithm D_k needs to know the position, or index, of the k received elements in Y to recover X. This information can often be derived by other means (such as the packet sequence number) and we will assume in the remaining discussion that this information is available implicitly to D_k. If the first k code packets are equal to the source packets, that is $\{y_1, ..., y_k\} = X$ where $\{y_1, ..., y_{(k+r)}\} \leftarrow C_{k,r}(X)$, we call the code *systematic* and the extra redundancy packets $\{y_{(k+1)}, ..., y_{(k+r)}\}$ are called parity packets. Systematic codes are very useful since they do not require any additional processing from the recipient in the case where no loss occurs.

It is important to note that Erasure Codes are not used in the same context in the Internet as in telephony. Here the codes are not designed to recover damaged packets but rather the loss of full packets in a block of several packets. Intuitively, a individual packet can therefore be viewed more like a single code symbol rather than a set of symbols. For a good introduction to practical erasure codes we refer the reader to the work of L. Rizzo[Riz97] where Reed-Solomon erasure codes are described. These codes operate in $GF(2^n)$ and may not be efficient for large data blocks of packets (several hundred kilobytes). However, they are suitable in our scenario since we work on data units that are much smaller than a packet (typically 16 or 20 bytes), as shown below. For faster codes, we refer the reader to the work of M. Luby et al. on Tornado Codes [LMS+97, BLMR98], where codes with near linear coding and decoding times are described.

In the remaining of this work, $C_{k,r}(.)$ will describe a practical systematic erasure code generation algorithm which takes k source packets and produces $(k+r)$ code packets. If $X = \{x_1, ..., x_k\}$ is the source data and Y are the r extra generated parity packets, we will write $\{X; Y\} \leftarrow C_{k,r}(X)$. The corresponding decoding algorithm will be denoted $D_k(.)$ and if Z describes the set of received elements and X the source data, we will write $X \leftarrow D_k(Z)$ to describe the recovery process (where $\#Z \geq k$).

4.2.2 Notations

In this work we will consider a stream to be divided in consecutive blocks of b packets. Since a stream does not necessarily exactly contain a number of packets that is an exact multiple of b we allow the use of dummy padding packets at the very end of the stream to match a b packet boundary. Our authentication scheme is parameterized by b the block size in packets and $p \in [0..1[$ the maximum expected loss rate per block.

We will denote H as a cryptographic hash function such as SHA[Nat95] or MD5[Riv92] which produces hashes of h bytes. The couple $(\mathcal{S}, \mathcal{V})$ will denote the digital signature and verification algorithms respectively associated with the source of the packet stream, such as RSA[RSA99, BR95] for example. The size of the signatures will be expressed as s bytes. For RSA, a typical value for s is 128 bytes (or 1024 bits).

4.3 Stream Authentication

4.3.1 Authentication Tags

Consider a block as a sequence of b packets $[P_1, ..., P_b]$. Let $\{h_1, ..., h_b | h_i \leftarrow H(P_i)\}$ be the set of hash values of these packets with a cryptographic hash function $H(.)$. From this hash set we build a set of b authentication tags $\{\tau_1, ..., \tau_b\}$ with the following algorithm $\mathcal{T}_{[b,p]}$, which uses some of the notations introduced in the previous section:

```
Tag generation: $\mathcal{T}_{[b,p]}$
INPUT: $\{h_1, ..., h_b\}$
OUTPUT: $\{\tau_1, ..., \tau_b\}$
$\{X; \overline{X}\} \leftarrow C_{b,\lceil pb \rceil}(X)$                                  (1)
$\sigma \leftarrow \mathcal{S}(H(h_1 || ... || h_b))$                                  (2)
$\{Y; \overline{Y}\} \leftarrow C_{\lfloor b(1-p) \rfloor, \lceil pb \rceil}(\overline{X} || \sigma)$                  (3)
Split $\{Y; \overline{Y}\}$ into $b$ equal length tags $\{\tau_1, ..., \tau_b\}$.        (4)
```

We propose a more visual representation of the tag algorithm on figure 4.1.

We observe that $\mathcal{T}_{[b,p]}$ uses two different erasure codes, in steps (1) and (3). The values $\{Y; \overline{Y}\}$ on line (3) is of total length that is a multiple of b bytes, because we have $b = \lfloor b(1-p) \rfloor + \lceil pb \rceil$. This allows us to divide $\{Y; \overline{Y}\}$ into equal length tags on line (4). To exploit the tag generation algorithm we will first define our authentication criterion:

Authentication criterion: In this work we say that a packet P_i is *fully authenticable* in a block if, given the set of hashes $Z = \{h_1, ..., h_b\}$ of packets in the block and their signature $\sigma = \mathcal{S}(H(Z))$, we can verify that both $\mathcal{V}(\sigma, H(Z)) = true$ and $H(P_i) = h_i$.

The proposed schemes in this work are based on the following property of the tag generation algorithm.

Proposition 4.1 *Let $\Pi = [P_1, ..., P_b]$ be a block of b packets and $\{h_1, ..., h_b | h_i \leftarrow H(P_i)\}$ its associated hash set. If we compute $A = \{\tau_1, ..., \tau_b\} \leftarrow \mathcal{T}_{[b,p]}(\{h_1, ..., h_b\})$ then any subset of at least $\lfloor b(1-p) \rfloor$ packets in Π can be authenticated using any subset of at least $\lfloor b(1-p) \rfloor$ tags in A.*

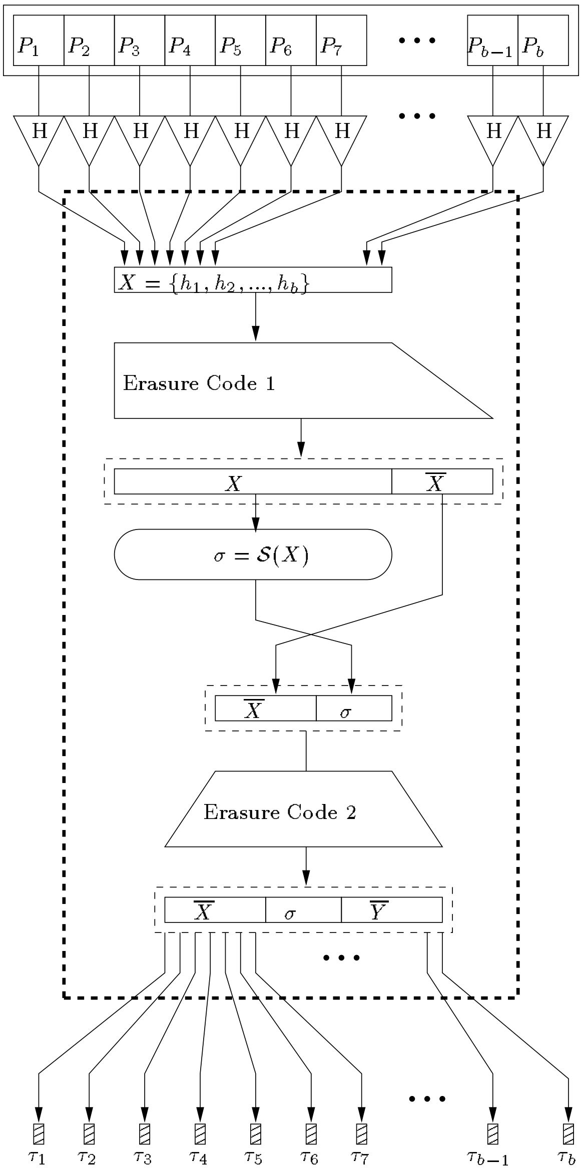

Figure 4.1: the tag generation algorithm

Proof. Define $r = \lfloor b(1-p) \rfloor$. Let $\Pi' = [P_{e_1}, ..., P_{e_r}]$ be a subset of r packets in Π and let $A' = [\tau_{a_1}, ..., \tau_{a_r}]$ be a subset of r packets in A. We can compute $\{\overline{X}||\sigma\} \leftarrow D_{\lfloor b(1-p) \rfloor}(A')$ since A' contains $r = \lfloor b(1-p) \rfloor$ elements. Let $E = \{h_{e_1}, ..., h_{e_r} | h_{e_i} \leftarrow H(P_{e_i})\}$ be the hashes of the received packets. We can recover $\{h_1, ..., h_b\}$ form B and $\overline{X}$ by computing $\{h_1, ..., h_b\} \leftarrow D_b(E||\overline{X})$. Finally we can compute $\mathcal{V}(\sigma, \{h_1, ..., h_b\})$ to authenticate the received packets Π' to verify our authentication criterion. $\Diamond$

A direct corollary of the proposition above is that both a block of packets and their authentication tags can withstand a loss rate of at most $\lceil pb \rceil$ elements while allowing us to authenticate the remaining packets.

Finally, from the construction of the algorithm above we can determine the size of an authentication tag:

Proposition 4.2 *Let h define the length of our cryptographic hashes and s the size of the signatures. The size of an individual authentication tag is expressed as a function $\delta(b, p)$ of both the number of packets in a block and p the maximum expected loss rate per block, as follows:*

$$\delta(b, p) = \frac{\mathcal{R}_{\lfloor (1-p)b \rfloor}(s + \lceil p.b \rceil h)}{\lfloor (1-p)b \rfloor}$$

where $\mathcal{R}_n(z)$ is an integer function which returns the lowest multiple of n greater or equal to z.

Proof. Let y denote the size of the value $\{Y; \overline{Y}\}$ and x the size of $\overline{X}||\sigma$ padded to the proper length, both on line (3) of the algorithm. We have $\delta(b, p) = y/b$. From the erasure code parameters on line (3) we have $y = x\frac{\lfloor (1-p)b \rfloor + \lceil p.b \rceil}{\lfloor (1-p)b \rfloor} = x\frac{b}{\lfloor (1-p)b \rfloor}$ and thus $\delta(b, p) = \frac{x}{\lfloor (1-p)b \rfloor}$. The value of x is the the sum of the size of $\overline{X}$ and the signature σ, *padded to the appropriate length* for the erasure code of line (3). From line (1) we compute the size of $\overline{X}$ as $\lceil p.b \rceil h$ and write s as the size of σ which yields $x = \mathcal{R}_{\lfloor (1-p)b \rfloor}(s + \lceil p.b \rceil h)$. $\Diamond$

4.3.2 Proposed Schemes

In our stream authentication scheme we propose to piggyback authentication tags in the packets of a block and use Proposition 4.1 to authenticate received packets when the loss rate in a block is less than p. We propose 3 different variants of our scheme which only differ by the positioning of the authentications tags.

In this section we will denote a stream as a set of m blocks $B_1, ..., B_m$. The individual b packets in each block B_i are identified as $P[i, 1], ..., P[i, b]$. The corresponding authentication tags are identified as $\tau[i, 1], ..., \tau[i, b]$. The packets $P[i, j]$ are a combination of just two things: a stream data packet $D[i, j]$ and an authentication tag.

ECU: The unbuffered sender scheme. In this scheme we use packets in a block $B_{(i+1)}$ to piggyback authentication tags related to block B_i. The j^{th} packet in a block B_i is thus defined as $P[i, j] = \{D[i, j]||\tau[i-1, j]\}$. This requires the

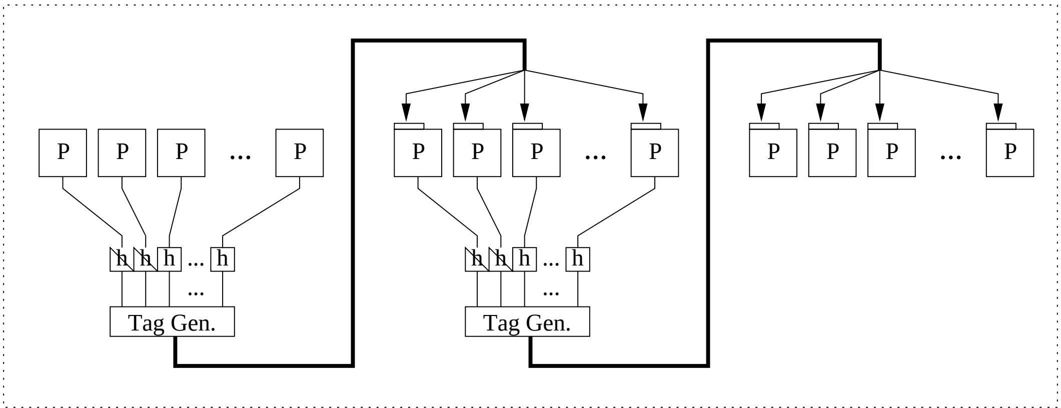

Figure 4.2: The ECU scheme

sender to create an extra padding dummy block $B_{(m+1)}$ to allow the last block B_m to be authenticated. This scheme has the particularity that it does not require any stream data packet buffering from the sender, only the hashes of the packets in the current block need to be stored by the sender who can then compute the necessary authentication tags to be piggy-backed in the next block. In this sense, this scheme is truly an *live* authentication scheme. The tradeoff of this construction is that the receiver will experience a delay of two blocks in the worst case before he can authenticate the first packet in a blocks he received.

This construction creates a dependency between two consecutive blocks, thus in the event of a loss that exceeds the threshold p and in particular if a whole block B_i is lost than we will not be able to authenticate $B_{(i-1)}$.

An interesting aspect of the ECU scheme is that it also gives an extra amount of time for the sender to compute the signature of a block and the second authentication code. Recalling line (3) of the tag generation algorithm we have $\{Y;\overline{Y}\} \leftarrow C_{\lfloor b(1-p)\rfloor,\lceil pb\rceil}(\overline{X}||\sigma)$ where $\{Y;\overline{Y}\}$ is split in b authentication tags. Accordingly we can rewrite $\{Y;\overline{Y}\}$ as $\{\overline{X}||\sigma;\overline{Y}\}$, thus the first $l_{|\overline{X}|}$ authentication tags will contain elements representing $\overline{X}$, then the next group of $l_{|\sigma|}$ tags will represent the signature σ and finally the last group of tags will represent the $b - l_{|\overline{X}|} - l_{|\sigma|}$ associated parities. Consequently, the first authentication coding operation on line (1) of our algorithm needs to be produced before sending block $B_{(i+1)}$, however, the signature on line (2) only needs to be computed after the first $l_{|\overline{X}|}$ packets of $B_{(i+1)}$ and the second code on line (3) only needs to be ready after the $l_{|\overline{X}|} + l_{|\sigma|}$ first packets of $B_{(i+1)}$.

EC2: The double buffer scheme. Instead of piggy-backing tags in the next block, we examine the possibility of piggy-backing tags in the previous block. In other words, the tags of block B_i are put in the packets of block $B_{(i-1)}$ and the packets in block B_i are defined as $P[i,j] = \{D[i,j]||\tau[i+1,j]\}$. This requires the sender to create an extra padding dummy block at the beginning of the data stream. The main advantage of this construction is that the receiver can authenticate each received packet immediately upon reception. The main drawback of this scheme is

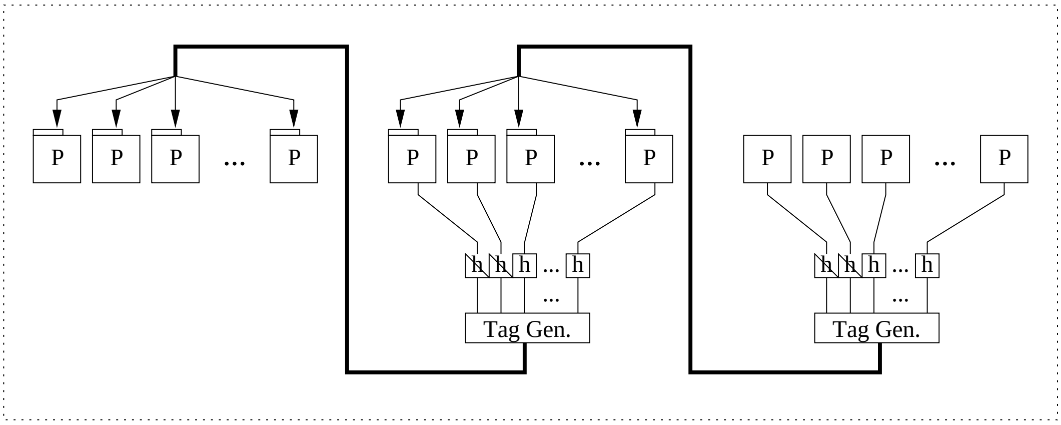

Figure 4.3: The EC2 scheme

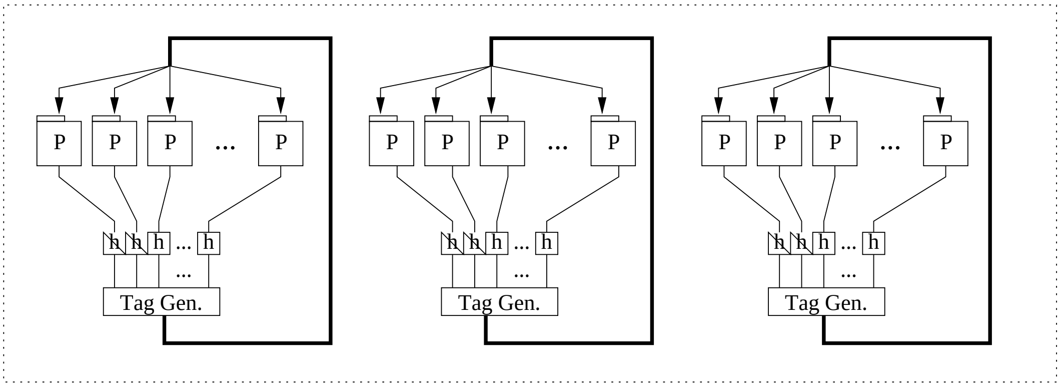

Figure 4.4: The EC1 scheme

that it requires the sender to buffer two blocks at a time. In this sense it is not a truly *live* scheme but in some applications, our double buffering is still acceptable.

This construction also creates a dependency between blocks similar to ECU, with similar consequences.

EC1: The single buffered scheme. The most obvious construction and perhaps the one that offers the best compromise between the sender buffering and the receiver authentication delay is to piggyback the tags of a block B_i in the block B_i itself. Packets in a block are simply defined as $P[i,j] = \{D[i,j]||\tau[i,j]\}$. This scheme requires the sender to buffer one block and adds a maximum verification delay of one block for the receiver.

A advantage of this scheme is that it does not create a dependency between blocks, thus if a block losses packets beyond the expected maximum loss rate p, the authentication of neighboring blocks in the stream remains unaffected.

4.3.3 Parameter Choice

Until now we proposed a method which can authenticate a block when a threshold of less than pb packets are lost in a block of b packets. However we need to relate these parameters to concrete average network loss patterns and we will now discuss the choice of the 2 main parameters of our scheme: b the block size and p the maximum loss rate per block.

The goal of an hybrid scheme is to amortize the cost of a signature over several packets. Thus the greater the block size, the less often we will need to compute a signature. On the other hand the block size influences the authentication delay and/or the sender buffer size, depending on which scheme is chosen. The EC2 has the lowest possible authentication delay (1 packet) but the biggest buffering, whereas ECU has no sender-side packet buffering but a maximum 2 block authentication delay. As we said above, EC1 seems to be a good compromise in most situations with both a buffering and a maximum authentication delay of one block. Once a scheme is chosen, we recommend to choose the largest possible block size b within the constraints of the application authentication delay requirements.

The parameter p depends on the loss pattern of our network, according to the Markov chain model presented in section 2.2.1. The strategy we followed in this work was first to choose b, then to simulate a Markov chain over a very large number of blocks and adjust the parameter p such that most blocks would be verifiable (we chose an arbitrary value of 99% verifiable blocks). The Markov chain parameters were derived from μ: the average loss rate and π_1: the average burst length. Note that here the number of losses in a block of b packets can be successfully modeled as a the number of successes in trials of a Bernoulli process with parameter π_1, which is approached by the normal distribution. This approximation could also give us some analytical results but we found the simulations to be more informative.

4.4 Discussion

4.4.1 Computational Cost

Our scheme involves 3 types of operations:

- cryptographic hash computations.
- a digital signature.
- 2 coding and decoding operations.

For each block, the source needs to compute b hash operations, a digital signature (which includes a hash), and generate the 2 codes. Here, the hashing and signing costs are equivalent to other hybrid schemes such as EMSS[PCTS00] or Hash Chains[GM01]. The amount of computation done by the recipient depends on the loss in the network, in an ideal situation he just computes b hashes and verifies a signature. If packets are lost some additional decoding operations will be needed. The codes are used to recover hashes of packets, rather then the packets themselves, thus we will be manipulating small amounts of data. In traditional uses of Erasure Codes, the packets size L is typically over a thousand bytes, while here, we are looking at figures ranging from $L = 1$ to $L = 150$ bytes in the most extreme cases.

If we take a simple Reed-Solomon Erasure Code[Riz97], the computational decoding cost is $\mathcal{O}(m.e.L)$ where m is the number of original message packets, and e the additional parities needed (corresponding to the loss) and L the size of a packet. The coding cost is similarly in $\mathcal{O}(m.k.L)$ where k is the number of parities.

For demanding situations, we can turn to more efficient codes such as Tornado Codes[LMS+97]. These codes are probabilistic and come with what is called a slight "decoding inefficiency": $(1+\varepsilon)m$ packets are needed to recover m original packets with high probability. These codes use the binary XOR operation as a basic operation as opposed to Galois Field operations in the Reed Solomon case, thus we achieve very efficient coding and decoding times of $\mathcal{O}((m+k)ln(1/\varepsilon)L)$. Note that the use of tornado codes would thus conduct us to modify our definitions in section 4.3 to take the decoding efficiency into account. However, in [BLMR98] significant values of $\varepsilon \approx 0.05$ are considered, thus the results we propose in this work should not be significantly different with such a small overhead increase if we use Tornado Codes.

Compared to other hybrid live authentication streams, the main tradeoff of our scheme is the additional computational cost generated by the erasure code. However, since we are operating on a small code packet size, the cost over a block should remain very reasonable. We will show in the next section that the substantial gain we can achieve in terms of overhead per packet is clearly worth the extra computational effort.

4.4.2 Overhead

Evaluation

The overhead in bytes per packet of our 3 schemes is uniquely defined by the size of an authentication tag. Thus, recalling Proposition 2 in section 4.3 we can express the overhead as a function $\delta(b,p)$ of the maximum expected loss rate per block p and the number b of packets in a block:

$$\delta(b,p) = \frac{\mathcal{R}_{\lfloor (1-p)b \rfloor}(s + \lceil p.b \rceil h)}{\lfloor (1-p)b \rfloor}$$

where $\mathcal{R}_n(z)$ is an integer function which returns the lowest multiple of n greater or equal to z.

We would like to emphasize again that this overhead *includes the signature overhead.* Table 4.1 presents a sampling of $\delta(p,b)$ for different values of p and b, with $s = 128$ bytes (1024 bit RSA) and $h = 16$ (MD5[Riv92]). Note that $\delta(b,p)$ remains surprisingly small if either b large or p is reasonably low.

Case studies

We now evaluate our scheme in the two EMSS case studies proposed in 2.2.2:

$p\backslash b$	16	32	64	128	256	512	1024
0.05	10	6	4	2	2	2	1
0.10	12	7	5	3	3	3	2
0.25	16	11	8	7	6	6	6
0.50	32	24	20	18	17	17	17
0.75	80	64	56	56	50	49	49

Table 4.1: Overhead bytes per packets for different values of p and b

	loss rate	mean burt length	b	p	$\delta(p,b)$
Example 1	5%	5	100	0.27	8
Example 2	60%	10	512	0.73	45
Example 3	10%	3	32	0.47	22
Example 4	10%	50	512	0.50	18
Example 5	80%	10	200	0.905	160
Example 6	5%	5	1024	0.1	2

Table 4.2: A few case studies.

- **CASE 1:** We propose to use the ECU scheme since the sensors may have limited memory, thus the verification delay of 10 seconds allows us to use a block of 100 packets (200/2 since a block is authenticated by the next one). Given the drop rate and the average length of bursts, we constructed a corresponding 2 state Markov chain with $r = 0.010526$, $q = 0.2$ and simulated it over 10000 blocks of 100 packets. For Markov chain simulation techniques we referred to HÄGGSTRÖM[Häg02]. We found that 99% of those blocks experienced a loss less than 27 packets, thus we decided to choose $p = 0.27$. The overhead[1] per packet is then only $\delta(100, 0.27) = 8$ bytes !

- **CASE 2:** We propose again the EC1 scheme and, because of the verification delay, we have to limit b to 512 packets. We simulated the corresponding Markov model over 10000 blocks and found that 99% of those blocks experienced a loss of less than 375 packets. We decided to choose $p = 0.73 = 375/512$, which gives us an overhead per packet of $\delta(512, 0.73) = 45$ bytes.

As a complement to the two proposed scenarios above, Table 4.2 shows a few of our other simulation results, following the same approach as above for different average burst loss lengths and loss rates. Example 1 and 2 simply repeat the two case scenarios above. Example 3 shows that with a small block size, parameter p is significantly higher than the network loss rate. Similarly, an extreme average burst length increases the value of p as shown in example 4. Finally we have two extreme examples of the parameters of our scheme: first, a very lossy network that requires 160 bytes of overhead per packet, which is more than the size of a public key signature, and finally we have an ideal case, with a small loss and a long block size, which gives us a surprisingly low overhead per packet of 2 bytes !

[1] If we had chosen the EC1 scheme instead, we would have $b = 200$, $p = 0.2$ and $\delta(b,p) = 5$.

4.4.3 Denial of Service

In their work presenting offline stream authentication techniques[MS01], Miner and Staddon briefly discuss the use of Erasure Code techniques as an additional robustness mechanism. Their objective is different from ours here, since they use Erasure Codes as a mean to "reinforce" their "hash and MAC chain" rather than as a substitute as we do. However, they make an interesting remark that Erasure Code techniques may be vulnerable to "Denial of Service" since an adversary who modifies the transmitted parities may render the authentication of the received packets impossible. We observe that this remark is also valid for our scheme: if a some packets are lost and if an adversary modifies the tags piggy-backed on the data packets the verification process may not function properly if the decoding algorithm requires those parities. However, we would like to highlight that with or without erasure codes, this type of vulnerability exists in almost all stream authentication schemes. If an adversary modifies a single bit of the signature packets then the corresponding block is not verifiable ! The only exception is perhaps the Wong and Lam hash tree scheme[WL99], which truly allows packets to be verified individually by transmitting a copy of the signature with *every* packet.

Consequently we believe that this issue is relevant to almost all stream authentication schemes and is not specific to the use of Erasure Codes.

4.5 Comparison

For online lossy stream authentication, following the conclusion of the previous chapter, we consider 4 possible alternatives: TESLA, Hash Chains, EMSS and our own scheme.

The comparison with the TELSA scheme is only partially relevant here because TESLA does not provide nonrepudiation and requires an additional time-based security condition, which is absent in the 3 other schemes. Nevertheless, we have chosen to include the TESLA scheme here for completeness.

Robustness

TESLA is clearly the best scheme in terms of robustness, while the 3 other schemes offer rather equivalent levels of robustness, with a little advantage for our scheme since it implicitly provides the signature with the same level of robustness as data packets.

Cost

Though TESLA may seem at first glance much less expensive than the 3 other schemes, we remark that this depends on the level of desired joinability. Since joinability in TESLA requires the periodic transmission of commitment signature packets, the cost of TESLA will approach the cost of the 3 other schemes, if frequent commitments are broadcasted. While Hash Chains and EMSS can be considered roughly equivalent, our own scheme is more expensive since it requires additional error correcting code mechanisms.

Joinability

The level of joinability of TESLA depends on the transmission of commitments. Nonetheless, once a recipient holds a commitment for the stream he can trace back any packet to that commitment. Consequently once a commitment is received, the stream offers joinability on any packet. Hash Chains and EMSS are joinable on every packet even if packets are signed over a block. Our scheme is designed to offer joinability on a block boundary.

Buffering

If we consider the ECU, the unbuffered version of our scheme, all 4 schemes have unbuffered server side packet authentication. All 4 schemes require the storage a hashes or MACs on the server, though in this regard TESLA is clearly more demanding since it requires the storage of set of keys representing the whole stream. It is however, reasonable to assume that servers in large multicast commercial applications will be provided with sufficient resources to store these keys.

Latency

In TESLA the latency depends on the parameter d as described in section 3.3. In the remaining 3 proposals it depends mainly on the block size b, but also on the reliable delivery of the signature. Our ECU scheme has a maximum latency of $2.b$ packets, while our EC1 scheme has a maximum latency of b packets. In both EMSS and Hash Chains, depending on the signature retransmission policy, the maximum latency will be at least b packets and may increase significantly with higher losses in the network if multiple signature retransmissions are necessary. Note the EC2 is the only scheme here which provides no latency (at the cost of increased sever side buffering).

Overhead

TESLA is the only scheme which has a constant and low overhead no matter what the network losses are. The transmission of the commitment increases the overhead if periodic commitment transmissions are used every r packets to allow joinability. However, as the receiver only needs to get one commitment to authenticate the stream while other schemes require a signature to be delivered for every block of b packets, it does not make sense to compare TESLA with the 3 other schemes, by assuming $r = b$ for example. Nonetheless we have a lower bound of one MAC and one MAC key on the communication overhead of TESLA, taking a 80 bit MAC with a 128 bit output such as MD5-MAC, we have at least 26 bytes.

Now, we turn our attention to the 3 nonrepudiation schemes. We reproduce the results of our two test cases, using readjusted hash sizes for EMSS in order to make the comparison fair :

	CASE 1	CASE 2
Hash Chains	38	58
EMSS	34	84
Our scheme	8	45
TESLA	26+	26+

In the two test cases, our scheme has a lower overhead than Hash Chains and EMSS. TESLA has a better overhead in the very lossy network in CASE 2. But perhaps more surprisingly our scheme achieves an overhead that is clearly lower than TESLA in CASE 1. In fact, in situations where the network does not have excessive losses, our scheme offers the lowest overhead. This can be explained by the fact that we replaced the hash chaining mechanism with erasure codes, consequently the lower optimal limits that Golle and Modadugu used for hash chains do not apply in our setting. Indeed, Hash Chains carry at least two hashes on average, which amounts to 32 bytes for MD5, and TESLA has an incompressible 26 byte overhead, while the table above shows that our scheme can achieve substantially lower overhead, as low as a few bytes.

Conclusion

In this chapter we proposed a new approach to online lossy stream authentication with nonrepudiation, which offers joinability on block boundaries. Where previous proposals used hash linking, we use erasure codes to achieve a lower overhead per packet. Moreover, we propose a concrete mechanism describing how to transmit the authentication information as well as the signature associated to a block with equivalent recovery probabilities. We proposed buffered and unbuffered variations of our scheme which offer an interesting alternative to other live stream authentication mechanism in many situations.

We believe that our scheme offers some interesting perspectives for future work. In particular, it would be interesting to deepen the analysis of the choice of parameter p, with a more analytical model.

Part II

Confidentiality

Chapter 5

Definitions and Requirements

Introduction

This part of the thesis addresses confidentiality for a large dynamic multicast group. Providing confidentiality in an open environment such as the Internet requires the use of encryption since it is assumed that adversaries are able to eavesdrop on the communication links. To clarify our discussion of encryption in the context of multicast we will use the following terminology:

Source: We call *source* an entity that wishes to transmit a certain *content* to a selected set of chosen recipients using multicast.

Content: We always use the words *multicast content* to refer to the actual cleartext data that the source wishes to transmit securely over a multicast channel.

Recipient: We call *recipient* any entity capable of receiving multicast packets from a certain group, regardless of its capacity to actually decrypt the multicast packets to access the content. The only limit to the set of recipients is dictated by multicast routing protocol restrictions, mainly the scope of the multicast group which is limited by the TTL[1] selected by the source.

Member: We call *member*, a selected recipient which has been given cryptographic keys that enable it to access the *content* of the received multicast packets.

Membership Manager: A *membership manager* describes the entity (or entities) that grants or refuses membership to recipients.

The primary focus of this part of the thesis is to analyze and propose methods that allow a dynamic set of members to access the multicast content transmitted by a source while disallowing other recipients to do so.

Typical large scale multicast application scenarios that require confidentiality services are:

- Pay-per-view TV,

[1] TTL: Time To Live field of the IP packet.

- High quality streaming audio,
- The distribution of software updates,
- News feeds and stock quote distribution,

All these scenarios involve one of very few sources and potentially a very large amount of members or clients. In this thesis we have chosen to put an emphasis on solutions guided towards these scenarios. In particular we do not deal with dynamic peer groups [BD95, STW96, STW98], which involves secure *n-to-n* communications in much smaller groups through the cooperation of the entities in the group. Though this work deals with a selective broadcasting medium, we do not either aim to provide a solution to the problem of *broadcast encryption* [FN93] where each message is encrypted for an arbitrary and potentially disjoint subset of a larger known group. In particular, in our setting, the set of potential members is not necessarily known a priori. Consequently, we assume at least a minimal unicast back channel between the recipients and a membership manager. This channel is used at least once when a recipient sends a request to a membership manager to become a member of the group.

5.1 The Security of Dynamic Groups

We qualify the set of members as *dynamic* because we consider the general case where the set of members evolves through time as members are *added* or *removed* from the group during a session. We define *add* and *remove* operations as follows:

Add: When a *recipient* becomes a *member* of the secure multicast group by receiving proper cryptographic access parameters, we say that he is *added* to the group. To be added to a group, a recipient needs to contact a membership manager through a secure authenticated unicast channel. If the recipient is allowed to become a member then it receives keys and other related parameters that are necessary to start accessing the multicast content. The membership policy is application dependent and may be subordinated to other issues such as payment and billing, which are all beyond the scope of this thesis.

Remove: We say that a *member* is removed from the group, when its ability to access the encrypted multicast content is suppressed by the membership manager. A membership manager can decide to remove a member from the group at any time during the session or at least on a certain interval boundary that is application dependent (a packet, a frame, a quantity of bytes or time...).

In many publications, add and remove operations are referred as *join* and *leave* operations, respectively. We prefer the former terminology because it highlights more clearly the fact that providing access to the multicast content is ultimately the decision of a membership manager rather than the recipient itself. We will restrict the terms *join* and *leave* for multicast routing, to describe the fact that an entity requests or relinquishes receiving multicast packets, independently of any security concerns.

All the application scenarios we described in the previous section involve dynamic groups where members are added or removed from the member group. A dynamic set of members implies two security requirements that are not found in traditional secure unicast communications:

backward secrecy and *forward secrecy.*

Backward secrecy defines the requirement that an added member should be able to access multicast content transmitted before it became a member. This means that if a recipient records encrypted multicast data sent prior to the time it is added to the group, it should not be able to decrypt it once it becomes a member. Similarly, *forward secrecy*[2] defines the requirement that a member leaving the group should not be able to further access multicast content.

5.2 Algorithmic Requirements

The forward and backward secrecy requirements have an immediate consequence: the parameters of the encryption algorithm that protects the multicast content must be changed each time a member is added or removed from the group. We must simultaneously allow members to infer the new parameters used to access the multicast data while disallowing other recipients to do so. This parameter change must be done in a scalable way, independently of the group size, which can be a real challenge, as illustrated in the simple financial content provider scenario of Chapter 1. The scalability issues related to multicast key management in dynamic groups were first highlighted in the IOLUS[Mit97] framework by S. MITTRA, who identified two generic problems:

1. The "one does not equal n" failure which occurs when the group cannot be treated as a whole but instead as a set of individuals with distinct demands.

2. The "one affects all" failure which occurs when the actions of a member affect the whole group.

Our experience with many multicast proposals has prompted us to refine these definitions to propose the following two requirements:

Processing scalability: The cost supported by an individual component, be it the source, an intermediary forwarding component or a member, should scale to any potential group size and membership duration.

Membership robustness: Members must be able to access the content of received multicast packets *as soon* as they are received.

[2] The term "forward secrecy" is used here in the context of multicast security and should not be confused with *perfect forward secrecy* in unicast communication, which deals with the security of past sessions when a session key is compromised.

The *processing scalability* requirement was informally illustrated in our simple scenario in Chapter 1, where we tried to use unicast techniques in a multicast setting to provide confidentiality. To remove a member from a group of N members it required the source (or the membership manager) to send out a new global key encrypted with $(N-1)$ different private keys to replace the previous global key used to access the content in order to guaranty forward secrecy. Clearly, this method does not offer processing scalability.

The best effort nature of the Internet is a potential factor that greatly affects membership robustness. Consider a scenario in which the membership manager is required to broadcast a single value R to the whole group each time a member is removed or added, in order for members to update their decryption parameters to continue accessing the multicast content. As already noted in Chapter 2 for authentication, many multicast applications can tolerate packet losses. However, if some members experience the loss or the delay of R they will not be able to update the keys needed to access the multicast content, thus the newly received content packets will be worthless. Consequently, in such a scenario, add and remove operations may impair membership robustness.

5.3 Collusion, Containment and Trust.

While the main goal of multicast encryption in dynamic group is to find a protocol which satisfies both the security requirements in section 5.1 and the algorithmic requirements in section 5.2, this is still not quite sufficient to provide security in a *large* group. A factor that is often overlooked in most secure multicast proposals is the potential compromise of a member. Indeed, as the member group becomes larger, the probability of member key exposure increases. In a sufficiently large group, there is no doubt that an exposure will occur. These exposures may be done intentionally by a member: it could send its security parameters to another recipient, or they may be unintentional: a "hacker" may steal the parameters held by a member and publish them in a newsgroup or on a web page. Thus, one of the important concepts we put forward in this thesis is that, since we cannot avoid security exposures, we need to limit the impact of such exposures, a property that we call *containment.*

A second issue is *collusion*: if a few members collude together they should not be able to elevate their privileges beyond the sum of their own privileges. For example, consider a scenario where each member holds secret keys that allow them temporary access to the group. If these members are able to achieve unlimited access by exchanging their secrets and making some computations, then the scheme is clearly weak and will be subverted quickly in a large group.

Some multicast schemes, including our own proposals, use intermediary elements in the network, such as routers, subgroup servers or proxies as participants in the security protocol. These elements are external to the member group and most likely not always fully controlled by the content provider, the members or a membership manager. In fact, it is quite possible that these intermediary elements will be shared between several secure multicast groups with different member sets. Moreover, in some situations, these elements are numerous enough such that the compromise of one or a few of these entities is probable. Consequently, while it is quite reasonable to assume that the source as well as the few other entities such as a membership manager are secure, this is not realistic for other network resources. Thus we need to *limit the trust* that we put in these intermediary elements if they are actively used in our protocols.

5.4 Summary

We started this chapter by describing the basic requirements of multicast confidentiality in a large dynamic group. Though the problem may seem simple at first, confidentiality brings out significantly more requirements in the multicast setting than in the unicast setting. We summarize these requirements here as a reference for later discussion, in particular to analyze current multicast proposals in the following chapters.

Security Requirements:

Requirement 5.1 Data Confidentiality: *the multicast content should be only accessible to members.*

Requirement 5.2 Backward and Forward Secrecy: *A new (resp. past) member should not have access to past (resp. future) data exchanged by the group members.*

Requirement 5.3 Collusion Resistance: *A set of members which exchange their secrets cannot gain additional privileges.*

Requirement 5.4 Containment: *The compromise of one member should not cause the compromise of the entire group.*

Requirement 5.5 Limited Intermediary Trust: *Intermediary elements in the network should not be trusted with the security of the group.*

Strictly speaking, the latter requirement (5.5) is only relevant to protocols which use intermediary elements to actively participate in the security of the group.

Algorithmic requirements:

Requirement 5.6 Processing Scalability: *The cost supported by an individual component, be it the source, an intermediary forwarding component or a member, should scale to any potential group size and membership duration.*

Requirement 5.7 Membership Robustness: *Members should be able to access the content of received multicast packets as soon as they are received.*

Organization of the Following Chapters

The next chapter reviews previously proposed work for multicast confidentiality. As we will see, none of those schemes satisfy all the requirements above. In chapters 7 and 8, we propose two algorithms which are based on the use of untrusted intermediary elements in the network to achieve the security requirements we established, and in particular containment.

Chapter 6

An Overview of Multicast Confidentiality Algorithms

This chapter is divided in three main sections, each dedicated to a different type of multicast confidentiality algorithm. The first section reviews the popular Logical Key Hierarchy construction and the second section describes a re-encryption tree construction: IOLUS. The third section reviews the MARKS[Bri99] approach which in not fully a multicast confidentiality scheme as defined in Chapter 5 in the sense that the removal time of a member needs to be chosen prior to the time it joins the groups and cannot be changed. Despite this limitation it can still be used in some scenarios and we include it here for completeness because it offers remarkable processing scalability and membership robustness.

In the remaining of this chapter, we denote $E_K(x)$ as the encryption of message x with key K with a standard symmetric encryption technique[BDJR97, oST01].

6.1 LKH: The Logical Key Hierarchy

The Logical Key Hierarchy was independently proposed by Wong et al. [WGL98] and Wallner et al.[WHA98].

6.1.1 Construction

Consider a set of n members $\{R_1, ..., R_n\}$. We build a balanced (or almost balanced) binary tree with n leafs, with a one to one correspondence between a leaf L_i and a member R_i. A random key k_j is then attributed to each vertex j in the tree, including the root and the leafs themselves. This construction that we will call *logical key hierarchy* (LKH) is illustrated on figure 6.1 with a small set of 7 recipients.

Setup.

Each member R_i receives the set of keys corresponding to the path from the root of the tree to its corresponding leaf L_i. The root of the tree which is known by all members is called the *Traffic Encryption Key* (TEK) and represents the symmetric

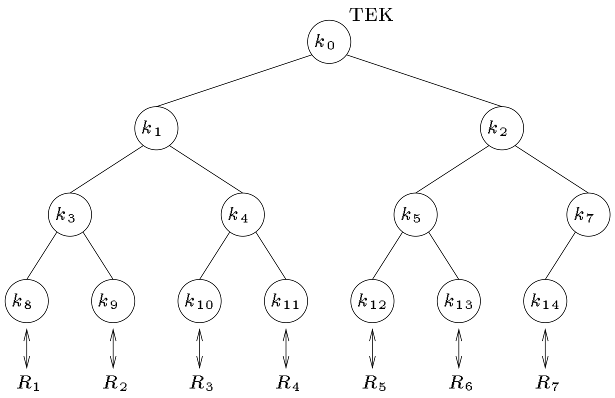

Figure 6.1: Basic key graph with 7 members

encryption key used to protect the content distributed over the multicast channel. The remaining keys in the tree are often called *Key Encryption Keys* and are used to update the TEK to guaranty backward and forward secrecy. Referring to our example in figure 6.1, member R_1 would receive the key set $\{k_0, k_1, k_3, k_8\}$ where k_0 represents the TEK and $\{k_1, k_3, k_8\}$ the KEKs.

6.1.2 Usage

Let us denote $\mathcal{L}(k_i)$ (respectively $\mathcal{R}(k_i)$) the predicate that returns *true* if the node representing key k_i has a left child (respectively a right child). Furthermore we will denote $\mathcal{LK}(k_i)$ the key held in the left child of the node representing k_i if it exists, and we similarly define $\mathcal{RK}(k_i)$ for the right child.

To *remove* a member R_i from the group:

All keys representing the path from the root to the the leaf L_i corresponding to departing member R_i are invalidated. The leaf L_i corresponding to the departing member is removed from the tree. All the remaining invalidated keys $\{k_1, ..., k_l\}$ in the path from the root to the former leaf are replaced with new random values $\{k'_1, ..., k'_l\}$. For convenience consider $\{k'_1, ..., k'_l\}$ to be ordered by *decreasing* depth in the tree. To allow the remaining members in the tree to update their keys, the membership manager proceeds as follows:

Algorithm 6.1 *LKH update*

```
For $i = 1, ..., (l-1)$ do
    if $\mathcal{L}(k'_i)$ then multicast $E_{\mathcal{LK}(k'_i)}(k'_{(i-1)})$
    if $\mathcal{R}(k'_i)$ then multicast $E_{\mathcal{RK}(k'_i)}(k'_{(i-1)})$
```

None of the keys that are used for encryption in the previous algorithm are known by the removed member. However, all the remaining members will be able to recover enough keys to update their own key set.

As illustrated on figure 6.2, if member R_4 leaves, then the membership manager needs to broadcast $E_{k_{10}}(k'_4)$, $E_{k_3}(k'_1)$, $E_{k'_4}(k'_1)$, $E_{k'_1}(k'_0)$ and $k_{k_2}(k'_0)$.

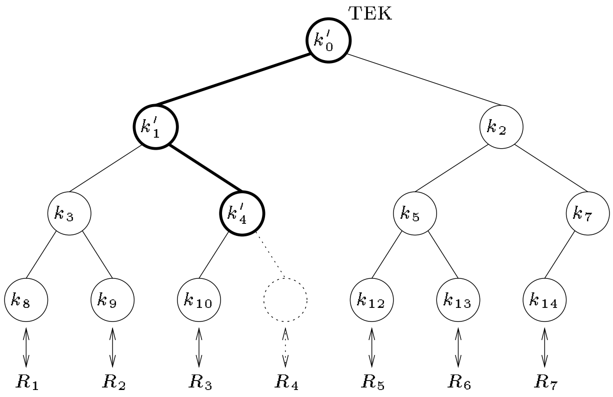

Figure 6.2: LKH: A member leaving the group.

To *add* a member to the group:

When a member is added to the group, the tree is extended with an additional node. If all leafs in the tree are already attributed we can take a leaf L_i and create two children: the left child is assigned to the member associated with the former leaf L_i and the right child is assigned to the newly added member. Alternatively, we can simply design the tree to be deep enough to accommodate the potential maximum number of members.

Assuming that the new member corresponds to leaf L_i, the sender makes all the keys in the nodes on the path from L_i to the root invalid. A random key is assigned to L_i and transmitted to the added member with a secure unicast channel. All other nodes in the path are updated with the same algorithm that was used above to remove a member from the group.

Key Usage Summary.

The scheme distinguishes two types of keys: a TEK and KEK. The TEK is the encryption key that protects the multicast content. It is changed each time a member is added or removed from the group. The KEKs are used to distribute and update the TEK for all members. Until a valid KEK set is explicitly invalidated by the membership manager, it can be maintained in a valid state provided that all update messages are received. Consequently a valid KEK set represents membership in a group and provides long term access to the group, while the TEK itself only represents a short term access to the content.

6.1.3 Algorithmic Properties of the Scheme

Considering a set of at most n members, the basic LKH scheme requires each recipient to store $\lceil \log_2(n) \rceil + 1$ keys. Each addition or removal of a member requires the broadcast of $2.\lceil \log_2(n) \rceil - 1$ messages (alternatively a single message with all key update messages grouped together). Thus, the processing load of the components and the key message sizes increase logarithmically with the group size, which allows to achieve good processing scalability (req. 5.6).

However, if a member fails to receive just a single key update message because of a network loss, then it cannot decrypt the new content. Losses have a worsened impact because subsequent key updates will often depend on previous ones. Consequently a member cannot always resynchronizes its parameters on a new key update message if it lost one of the previous key update messages. A member who didn't receive a KEK update message might have to re-register with the membership manager through an authenticated secure unicast channel to receive the proper keys. If many members do this at the same time we may run into scalability issues. Consequently in this basic version of the scheme, both add and remove operations do not provide membership robustness (req. 5.7) in the presence of losses or strong key update delays.

6.1.4 Improvements

The basic scheme we described previously has been constantly improved by several authors. We summarize these contributions here.

No-message Member Addition

An interesting improvement was proposed by Caronni et al. [CWSP98] which eliminates the need to multicast a key update message when a member is added to the group. In their proposal, the set $\{k_1, ..., k_l\}$ of invalidated keys represented on the path from the root of the tree are replaced with a set $\{k'_1, ..., k'_l\}$ where each key k'_i is the result of a one way function applied to the previous key, that is $k'_i = F(k_i)$ where F is a publicly known one-way function. This allows members to compute the new keys from the previous ones, and the one-way property of F disallows the new member to compute the previously used keys. The set $\{k'_1, ..., k'_l\}$ is transmitted to the newly added member through a secure unicast channel. In their work Caronni et al. use a counter (a sub-version number) for the TEK and KEKs as ancillary information in the multicast data packets to keep track of the number of times the one-way function was applied (this naturally adds a few bytes of overhead to each packet). Since adding a member does not require the transmission of a multicast message anymore, membership robustness (req. 5.7) is always assured during this operation, and the problem is now restricted to the removal of a member.

Halving the Update Overhead

McGrew and Sherman were the first to propose a method to halve the communication overhead of add and remove operations, with a construction called OFT (One Way Function Trees, [MS98]). This further increases the processing scalability of the protocol. We will not describe the OFT construction here, instead we will focus on the more recent ELK approach presented below, which shares some similarity with OFT.

More recently Canneti et al. have proposed a method[CGI$^+$99] that achieves the same reduction and that is compatible with the "no-message-member-addition" we described above. We will describe the core idea behind their protocol here. Assume that f is a one way function, and let $f^k(x)$ denote k successive applications of f to x, where $f^0(x) = x$. As usual, when a member is removed from the group, the remaining keys $\{k_1, ..., k_l\}$ represented on the path from the root to the departing leaf need to be changed to a new set $\{k'_1, ..., k'_l\}$. Here, instead of choosing all these keys at random, the membership manager chooses only one random value r and sets

$$k'_l \leftarrow r,\ k'_{(l-1)} \leftarrow f^1(r),\ k'_{(l-2)} \leftarrow f^2(r),\ ...,\ k'_1 \leftarrow f^{(l-1)}(r)$$

The membership manager then encrypts each k'_i with the key represented by the child node of k'_i that is **not** in the invalidated path. Thus each individual k'_i is encrypted only once as opposed to the original LKH protocol in algorithm 6.1 that encrypted each key with both child node keys. Because of the relationship between the keys created by f, the nodes can still reconstruct the set of keys they need by applying f.

Example: Recall figure 6.2 in which member R_4 is removed from the group, the membership manager will compute a random value r and send $E_{k_{10}}(r)$, $E_{k_3}(f(r))$, $E_{k_2}(f(f(r)))$.

- Member R_3 decrypt $k'_{10} = r$ and computes $k'_4 = f(k'_{10})$, $k'_1 = f(f(k'_{10}))$ as well as $k'_0 = f(f(f(k'_{10})))$.
- Members R_1 and R_2 decrypt k'_1 and compute $k'_0 = f(k'_1)$.
- Members R_5, R_6, R_7 decrypt k'_0.

Assuming that the tree is balanced, the communication overhead is at most $\lceil \log_2(n) \rceil$ keys instead of $\lceil 2.\log_2(n) - 1 \rceil$ keys as in the original LKH construction.

Increasing Reliability

Clearly, if the network was perfectly reliable over the whole multicast group the removal of a member would not introduce any membership robustness issues in the LKH scheme (req. 5.7). However, multicast uses an unreliable transport protocol such as UDP. Consequently, increasing the reliability of the leave operation is crucial if we want to reduce membership robustness issues (req. 5.7).

Wong and Lam implemented their LKH[WGL98] protocol over IP-Multicast in Keystone[WL00], and use FEC (Forward Error Correction) to increase the reliability of the addition and removal operations (since they do not use the no-message-member-addition method). Recall that when a member is added or removed from the group in the basic scheme, a set of keys $\{k'_1, k'_2, ..., k'_l\}$ is encrypted with other keys in the tree and transmitted to the group. In the Keystone scheme, these encrypted keys are represented over s packets to which we add r parity packets. The receiver may reconstruct the encrypted key set if he recovers any subset of s packets out of the $(s + r)$ transmitted packets. Since we do not have an absolute guaranty that s packets out of $(s+r)$ packets will always arrive, they combine this mechanism with a unicast re-synchronization mechanism that allows a member to contact a "key cache" server in order to update its keys if losses exceed the correction capacity of the FEC. In their work, they consider a 10-20% loss rate and they show that adding redundancy can substantially increase the likelihood that a member will be able to update its keys. However, as noted by PERRIG ET AL.[PST01] in criticism of the Keystone scheme, the authors assume independent packet losses to compute the key recovery probabilities, while it has been shown that Internet losses occur in bursts (For further discussion and references related to Internet loss patterns and FEC we refer the reader to section 2.2 which discusses these issues in the context of multicast authentication.)

PERRIG ET AL. proposed a protocol called ELK[PST01] which combines previously proposed ideas such as "no message member addition" and halving the update overhead, with new tricks to increase the reliability of the KEK update operation. The main idea of their work stands in two points:

- First, send a key update message U with reduced communication overhead.
- Second, send many small *hints* that allow a member to recover updated keys if U is lost. The *hints* are smaller than U but require a much higher computational cost from the recipient.

We will give a slightly simplified description of their protocol, omitting some cryptographic key derivation issues that are described in detail in their work[PST01] but are not useful to understand the methods employed here.

Reducing the communication overhead. To reduce the communication overhead of a key update they construct new keys based on contributions from the two children nodes. Consider for example a node containing the key k_i on the path from the root to a leaf representing a removed member that needs to be updated to a new value k'_i. Let k_R (*resp.* k_L) define the key held by the right (*resp. left*) child of the node associated to k_i. Denote $F_k^{\langle p \to m \rangle}(x)$ as a pseudo-random function that takes a p bit value x as input and outputs m bits using a key k. In the ELK construction, the keys in a node are p bits long and are derived from p_1 bits of the left child and p_2 from the right child. To update k_i to k'_i we first derive the right and left contributions C_R and C_L from the old key k_i as follows:

$$C_L \leftarrow F_{k_L}^{\langle p \to p_1 \rangle}(k_i)$$

$$C_R \leftarrow F_{k_R}^{\langle p \to p_2 \rangle}(k_i)$$

Now we compute the new key k'_i by assembling these two contributions to derive k'_i:

$$k'_i \leftarrow F_{(C_L || C_R)}^{\langle p \to p \rangle}(k_i)$$

Consequently, to derive k'_i from k_i the members find themselves in two possible situations:

- They know k_L and thus C_L, and they need to receive C_R to compute k'_i, *or,*

- They know k_R and thus C_R, and they need to receive C_L to compute k'_i.

Consequently, the membership manager will send $E_{k_L}(C_R)$ and $E_{k_R}(C_L)$ to the group where E denotes a stream cipher rather than a block cipher (using the old root key or TEK as Initial Vector). Using a stream cipher allows the membership manager to send $\langle E_{k_L}(C_R), E_{k_R}(C_L)\rangle$ over exactly p_1+p_2 bits. A full key update will thus require the broadcast of $log_2(n)$ chunks of p_1+p_2 bits. Considering $p_1+p_2 \leq p$ this amounts to having at most 50% of the communication overhead of the original LKH protocol.

Using Hints to increase reliability. Consider the case where $p_1 < p_2$ and where p_1 is small enough such that computing 2^{p_1} symmetric cryptographic computations remains feasible in a reasonably short time. In this situation it's possible to construct an even smaller key update procedure that the authors of ELK call a *hint,* which is defined as the pair $\{V_{k'_i}, E_{k_L}(h)\}$ where $V_{k'_i}$ is a p_3 bit cryptographic checksum computed as follows :

$$V'_i \leftarrow F_{k'}^{\langle p \to p_3 \rangle}(0)$$

With $LSB^{\langle p_2-p_1\rangle}(x)$ being the function that returns the (p_2-p_1) least significant bits of x, h is defined as:

$$h \leftarrow LSB^{\langle p_2-p_1\rangle}(F_{k_R}^{\langle n \to p_2\rangle}(k_i)) = LSB^{\langle p_2-p_1\rangle}(C_R)$$

The hint $\{V_{k'_i}, E_{k_L}(h)\}$ of which the second member h is encrypted with k_L, is broadcasted to the group after the initial key update we described previously. The hint $\{V_{k'_i}, E_{K_L}(h)\}$ can be used to recover the value k'_i in a node. Again, depending on left or right considerations, there are two ways to use the hint:

- Members who know k_L can decrypt h, thus they can recover the $(p_2 - p_1)$ least significant bits of C_R. Since they also know C_L they have to make and exhaustive search for the n_1 missing bits of C_R to recover a value of the form $C_L||\widetilde{C_R}$. For each candidate $C_L||\widetilde{C_R}$ they compute the corresponding $\widetilde{k'_i}$ as $\widetilde{k'_i} \leftarrow F_{(C_L||\widetilde{C_R})}^{\langle p \to p\rangle}(k_i)$ and use the checksum $V_{k'_i}$ to verify that the candidate is good, otherwise they continue the exhaustive search.

- Members who know k_R will perform an exhaustive search through the 2^{n_1} values that k_L can take. For each potential k_L value they compute the corresponding candidate for k'_i. They use the checksum $V_{k'_i}$ to validate the right candidate.

In practice the authors of ELK suggested in a example to use the values $p_1 = 16$, $p_2 = 35$, $p_3 = 17$ and $n = 64$ using the RC5 cipher[Riv95] both for encryption and also to construct a CBC-MAC based pseudo random function. The authors of ELK implemented this example on a 800Mhz Pentium machine to achieve a high number of computational operations per second.

Aggregating Member Additions and Removals.

It should be clear from the description of the LKH algorithm that frequent membership changes increase the workload of all the members of the group and require the membership manager to broadcast a proportional number of key update messages. In situations where the group has a highly dynamic set of member, the membership manager will have to deal with both membership changes and re-synchronization messages for recipients who failed to update their keys correctly, which creates a potential bottleneck. Consequently several authors [CEK$^+$99, LYGL01] have proposed methods to reduce this problem by combining several removal operations into one to achieve a lower communication overhead than the sum of these individual removal operations. In a related but more theoretical approach, [PB99] proposes to organize members in the tree according to their removal probability to increase the efficiency of batch removals and additions.

6.1.5 LKH: Summary and Conclusion

The LKH construction is an interesting approach that satisfies the following requirements of multicast confidentiality:

- *Data confidentiality* (req. 5.1).
- *Backward and forward secrecy* (req. 5.2).
- *Collusion Resistance*[1] (req. 5.3).
- *Processing scalability* (req. 5.6).

Beyond classical multicast forwarding mechanisms, it does not require any additional contribution from the intermediary elements in the network, which clearly also satisfies requirement 5.5. While some authors have tried to increase the processing scalability of the scheme, the main concern about LKH scheme is membership robustness. Indeed, each time a member is added or removed from the group, all the members need to receive a key update message. If that message is not received by some members, it will disallow them to access the multicast content and it may also disallow them to access further key update messages. The Keystone framework [WL00] uses FEC to increase the likelihood that the key update message will be received by the members, while the ELK framework relies on small “hints” that can be used by a member to derive missing keys in combination with computational tradeoffs.

There remains one requirements that none of the LKH protocol addresses: Containment (req. 5.4). Indeed, each set of keys held by a member can be used anywhere in the network within the scope of the multicast group to access the encrypted content.

[1] An exception is the *flat key management scheme* presented in [CWSP98], which can be defeated by a coalition of only 2 members.

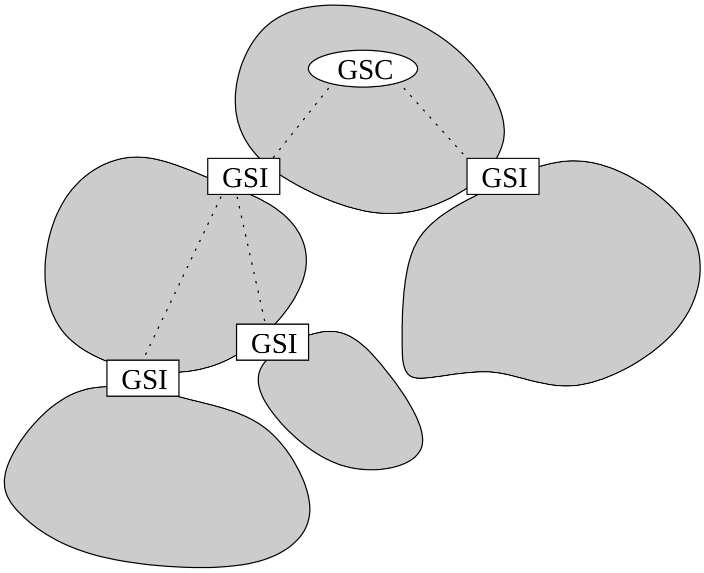

Figure 6.3: IOLUS

6.2 Re-Encryption Trees

In 1997, Mittra proposed the Iolus framework, which partitions the group of members into many subsets and creates a tree hierarchy between them. Each subset of members is managed by an distinct entity called a Group Security Intermediary (GSI). The central or top-level subgroup in the hierarchy contains the central group manager called Group Security Center (GSC), as illustrated on the example on figure 6.3.

6.2.1 Basic scheme

The GSC produces data encrypted with a key K_1 and multicasts it to a chosen group address G_{A1}. The GSI entities that are direct children of the GSC in the hierarchy then decrypt the data and re-encrypt it with another key K_i, and broadcast it to their own multicast group G_{Ai}. This mechanism is applied recursively all the way down the tree hierarchy. Consequently, this construction can be viewed as a tree network where each node decrypts data received from its parent and re-encrypts it for its children. In this dissertation we refer to this type of approach as "re-encryption trees".

Each node or GSI only needs to know the key from its parent in the hierarchy and its own subgroup key. Members in a subgroup are managed by a single GSI and only know the GSI's key K_i.

Removing a member.

When a member is removed from a subgroup, the GSI chooses a new subgroup key K_i' and uses several authenticated secure unicast channels to send the new key to the remaining members in the subgroup (alternatively one long multicast message corresponding to the concatenation of these unicast messages).

Adding a member to the group.

When a member is added to a subgroup we can proceed in a similar way as we did to remove a member, except that we also send the new subgroup key $K_i^\prime$ to the added member. An improvement in terms of overhead is to encrypt the new subgroup key $K_i^\prime$ with the old one K_i and multicast the message $E_{K_i}(K_i^\prime)$ to the subgroup while transmitting $K_i^\prime$ to the added member on a separate authenticated secure unicast channel. However, this improvement is less secure because it creates a potential chain between subsequent keys, thus the exposure of a single key has a potentially stronger impact.

6.2.2 Key Distribution

The transformations in a hierarchical tree we describe above operate directly on the encryption of the multicast content. As an alternative, S. Mittra also suggests to use the IOLUS framework to distribute a global short term key. This key is used to encrypt content to be transmitted on a separate autonomous multicast channel, which can be based on a different multicast routing scheme.

Hardjono et al. proposed a re-encryption tree for key distribution in IGKMP[HCD00], the Intra-Domain Group Key Management Protocol. This protocol essentially describes a 2 level re-encryption tree that is used to distribute a common group key $\mathcal{K}$ to the group.

6.2.3 Analysis

This scheme provides Data Confidentiality (req. 5.1) as well as backward and forward secrecy (req. 5.2). Since keys in each subgroup are chosen independently, colluding members do not seem to gain additional privileges beyond the sum of their own access rights, thus this scheme is collusion resistant (req. 5.3).

In his work on IOLUS, S. Mittra did not specify a limit to the size of a subgroup. In practice, however, we need to assume that a subgroup contains no more than M members, otherwise processing scalability issues will arise in subgroups. The benefit of IOLUS is precisely to divide a large multicast group into subgroups that are small enough so that they do not exhibit the scalability issues of a large multicast group. If subgroups are small enough then this scheme will provide processing scalability, since computations and communication overhead will depend on M and not on the full group size.

A second benefit of subgrouping is that it provides stronger membership robustness (req. 5.7). When a member is added or removed from the group it only affects the other members that depend from the same GSI. Members in other subgroup are not even aware that a member was added or removed from the group. Consequently, the impact of a membership change is restricted to a subgroup of at most M members.

The third benefit of subgrouping is containment (req. 5.4). Since only at most M members of the same multicast group use the same key K_A to access the multicast data then the exposure of K_A will only have a local impact, limited to the subgroup G_A that uses the same key K_A. If the attacker is in another subgroup G_B, out of scope of G_A than the exposed key K_A will be useless unless he also manages to forward traffic from G_A to G_B.

The main drawback of this scheme is that it fully trusts all the intermediary elements to access the multicast content, which is contrary to requirement 5.5. Good processing scalability, membership robustness and containment all depend on a reasonably small value of M, the size of the subgroup. But if M is small than it increases the number of Groups Security Intermediaries needed. For large multicast groups, this means that each content provider will need to construct a large network of trusted intermediaries. This adds a significant cost for the content provider.

6.3 MARKS

The MARKS scheme was proposed by Briscoe in [Bri99], and features a subscription based approach to multicast confidentiality. Consider a content divided in n segments $\{S_1, ..., S_n\}$ each encrypted with a different key $\{k_1, ..., k_n\}$: the source simply broadcasts $E_{k_i}(S_i)$ for each $1 \leq i \leq n$. To provide access to a member R for segments S_j through $S_{(j+l)}$ the membership manager needs to provide R with the keys $\{k_j, k_{(j+1)}, ..., k_{(j+l)}\}$. Simply sending the list of keys creates an overhead that linearly increases with l, which does not provide processing scalability (req. 5.6). In his work, Briscoe provides a method to get around this limitation by constructing a one-way hash tree as follows.

6.3.1 The Hash Tree

Let n define the number of keys or segments used in the broadcast where $n = 2^d$. Let $\mathcal{L}$ and $\mathcal{R}$ define two one-way functions. For example, we can choose $\mathcal{L}$ (*resp.* $\mathcal{R}$) to be the left (*resp. right*) half of of a pseudo-random generator which doubles its input, or we can use two independently keyed hash functions [BCK96]. We initially choose a random value u_0 and we construct a balanced binary tree as follows:

- assign u_0 to the root.
- if a node is assigned a value u_i we assign to the left child of u_i the value $\mathcal{L}(u_i)$ and to the right child the value $\mathcal{R}(u_i)$.

The leafs of the tree ordered from left to right represent the keys $\{k_1, ..., k_n\}$ used to encrypt the segments $\{S_1, ..., S_n\}$.

The advantage of this construction is that any sequence $\{k_j, k_{(j+1)}..., k_{(j+l)}\}$ of l consecutive keys in $\{k_1, ..., k_n\}$ can be represented with at most $2.log_2(n) - 1$ values. Indeed, instead of describing a sequence of l keys by listing them in a sequential order, we can simply provide the points in the tree that can be used to derive the needed keys (and only those keys) by applying $\mathcal{R}$ and $\mathcal{L}$. This problem is quite

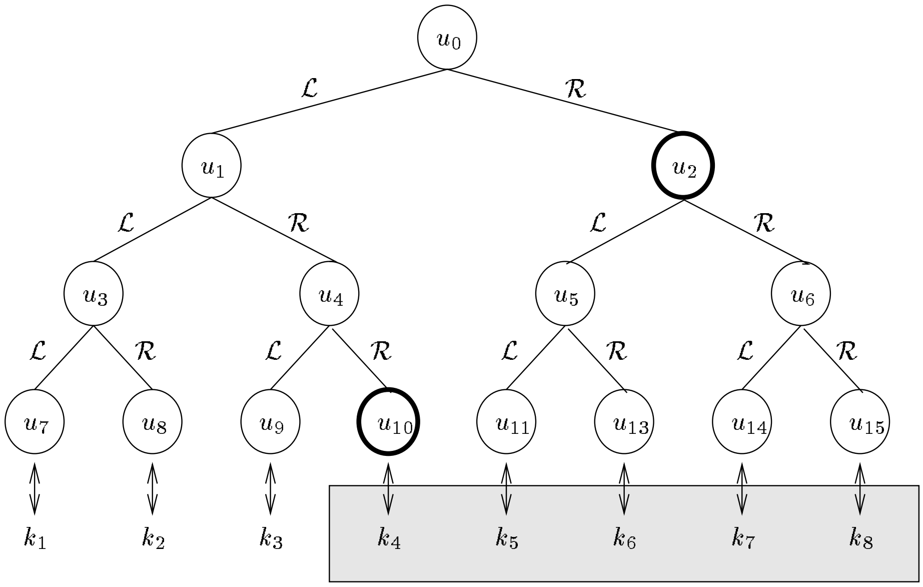

Figure 6.4: MARKS with an eight segment stream.

similar to finding IP network aggregations in CIDR[FLYV92]. Consider the example on Figure 6.4 where a member subscribes for segments 4 to 8. He receives u_2 and u_{10}, next he can efficiently compute $k_4 = u_{10}$, $k_5 = u_{11} = \mathcal{L}(\mathcal{L}(u_2))$, $k_6 = \mathcal{R}(\mathcal{L}(u_2))$, $k_7 = \mathcal{L}(\mathcal{R}(u_2))$, $k_8 = \mathcal{R}(\mathcal{R}(u_2))$.

6.3.2 Analysis and Conclusion

In the MARKS approach, once the membership manager provides a member with access to l consecutive segments $\mathcal{S} = [S_{j+1}, ..., S_{j+l}]$, it cannot revoke this access unless it redistributes new keys to all other members whose segments intersect with $\mathcal{S}$. In practice it would require the construction of a new key tree over the segments of the stream. Revocation is not possible here without running into strong processing scalability issues. However, there are some multicast applications that will be satisfied with a non revocable subscription based approach. Consider, for example, some Pay-per-view TV applications where the content length is know in advance and the clients pay before the show is broadcasted. For such applications MARKS offers benefits such as:

- Data confidentiality, backward and forward secrecy, collusion resistance and no intermediary trust.
- Processing Scalability.
- Perfect membership robustness.

The last characteristic is the strongest point of MARKS: it does not require any message to be broadcasted to anybody when a recipient's membership expires in the group. When a member is added to the group, he only needs to receives the keys corresponding to his subscription, which can be transmitted to him in advance.

Similarly to LKH, this scheme does not provide any containment. In terms of key exposure, the LKH has a little advantage over MARKS, because the membership can invalidate an exposed key easily, while MARKS by definition does not provide any practical way to do this.

6.4 Summary of Multicast Confidentiality Algorithms

The three types of multicast confidentiality schemes we reviewed in this chapter all have a few drawbacks. The LKH scheme does not offer containment and has some membership robustness issues though some progress has recently been made in that area. The re-encryption tree approach relies on a potentially large infrastructure of fully trusted intermediary elements. Finally the MARKS approach does not allow member revocation prior to the end of their subscription, neither does it offer any form of containment.

The next chapters provides two schemes that achieve containment in a similar way to IOLUS but which only rely on semi-trusted intermediary elements. These intermediaries are semi-trusted in the sense that they modify the transmitted data yet they do not have access to the distributed content, nor can they provide membership to other entities in the network.

Chapter 7

Untrusted Re-encryption Trees: Cipher Sequences.

Introduction

The multicast confidentiality framework presented in this chapter is based on [MP99, MP00]. We classify it in the same family as IOLUS: re-encryption trees. One of the main drawbacks of IOLUS was that it required to put full trust in all the intermediary elements in the network. The scheme we propose here solves this issue by proposing an untrusted re-encryption tree where intermediary elements are only trusted to perform certain transformations on the data without having actual means to access the content.

Our framework defines basic properties of a set of cryptographic functions that assure content confidentiality. Depending on the performance of the underlying algorithm, implementations of the framework may be suitable either for the encryption of bulk data or only for the encryption of short messages as required by key distribution. The framework is first validated with respect to the security and algorithmic requirements of chapter 5. Two different implementations of our framework are then discussed. Both solutions are based on asymmetric techniques: an extension of the ElGamal algorithm [Elg84] and a variation on RSA [RSA78]. These algorithms which offer strong protection are only suitable for key distribution since, due to their inherent complexity, bulk data encryption with these solutions seems prohibitive.

7.1 Motivation for The Proposed Framework

The inspiration for the approach we propose in this chapter was to apply to multicast confidentiality some of the principles that make multicast packet forwarding scalable, and in particular the use of intermediary components.

Intermediary components, be they network nodes, routers, or application proxies, are inherent participants in the basic multicast transmission process. The key scalability factor in the basic multicast transmission schemes is the spread of the multicast routing and packet forwarding load over a network of intermediary nodes. Placing security mechanisms on existing intermediary components seems to be a natural extension of existing multicast protocols. Moreover, partitioning the cost of security mechanisms over the intermediary components appears to be a good way of assuring processing scalability (Req. 5.6). When the multicast group grows, new intermediary components are added to support new group members and the cost of security mechanisms can still be equally distributed by placing the additional security processing load due to the new members on the new intermediary components.

The involvement of intermediary components in the security process is also a premise for meeting multicast security requirements. If the security mechanisms can be made dependent on the intermediary component in which they are implemented, group members attached to different intermediary components can be treated independently or with different keying material. In addition to its relationship with the group membership, the keying material can have a relationship with the topology of the multicast network. The keying material associated with each group member can thus be a function of the intermediary component to which the member is attached. This topological dependency assures the containment of security exposures: if some keying material belonging to a group member attached to an intermediary node is compromised, this keying material cannot be exploited by recipients attached to other intermediary nodes (Req. 5.4).

We introduce our solution in two steps: first, we define a general framework for multicast data confidentiality based on distributed mechanisms involving intermediary components and preserving the scalability and security properties, then we propose actual solutions based on cryptographic functions that comply with the framework.

This chapter is organized as follows. In section 7.2 we introduce a set of cryptographic sequences with special properties that are organized in a tree. In section 7.3, we apply this formal graph to a multicast tree. We show how the properties of our formal graph can be used to offer a multicast confidentiality framework that deals with the security and algorithmic requirements of Chapter 5. In section 7.4 we consider our framework for key distribution and present two key distribution schemes based on asymmetric cryptography in section 7.5 and 7.6. Finally, we conclude this chapter by a comparison with the other key distribution schemes we reviewed in the previous chapter.

7.2 Cryptographic Functions

The building blocks we have chosen to use to design data confidentiality protocols over a multicast tree are called ***Cipher Sequences*** (CS). This section will give a formal definition of these sequences and associate them in trees. These threes will be used in further sections to describe our multicast confidentiality framework.

7.2.1 Cipher Sequences

Definition 7.1 *Let $\mathcal{G} = \{g_i : \mathcal{N} \mapsto \mathcal{N},\ i \in \mathcal{A}\}$ denote a set of permutations operating on a message space $\mathcal{N}$ and indexed in $\mathcal{A}$. $\mathcal{G}$ will be called a* **Cipher Group** *if it verifies the following properties:*

(i) $(\mathcal{G}, \circ)$ forms a group trough the composition law $\circ$. Specifically, it is associative and contains the identity function Id; if $g', g'' \in \mathcal{G}$ then $g'' \circ g' \in \mathcal{G}$ (closure) moreover, $\forall\, g \in \mathcal{G}$, $\exists\, g^{-1}$ such that $g^{-1} \circ g = Id = g \circ g^{-1}$ (inverse).

(ii) $(\mathcal{G}, \circ)$ is indexable through composition: there exists a polynomial time algorithm $Comp : \mathcal{A}^2 \mapsto \mathcal{A}$, which given an index pair $(i, j) \in \mathcal{A}^2$, computes $k = Comp(i, j) \in \mathcal{A}$ such that $g_{(k)} = g_{(j)} \circ g_{(i)}$

Moreover:

- $\mathcal{G}$ will be called a ***Symmetric Cipher Group*** if the knowledge of $g \in \mathcal{G}$ allows the computation of g^{-1} in polynomial time.
- $\mathcal{G}$ will be called an ***Asymmetric Cipher Group*** if the computation of g^{-1} from $g \in \mathcal{G}$ is computationally infeasible without the knowledge of a trapdoor.

Definition 7.2 *Let $\mathcal{F}$ be a random sequence $(f_{0<i\leq n})$ of n elements in a cipher group $\mathcal{G}$. By definition, there exists a function $g \in \mathcal{G}$ such that $g = (f_n \circ f_{n-1} \circ \cdots \circ f_1)^{-1}$. We will call this function the* **Reversing Function** *of $\mathcal{F}$ and denote it as $h_{\mathcal{F}}$ or, more simply h. The sequence $\mathcal{F}$ will be called a* **Cipher Sequence** *in $\mathcal{G}$, or* $\mathbf{CS}_{\mathcal{G}}$.

Consequently, if $\mathcal{G}$ is a symmetric cipher group, we will say that $\mathcal{F}$ is a ***Symmetric Cipher Sequence*** in $\mathcal{G}$ or $\boldsymbol{SCS}_{\mathcal{G}}$. Similarly, if $\mathcal{G}$ is a asymmetric cipher group, we will say that $\mathcal{F}$ is an ***Asymmetric Cipher Sequence*** in $\mathcal{G}$ or $\boldsymbol{ACS}_{\mathcal{G}}$.

Definition 7.3 *Let $\mathcal{F}$ define a Cipher sequence of n functions in the cipher group $\mathcal{G}$. From this sequence $\mathcal{F}$ we can derive a sequence $(S_{0\leq i\leq n})$ of elements in $\mathcal{N}$ as follows:*
$S_i = f_i(S_{i-1})$, for $n \geq i > 0$.
S_0, the initial value of the sequence.

We will call $(S_{0\leq i\leq n})$ an ***instance*** of $\mathcal{F}$ and we will denote it as $\mathcal{F}(S_0)$.

7.2.2 CS$_{\mathcal{G}}$'s over a General Tree

A tree can map a family of $CS_{\mathcal{G}}$'s which have terms that differ only after a certain rank, greater than 1. For example, if $\mathcal{F}_1$ (resp. $\mathcal{F}_2$) is a $CS_{\mathcal{G}}$ defined as $\mathcal{F}_1 = \{f_1, f_2, f_3, f_4\}$ (resp. $\mathcal{F}_2 = \{f_1, f_2, f_3', f_4'\}$), a simple tree that maps $\mathcal{F}_1$ and $\mathcal{F}_2$ can be constructed as in figure 7.1. This tree illustrates the fact that $\mathcal{F}_1$ and $\mathcal{F}_2$ differ after rank 2.

This property can be extended from 2 to n different $CS_{\mathcal{G}}$'s, $\mathcal{F}_{0<i\leq n}$, with a n-ary tree that branches each time at least two sequences differ. If we instantiate each $\mathcal{F}_k$ in the tree with the same value S_0 we can label the edges of the tree with the elements S_k^i of each $\mathcal{F}_i(S_0) = \left(S^i_{0<k\leq n_i}\right)$. We give a small example on figure 7.2. Note that all the instantiations $\mathcal{F}_i(S_0)$ share at least two elements, namely S_0 and S_1. These two values are the input and the output, respectively, of the root node of the instantiated tree.

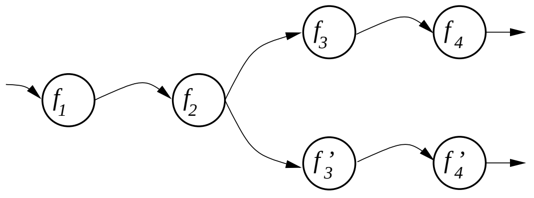

Figure 7.1: Two $CS_{\mathcal{G}}$'s mapped over a tree.

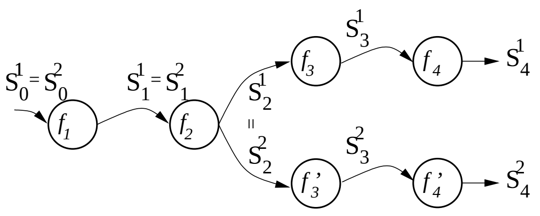

Figure 7.2: Two instantiated $CS_{\mathcal{G}}$'s in a tree.

Remark 7.4 *We implicitly require that for all* $0 < i \neq j \leq n$, $\mathcal{F}_i$ *is not included in* $\mathcal{F}_j$. *In our trees, this translates to the fact that the number of leafs is equal to the number of cascade sequences mapped over the tree.*

7.3 Multicast Confidentiality Framework

The proposed multicast confidentiality framework consists of a model that is an abstract definition of the components involved in the security mechanisms and the relationship between them which is an application of the functions we defined in section 7.2.

7.3.1 Model

In the abstract definition of the framework, the components of the multicast security framework form a ***tree***, as suggested in 1.1.2. The ***root*** of the tree is the multicast source and the members of the multicast group form the ***leaves*** of the tree. The intermediary nodes of the tree -referred to as ***inner nodes***- correspond to the intermediary components of the multicast communication network. Like the multicast scheme itself, inner nodes can be implemented at the application layer or at the network layer. In the case of application layer multicast, inner nodes can be application proxies, such as those in a hierarchical web caching structure. In the case of network layer multicast, inner nodes can be intelligent routers capable of performing security operations in addition to multicast packet forwarding functions.

In further abstraction, each leaf of the tree will represent the set of group members attached to the same terminal inner node. In the application layer case, a leaf will delimit a sub-group of members attached to a proxy. In the network layer case, a leaf will delimit a sub-network of recipient stations attached to an IGMP router. Hence a leaf will refer to a set of multicast group members with a common attachment node in the tree.

If a set of users represented by a leaf becomes too large, the leaf can easily be subdivided into several "sub-leaves" by adding new inner nodes. Hence the leaf size in terms of the group members it represents is not a scalability issue for algorithms that treat a leaf as a single entity.

7.3.2 CS$_{\mathcal{G}}$'s over a Multicast Tree

Next, we turn to multicast by applying the previous concept of $CS_{\mathcal{G}}$ from section 7.2 over a tree as a means of performing secret transforms in multicast. Let S_0 be the information to be transmitted over the multicast channel by the source under confidentiality. Furthermore, S_0 might either be the actual data or an encoding thereof, if possible data values are different from possible values that S_0 can take on from the point of view of the security algorithm.

As part of the setup for a series of secure multicast transmissions, each inner node N_i is assigned a secret function $f_{i>0} \in \mathcal{G}$. We require that each inner node is capable of performing this function $f_{i>0}$ as defined in the previous section. During secure multicast transmission, upon receipt of multicast data S_j from its parent node N_j, node N_i computes $f_i(S_j)$, and forwards the resulting value S_i as the secure multicast data to the child inner nodes or the leaves.

Assuming $\mathcal{F}_i(S_0)$ is an instance of a $CS_{\mathcal{G}}$ mapped over a path from the root to a leaf on the multicast tree, the leaf will eventually receive S^i_n, which is the final term of $\mathcal{F}_i(S_0)$. The leaves in the multicast tree bear a special role in that they are able to recover the original message S_0. Each leaf is assigned a reversing function h^i that allows it to compute $S^i_0 = S_0$ from $S^i_{n_i}$ since $S_0 = h^i(S^i_{n_i})$. The leaves don't use any other function in $\mathcal{G}$.

The distribution of the secret f_i functions to the inner nodes and the reversing functions to the leafs can be assured by a central membership manager using classical unicast security mechanisms. Because of the structure of the algorithm, the central server will need to have a precise image of the tree structure. This doesn't mean, however, that membership management cannot be distributed over several network entities. In fact, the tree structure allows membership management to be distributed over an overlapping tree of membership managers, each one managing a large subtree.

Working example

Figure 7.3 depicts a simple tree with three $CS_{\mathcal{F}}$'s. Looking at the path from the root to leaf 3 on figure 7.3, we have:

- The root computes $f_1(S_0)$ and sends the result to its children inner nodes.
- N_1 receives $S^3_1 = f_1(S_0)$, computes and sends $f_7(S^3_1)$ to N_2.
- N_2 receives $S^3_2 = f_7(S^3_1)$ and sends $f_8(S^3_2)$ to leaf 3.
- Leaf 3 receives $S^3_3 = f_8(S^3_2)$ and recovers the original multicast data by computing $S_0 = h^3\left(S^3_3\right)$.

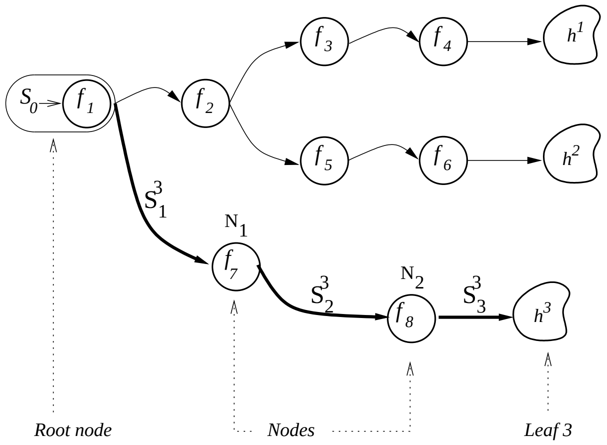

Figure 7.3: A simple 3 $CS_{\mathcal{G}}$ tree.

Adding A Member

When a member is added to a group by contacting a membership manager, two situations can arise:

1. the new member is already in an existing leaf (sub-group) of the tree.
2. the new member is not in an existing leaf (sub-group) of the tree.

In the first situation, a cipher sequence $\mathcal{F} = (f_{0\leq i\leq n})$ is already mapped between the source and the members in the existing leaf. The last inner node on the path which holds function f_n will be assigned a new value $\widetilde{f_{\underset{\sim}{n}}}$, updating the last transformation in the sequence. Hence, the corresponding new $\widetilde{h}$ function will be distributed to all the member in the leaf including the new member.

In the example of figure 7.4 where C wishes to join the leaf including existing members A and B, the addition of C will result in the following events:

1. $\widetilde{f_8}$ will be substituted to f_8 by the membership manager in the last inner node.
2. h^3 will be sent to A, B, C by a membership manager.

If M is the upper bound on the number of members in a leaf, adding a member requires the exchange of the following messages:

- 1 message sent to update the value in the last inner node on the path,
- at most $M - 1$ messages sent to the current members in the leaf,
- 1 message sent to the new member.

A member addition thus requires at most $M + 1$ messages.

In fact, it's possible to reduce the number of messages to 3, by slightly changing the order of operations in the member addition procedure. Instead of changing the

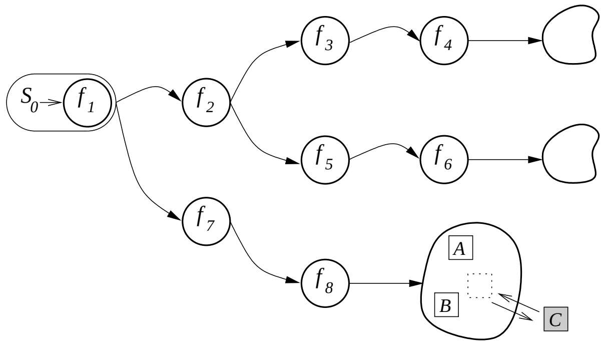

Figure 7.4: User C joins/leaves.

value f_n in the node right away, it's possible to use the secure sequence to vehicle the new $\widetilde{h}$ function to the current members in the leaf, thus reducing the update to one message (versus an upper bound of $M-1$ messages). Then the value f_n in the node can be changed and the new $\widetilde{h}$ function can be transmitted to the added member. However this approach has a drawback: it creates a chain between the different values of $\widetilde{h}$ which potentially weakens the security of the scheme. Unless the cost of individually sending a message to each member in the leaf is more important than the security of the group, such an option should be avoided.

In the second case, the authority which receives a member addition request has to figure out the path from the new member to the closest inner node in the active tree. The path establishment method used in this case depends on the layer (application/network) at which the multicast security scheme is implemented. A similar decision has to be taken by IP multicast routing algorithms when a new router needs to be included in a multicast routing tree. Once the path to the new member is selected, the membership management authority will assign values to the newly added inner nodes on the path, thus extending the $CS_{\mathcal{G}}$ mapping. Finally the new member will receive the h function needed to recover the original multicast data in the newly created $CS_{\mathcal{G}}$. Hence, it's the only member of the new leaf in the tree.

The number of messages exchanged here depends on the algorithm used to set the path between the new member and the tree. Consequently, as stated in section 7.1, this security framework would be a natural extension of multicast routing schemes. The number of messages exchanged here to create a new leaf can be assumed to be proportional to the number of messages exchanged by the multicast routing protocols when adding a new element in the multicast tree.

In many cases, it will be possible to perform the node setup ahead of time, leaving only the h function to be distributed when the member is effectively added to the group. The authority that manages the group does not need to be the root itself and its functionality can be distributed in a tree hierarchy, where each sub-authority manages a multicast subtree.

Removing A Member

Removing a member shares a lot of similarity with the addition of a member. When a member is removed from a leaf in the tree, the function in the terminal inner node is changed from f_n to $\widetilde{f_n}$ and the new $\widetilde{h}$ reversing function is distributed to the remaining members in the leaf. In effect, the associated $CS_{\mathcal{G}}$ has its last term changed.

7.3.3 Evaluation of the Framework

The previous discussion has focused on the use $CS_{\mathcal{G}}$ to achieve data confidentiality over a multicast tree. This section will show how the $CS_{\mathcal{G}}$ construct meets the requirements established in Chapter 5, assuming *for now* that the intermediary nodes are trusted and secure. The implications of node compromise will be discussed in the next section.

Data Confidentiality (Req. 5.1). The security of the scheme depends on the strength of the ciphers that are used to implement it. The functions in a cipher group should be viewed as building blocks for confidentiality services. A possible approach, which is beyond the scope of this work, would be to consider these function as pseudo-random permutations. (As a side effect, it would require the cipher groups to be of large order: a small order would provide a method to build a polynomial time distinguisher between cipher group elements and pseudo-random permutations).

Backward and Forward Secrecy (Req. 5.2). A new member is added to the group a leaf gets a new reversing function $\widetilde{h}$ that cannot be used to recover the old reversing function h . As a consequence, past data is not accessible to a new member. Similarly, a former member using an old reversing function cannot access data that is transmitted subsequently to its removal.

Containment (Req. 5.4) Because of the topological dependency introduced by the model, the reversing function h used in a leaf of the tree will be useless outside that leaf. An intruder will only benefit from an attack if he is located in the same leaf as the victim. This greatly reduces the impact of member compromise.

Collusion Resistance (Req. 5.3) Let $(G, .)$ define a group. For a random $(a_0, a_1, ..., a_n) \in G^{n+1}$, define $b_i = a_0.a_i$ for all $1 \leq i \leq n$. An adversary observing $(b_1, b_2, ..., b_n)$ does not gain any knowledge about a_0.

Proof. For all $a_0 \in G$ we can write each $b_i \in G$ as $b_i = a_0.a_i$ where $a_i = (a_0^{-1}.b_i)$. Thus, a_0 can be any of $|G|$ possible values. This means that in a fully balanced tree of depth 1, members who collude and exchange their reversing functions gain no knowledge of the intermediary functions. This can be easily generalized to any tree, provided that no node is both an inner node and a leaf, as already highlighted in remark 1 of section 7.2.

Processing Scalability (Req. 5.6) The amount of processing per component is independent of the group size. First, in our framework, the size of messages transmitted by a node (be it the source or an intermediary component) does not depend on the number of group members but it depends only on the size of the original secret message. Second, the number of messages transmitted by a node does not depend on the number of group members but it depends only on the number of child nodes attached to this node.

In a leaf the number of messages exchanged after a member is added or removed from the group is limited by the size of the leaf and not the whole group. Since the leafs have a maximum size, this is not a scalability issue.

Membership Robustness (Red. 5.7) There are three basic actions a group member is involved in: addition, removal and accessing the content. The model is designed so that none of these actions affects the whole group. In fact these actions have an impact that is limited to the leaf containing the member performing these actions as shown in section 7.3.2. Membership Robustness issues are fully localized and never appear over the whole group.

7.3.4 Node Compromise.

The previous section assumed that the inner nodes of the tree were completely secure. This has to be true for the root node of the tree but it might not be possible to make such an assumption about the intermediary nodes in the network. Hence the following section will focus on the impact of intermediary node compromise. Two type of attacks that derive from node compromise are highlighted in this section: unauthorized membership extension and mode compromise by external users. Unauthorized membership extension happens when a former member of the secure group is able to maintain access to the data even though he has not received the new reversing function. Node compromise by external users more generally describes unauthorized access to the group by users that never became group members.

Unauthorized Membership Extension

If a member *Eve* in a leaf controls the last inner node on the path from the source to the leaf i, he can intercept changes in the last element of the $CS_{\mathcal{G}}$.

Let N be the last inner node on the path from the root to leaf i of the tree and f the secret function held by N. N receives S^i_{j-1} from its parent node and sends $S^i_j = f(S^i_{j-1})$ to the leaf elements which will use a h^i reversing function to recover S_0. If the group membership manager decides that *Eve* must leave the group, the function f in N will be changed to a new function $\widetilde{f}$ and the corresponding reversing function $\widetilde{h}^i$ will be send to all leaf members except *Eve*.

Despite its formal exclusion from the group, *Eve* can ignore the change in the router and compute S_0 from S^i_{j-1} using f and the old reversing function h^i obtained trough the compromise of node N, simulating the older sequence, where $S_0 = h^i\left(f(S^i_{j-1})\right)$.

This attack works whatever the nature of the sequence, ACS_f or SCS_f, but requires several conditions to be met:

1. *Eve* should be a former member of the group.
2. *Eve* should be able to access the secret function f held by node N.
3. *Eve* should have access to the data transmitted to node N by its parent node (i.e. S_{j-1}).

Moreover, updates of f_i functions in inner nodes at a higher level will limit the scope of this attack because the resulting reversing functions cannot be retrieved based on the information gathered in a leaf or from the compromise of the last inner node.

A $SCS_{\mathcal{G}}$ specific attack Condition 3 described in the previous paragraph is not required if the sequence is an instance of a $SCS_{\mathcal{G}}$. To work around condition 3, the intruder first computes:

$$\widetilde{f}^{-1} \text{ from } \widetilde{f}$$

because the sequence is symmetric.
Now, using $\widetilde{f}^{-1}$ and the new sequence value $\widetilde{S}^i_j$ received in the leaf, the former member computes:

$$S^i_{j-1} = \widetilde{f}^{-1}(\widetilde{S}^i_j)$$

Next, the intruder uses the value f obtained through the compromise of N to compute:

$$S^i_j = f(S^i_{j-1})$$

Finally, using the old reversing function h^i and applying it to S^i_j, we have:

$$S_0 = h^i(S^i_j)$$

where S_0 is the original multicast data.

This attack doesn't apply in an ACS_f tree because by definition $\widetilde{h}^i$ cannot be deduced form $\widetilde{f}$.

Node Compromise by External Users

If the intruder *Eve* is not even a former member of the group, an attack is still possible if the sequence is symmetric provided that:

1. *Eve* has access to the value of the reversing function used by a legitimate member .
2. *Eve* controls all the inner nodes on the path between him[1] and the legitimate member except the first common ancestor they have in the tree.

If these conditions are met, *Eve* will be able to forge a reversing function he can use to access the group.

Instead of a lengthy formal discussion, we choose to illustrate the attack with the example scenario depicted on figure 7.7, where the malicious user *Eve* gets multicast

[1] *Eve* does not have to be in a real leaf, he can simply intercept traffic somewhere in the tree.

data S_3^2 from node N_5. If Eve knows h^3 from a compromised user and $\{f_2, f_5, f_7, f_8\}$, he can compute:

$$S_2^2 = f_5^{-1}(S_3^2)$$

because f_5^{-1} can be derived from f_5. Similarly,

$$S_1^3 = S_1^2 = f_2^{-1}(S_2^2)$$

because f_2^{-1} can be computed from f_2. Then

$$S_2^3 = f_{a_7}(S_1^3)$$

and

$$S_3^3 = f_{a_8}(S_1^3)$$

yielding to

$$S_0 = h^3(S_3^3)$$

Again, this attack doesn't apply to an $ACS_{\mathcal{G}}$ based tree because reversing functions associated with an $ACS_{\mathcal{G}}$ cannot be derived from the parameters used in the intermediary nodes.

Node Compromise Summary

The distinction between an $ACS_{\mathcal{G}}$ and a $SCS_{\mathcal{G}}$ is totally relevant with respect to node compromise scenarios. Unlike the IOLUS scheme we reviewed in the previous chapter, when using $ACS_{\mathcal{G}}$'s our framework is immune to node compromise by external users. The framework does not however dictate the choice of an $ACS_{\mathcal{G}}$ over $SCS_{\mathcal{G}}$ as one could expect because $SCS_{\mathcal{G}}$ are likely to be easier to design than $ACS_{\mathcal{G}}$.

It should be noted that the security containment property is also effective in case of node compromise. Hence, previously described node compromise scenarios don't allow the intruder to provide unauthorized access to just any other user in the network.

This section concludes the formal presentation of our secure multicast framework. The next sections present two implementations of this framework based on extensions of public key cryptographic schemes. The first scheme is an $SCS_{\mathcal{G}}$ and will therefore lend itself to further description of a concrete node compromise scenario.

7.4 Key Distribution

Depending on the performance of functions in $\mathcal{G}$ our framework can be used either for bulk data confidentiality or only for key distribution. Current symmetric cryptographic systems provide sufficient encryption speed but they don't exhibit the mathematical properties required to create a $CS_{\mathcal{G}}$ as demonstrated in [KRS85][CW93].

On the other hand asymmetric cryptography offers suitable properties to build a solution compliant with the framework but it doesn't offer yet the necessary performance for bulk data confidentiality. Consequently, the next two sections will describe the framework based on asymmetric cryptography for multicast key distribution. The first scheme, derived from the ElGamal encryption algorithm, allows the creation of a $SCS_{\mathcal{G}}$ key distribution tree, whereas the second scheme, based on RSA, effectively creates an $ACS_{\mathcal{G}}$ key distribution tree.

Using a $CS_{\mathcal{G}}$ tree, the source can distribute a secret key k by instantiating the tree with $S_0 = k$ (or otherwise a function of k). The data confidentiality mechanism of the secure multicast framework will allow to securely transmit k to the members of the group. Unlike the reversing function that is different in each leaf, k is shared among all members of the group so the exposure of k affects the group as a whole. However, unlike the reversing function that enables each member to access the multicast group, the shared key k is a short term value that can be frequently updated by the source using the secure multicast framework. Consecutive values of k are independent.

7.5 Key Distribution using ElGamal.

The ElGamal cryptosystem can be extended to create a $SCS_{\mathcal{G}}$.

Proposition 7.5 *Let p be a large random prime. Define $\mathcal{N}$ as the set of all primitive elements of $\mathbb{Z}_p^*$. The set $\mathcal{G} = \left\{ f_a(x) = x^a \, mod \, p; a \in \mathbb{Z}_{(p-1)}^* \right\}$ forms a symmetric cipher group for the message space $\mathcal{N}$.*

Properties (i) and (ii) in definition 1 are easily verified. Given an index $i \in \mathbb{Z}_{(p-1)}^*$, we can compute $f_{(i)}(x) = x^i \, mod \, p$ as well as $f_{(i)}^{-1}(x) = x^{1/i} \, mod \, p$. Given $(i,j) \in \mathbb{Z}_{(p-1)}^*$, we can compute $k \in \mathbb{Z}_{(p-1)}^*$ such that $f_{(k)}(x) = f_{(i)} \circ f_{(j)}(x) = f_{(j)} \circ f_{(i)}(x) = x^{i.j} \, mod \, p = x^k \, mod \, p$.

By definition, all the elements in $\mathcal{N}$ are of order $p-1$. Reasonable assumptions about the security of these functions are:

1. For random primitive elements $x_i \in \mathcal{N}$ and a random $f \in \mathcal{G}$, an adversary has a negligible chance to recover any x_i without knowing f.

2. For random primitive elements $x_i \in \mathcal{N}$ and a random $f \in \mathcal{G}$, an adversary has a negligible chance to recover f by observing pairs of the form $(x_i; f(x_i))$.

7.5.1 Setup

The source chooses a generator g of the cyclic group $\mathbb{Z}_p^*$ and a secret random value r in $\mathbb{Z}_{(p-1)}^*$. The inner nodes and the root are assigned f_{a_i} values in $\mathcal{G}$ to form a $SCS_{\mathcal{G}}$ tree. The tree is instantiated with $S_0 = g^r \, mod \, p$.

Let $\{S_{i_k > 0}\}$ denote the instantiated sequence elements. The reversing function distributed to the leafs is defined as:

$$h^k(x) = x^{(a_{i_1} . a_{i_2} . a_{i_3} \ldots a_{i n_k})^{-1}} \, mod \, p$$

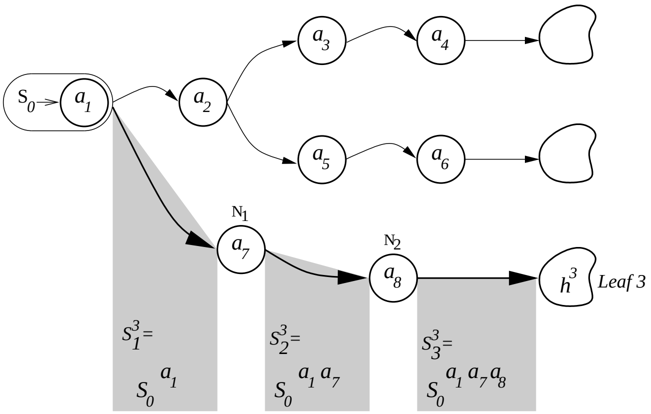

Figure 7.5: A discrete log tree.

7.5.2 Key Distribution

The source wishing to distribute a key K sends the following initial data to its children in the tree:

$$S_1 = (S_0)^{a_1} \, mod \, p$$

$$T = K \oplus S_0$$

Remark 7.6 *Issues such as a proper padding of K, have been omitted here for simplicity. Depending on the exact security properties we need, we can substitute T with a better encoding of K as suggested, for example in [ABR98].*
Alternatively, we can simply use a cryptographic hash function and set $K = Hash(S_0)$, since S_0 is changed for each new K.

The intermediary elements in the tree perform $f_{a_{i_k}}$ on their input $S_{i_k - 1}$ and send S_{i_k} to their children, along with T, where:

$$S_{i_k} = f_{a_{i_k}}(S_{(i_k-1)}) = \left(S_{(i_k-1)}\right)^{a_{i_k}} \, mod \, p$$

An example of this scheme is illustrated on the path from the root to the leaf 3 of the tree on figure 7.5:

- The source sends $S_1^3 = (S_0)^{a_1} \, mod \, p$ and $T = K \oplus S_0$ to its children.
- N_1 receives (S_1^3, T) and sends $S_2^3 = (S_0)^{a_1 a_7} \, mod \, p$ and $T = K \oplus S_0$ to its children.
- N_2 receives (S_2^3, T) and sends $S_3^3 = (S_0)^{a_1 a_7 a_8} \, mod \, p$ and $T = K \oplus S_0$ to leaf 3.

Decryption

The decryption process is straightforward, the reverse function h is simply applied to the received value, and the result is used to extract K from T:

$$h(S_{i_k}) = S_0 \, mod \, p$$

$$K = T \oplus S_0$$

Recalling the previous example, where $h^3(x) = x^{\frac{1}{a_1 a_7 a_8}} \, mod \, p$, the value of K is computed simply:

$$K = T \oplus S_0$$

where

$$S_0 = h^3\left(S_3^3\right) = \left((S_0)^{a_1 a_7 a_8}\right)^{\frac{1}{a_1 a_7 a_8}} \; mod \, p$$

The Next Key

Sending the next key $\widetilde{K}$ only requires $S_0 = g^r \, mod \, p$ to be updated as $\widetilde{S}_0 = g^{\widetilde{r}} \, mod \, p$ where $\widetilde{r}$ is a random element in $\mathbb{Z}^*_{(p-1)}$.

7.5.3 Node Compromise and Member Collusion

Many of the requirements established in Chapter 5 are naturally fulfilled by implementing the framework as described above. However member collusion and node compromise need to be considered on a per-algorithm basis.

Node Compromise

The previously described sequence is clearly a $SCS_{\mathcal{G}}$ because the reversing functions can be computed with the knowledge of the secret parameters in the inner nodes. Hence, compromise of the inner nodes offers some potential for unauthorized membership extension as described in 7.3.4.

Figure 7.6 will illustrate the node compromise scenario. The hypothesis here will be that a malicious member E of leaf 3 wishes to maintain membership in the group using the information of the terminal inner node N_2 he has compromised.

In a normal scenario where node compromise is not taken into account, in a leaf consisting of members $\{A, B, C, E\}$, when E is removed from the group, the following actions take place:

- In N_2, f_{a_8} is changed to $\widetilde{f}_{a_8}$ by the membership manager.
- The newly computed reverse function $\widetilde{h}^3$ is sent to $\{A, B, C\}$ but not to E.

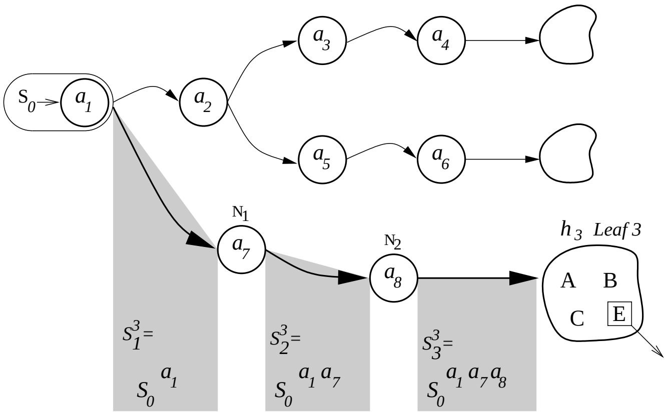

Figure 7.6: Node Compromise.

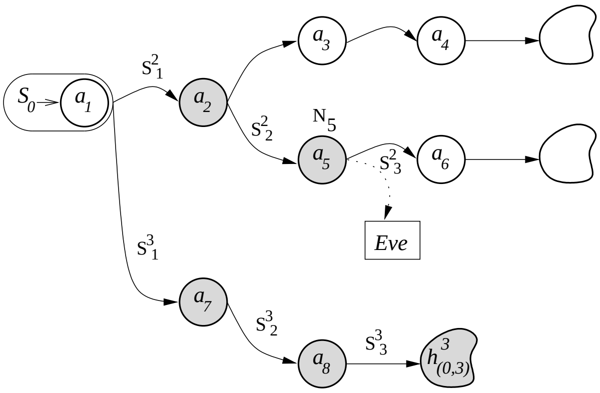

Figure 7.7: Multiple node compromise attack.

Once the leave procedure is complete, E cannot access further keys distributed to $\{A, B, C\}$ it cannot derive $\widetilde{h}^3$ from h^3.

However, in the case of inner node compromise, if E controls the last inner node, he can monitor the change from f_{a_8} to $\widetilde{f}_{a_8}$. E can then derive $\widetilde{h}^3$ from h^3, because if $h^3(x) = x^{(a_1a_7a_8)^{-1}}\ mod\, p$ then:

$$\widetilde{h}^3(x) = x^{\frac{1}{a_1a_7a_8}\times\frac{a_8}{\widetilde{a}_8}}\ mod\, p$$

In summary, even if E doesn't receive the new reversing function, he will be able to compute it and thus access the keys distributed subsequently to the removal of a member.

This attack can be extended to allow a malicious user to derive a reversing function from another one even if the reverse function comes from another leaf. It requires the attacker to compromise nearly all the inner nodes on the graph between him and the compromised member.

Figure 7.7 will serve as an example where user *Eve* -not a member of the group- listens to traffic coming out of N_5. The malicious user is assumed to know the following node functions $\{f_{a_2}, f_{a_5}, f_{a_7}, f_{a_8}\}$ as well as h^3 from a compromised member in leaf 3. With these conditions together, *Eve* can compute a new local reverse function h^2 from h^3 thus violating the containment property of the model:

$$h_3^3(x) = x^{\frac{1}{a_1 a_7 a_8}} \; mod\, p$$

which allows to compute:

$$h^2(x) = x^{\frac{1}{a_1 a_2 a_5}} \; mod\, p = \left(h_3^3(x)\right)^{\frac{a_7 a_8}{a_2 a_5}} \; mod\, p$$

This second attack assumes that the inner nodes are easy to compromise, and the first one makes strong assumptions about the compromise power of the attacker. While these attacks on $SCS_{\mathcal{G}}$'s might be considered hard to implement in some cases, the possibility itself pushed us to study a second and stronger construction based on $ACS_{\mathcal{G}}$'s, as described in section 7.6.

7.6 Key Distribution using RSA.

Extending RSA to use multiple keys as in [HK89] allows the creation of an $ACS_{\mathcal{G}}$ scheme. Let $\varphi(.)$ denote the Euler totient function.

Proposition 7.7 *Let $n = pq$ be the product of two carefully chosen large primes, as in the RSA cryptosystem. Define $\mathcal{A} = \mathbb{Z}^*_{\varphi(n)}$ and $\mathcal{N} = \mathbb{Z}_n$. The set $\mathcal{G} = \{f_a(x) = x^a \, mod\, n; a \in \mathcal{A}\}$ is a cipher group for messages in $\mathcal{N}$.*

This construction is quite similar to the one from the previous section, except that we are working with a composite modulus and a different message space.

7.6.1 Setup

The setup is even simpler here than in the ElGamal case. Each inner node in the tree is assigned a value $a_{i>1}$ and the root uses a_1 where $\gcd(a_{i\geq 0}, \varphi(n)) = 1$. This assures that the product A of any subset of these a_i values also verifies $\gcd(A, \varphi(n)) = 1$. The multiplicative inverse B of A defined as $AB \equiv 1 \; (mod\, \varphi(n))$ can be computed using the Euclidean algorithm.

Let $\{a_{k_i>0}\}$ denote the set of parameters used in the inner nodes between the source and leaf k, plus $a_{k_1} = a_1$ in the root. The reversing function distributed to the leafs is defined as:

$$h^k(x) = x^{D_k} \; mod\, n$$

where,

$$(a_{k_1}.a_{k_2}...a_{k_m}).D_k \equiv 1 \; (mod\, \varphi(n))$$

Like the basic RSA algorithm, the asymmetric property of this scheme relies on the difficulty of computing D_k from the reversing function without the knowledge of $\varphi(n)$ (which currently seems to be only derivable from the factors of $n = pq$).

7.6.2 Key Distribution

The source wishing to distribute a key K, sends the following value to its children in the tree:

$$S_1 = K^{a_1} = (S_0)^{a_1} \, (mod\, n)$$

Remark 7.8 *Here we have $S_0 = K$, and for simplicity we have omitted issues such as padding or semantic security. A proper (and probabilistic) encoding of K is suggested in [BR95].*

Each inner node N_i in the secure multicast tree processes the S_{i-1} value received from its parent node and sends S_i to its children inner nodes where:

$$S_i = f_{a_i}(S_{i-1}) = (S_{i-1})^{a_i} \, (mod\, n)$$

Recalling figure 7.6 while assuming an RSA like $ACS_{\mathcal{G}}$ sets the following scenario on the path from the root to leaf 3 of the tree:

- The root send $S_1^3 = (S_0)^{a_1} \; mod\, n$ to its children.
- N_1 receives S_1^3 and sends $S_2^3 = (S_0)^{a_1 a_7} \; mod\, n$ to its children.
- N_2 receives S_2^3 and sends $S_3^3 = (S_0)^{a_1 a_7 a_8} \; mod\, n$ to leaf 3.

Decryption

The decryption process is also simpler than in the ElGamal case. The decryption function h is applied to the received value in the leaf to recover K. For example, on figure 7.7:

$$K = S_0 = h^3(S_3^3) = ((S_0)^{a_1 a_7 a_8})^{D_3} \; mod\, n$$

assuming

$$a_1 a_7 a_8.D_3 \equiv 1 \; (mod\, \varphi(n))$$

The Next Key.

Sending a new key $\widetilde{K}$ only requires S_0 to be changed in the preceding description. Nothing else needs to be done.

7.6.3 Node Compromise

The node compromise attack previously described in section 7.5 regarding the discrete log case does not apply here essentially because the RSA based sequences are asymmetric: to compute the inverse of any function $\{f_{a_1}, f_{a_2}..., f_{a_k}\}$, the knowledge of the intermediary parameters $\{a_1, a_2..., a_k\}$ wouldn't be sufficient as $\varphi(n)$ is also required. However the node compromise attack based on membership extension as depicted in section 7.3.4 is still possible with the RSA based scheme.

7.7 Related Work

In this section we compare our framework with the other proposals from chapter 6. Moreover we will look at the particular implication of using our scheme for key distribution.

Processing Scalability and Membership Robustness.

We believe that MARKS is the most scalable scheme, since the remove operation introduces no side effects. Containment oriented schemes such as IOLUS and ours, come next since the side effects are limited to a small subgroup. LKH require a global update each time a member leaves the group and current research about LKH aims at reducing the cost of this operation through various techniques.

Trust

Although our solution uses intermediary components, it has a major difference with IOLUS: our framework puts limited trust in the intermediary components, whereas in IOLUS each intermediary component has access to the multicast data. This problem does not appear in LKH or MARKS since no intermediary elements are involved.

Containment

In terms of containment, our scheme is equivalent to IOLUS, where each subgroup uses a different key to access the multicast data. On the other hand LKH and MARKS do not address containment issues even though they use a computational tree structure. In those schemes, the keys held by any user can be used to access the multicast group anywhere and all users are equivalently trusted with the security parameters of the group.

Key distribution

Even though we offer higher security in terms of trust and containment, this has a cost. Indeed, IOLUS, LKH and MARKS have a clear advantage over our scheme in terms of performance. This has lead us to consider our scheme for key distribution and not bulk data encryption. In that respect the framework is used to distribute a short term data encryption key k. As this short term key is common to all recipients, it may look as our scheme looses its containment advantage over LKH or MARKS. However, the short term key can be frequently updated and its disclosure does not provide a means of long term group access to intruders. This is because in our scheme the group membership is represented by the long term reversing functions that are different in each leaf of the multicast tree as opposed to the shared secret group membership key(s) of LKH and MARKS.

7.8 Conclusion

This chapter presented framework designed to support data confidentiality in a large dynamic multicast group. The framework meets a set of requirements wider than previous proposals. In particular this work was the first one to introduce the concept of containment in [MP99] and the use of untrusted intermediary elements in the context of multicast confidentiality.

The introduction of Cipher Sequences, or $CS_{\mathcal{G}}$, provides a formal but yet practical description of the framework elements, with a voluntary distinction between symmetric and asymmetric behaviors. The mapping of these sequences over a multicast tree is the core mechanism that allows this framework to meet the requirements of Chapter 5.

Two key distribution schemes have been presented as implementations of the framework. They have served as a proof of concept for the framework and they have allowed us to discuss the implication of various node compromise scenarios, as the possibility of node compromise cannot be neglected in a large multicast network.

The next major step would be the design of efficient functions that could be used to build $CS_{\mathcal{G}}$'s that operate on bulk data, in order to fully capitalize on this framework. However, it seems difficult to find symmetric cryptographic transformation which exhibit the mathematical structures needed to construct $CS_{\mathcal{G}}$'s. In the next chapter, we propose a solution that operates on bulk data, which approaches the Cipher Sequences with certain limitations using efficient cryptographic techniques.

Chapter 8

Untrusted Key Encryptions Trees: Layered Encryption.

8.1 Introduction

The previous chapter proposed a construction based on sets of functions which exhibited a group structure through composition. The implementations of the framework we proposed relied on asymmetric cryptographic techniques that are typically much more expensive than symmetric ones and restrict the field of application of our framework to key distribution rather than bulk data encryption. It would be nice however to find a block cipher algorithm $\mathcal{E}$ that would verify $\mathcal{E}_{k_1}(\mathcal{E}_{k_2}(M)) = \mathcal{E}_{k_3}(M)$ for a triplet of keys $\{k_1, k_2, k_3\}$ and any message M. It turns out that such a property would probably be considered a weakness by cipher designers. Not only would it make multiple encryption useless, but also it would allow a meet-in-the-middle known plaintext attack that would achieve a high probability of success with a cost of $O(2^{\frac{k}{2}})$ operations. Let $C = \mathcal{E}_k(M)$, the attack would proceed by encrypting M and decrypting C with two varying keys, and then halting on a collision between any encryption of M and any decryption of C, which is a typical birthday paradox attack in terms of complexity.

In this chapter we propose to explore another way[PM02b] to combine several encryption operations, which is in between classical multiple encryptions and the Cipher Sequence construction from the previous chapter. We propose a re-encryption tree framework that takes advantage of the fact that the so called "counter based" cipher mode of operation uses the commutative XOR operation to combine a pseudorandom pad with the plaintext to produce the ciphertext. We use a multiple key version of this encryption scheme and distribute the keys in a way that limits the trust we put in the intermediary elements in the network.

This chapter is organized as follows. In the first section, we will look at some interesting cryptographic primitives that we use in our framework. Then, we present our framework based on multiple layers of encryption, or l-layer trees. In the following section we analyze the security of our construction, and discuss its scalability and relate it to the list of requirements we presented in chapter 5. Finally, we present a potential improvement of our scheme with a shorter message expansion.

8.2 Cryptographic primitives.

Our framework uses a multiple key version of the counter based block cipher mode of operation (XORC) as described in [BDJR97] and can alternatively be constructed with any general stream cipher such as SEAL[RC98]. Counter mode has been proposed for standardization to NIST as an official AES mode of operation in [LRW00]. In this section we will briefly recall counter mode (XORC) and present our own multiple key extension that is used in the framework. We denote "$\oplus$" as the binary XOR operation.

8.2.1 XORC encryption scheme.

In [BDJR97] Bellare et al. describe and analyze various cipher modes of operation. We will briefly recall their work on the counter-based XOR mode of operation or XORC, which we use in our own scheme. Assuming that we have a pseudo-random function family F of domain $\{0,1\}^d$ and range $\{0,1\}^D$, the scheme XORC$(F) = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined as follows:

- the function $\mathcal{K}$ flips coins and outputs a random k bit key a, thereby selecting a function f_a from the family F.
- the function $\mathcal{E}(\sigma, x)$ is defined as:

```
let $x = x_1 x_2 ... x_n$
for $i = 1, ..., n$ do $y_i = f_a(\sigma + i) \oplus x_i$.
return $(\sigma, y_1 y_2 ... y_n)$.
$\sigma \leftarrow \sigma + n$
```

- the function $\mathcal{D}(\sigma, y)$ is defined as:

```
let $y = y_1 y_2 ... y_n$
for $i = 1, ..., n$ do $x_i = f_a(\sigma + i) \oplus y_i$
return $x = x_1 x_2 ... x_n$
```

Note: The state *or counter* σ is maintained by the encryption algorithm across consecutive encryptions with the same key. The decryption algorithm is stateless.

The authors of [BDJR97] have shown essentially that if F is a secure pseudo-random function family then the XORC is a secure encryption scheme. Their notion of security is left-or-right indistinguishability in a chosen plaintext attack and we refer the reader to their work for more details. In practice, we will use a family of pseudo-random permutations such as DES or AES[oST01] as satisfying approximation of F.

This scheme has many advantages. First it's paralellizable because the encryption of each block is independent of another. Second, the decryption can under certain circumstances be "prepared" in advance. Since the state is incremented in a predictable way across several messages, it means that the receiver can pre-compute some of the values of f_a to reduce online computations. Finally, this scheme uses the XOR operation which is commutative, a property that we will show to be useful.

8.2.2 Multiple encryptions.

The commutative nature of the XOR operation makes the XORC=$(\mathcal{K},\mathcal{E},\mathcal{D})$ interesting for a special form of multiple encryption. Normally if we encrypt a message x several times with a set of keys $a_1,...,a_m$ we would compute $(\sigma_m, y) = \mathcal{E}_{a_m}(\sigma_m, ...\mathcal{E}_{a_2}(\sigma_2, \mathcal{E}_{a_1}(\sigma_1, x)...)$ but we proceed slightly differently, by leaving the counters $\sigma_1,...,\sigma_m$ outside the consecutive encryptions. We define the XORC$^{(m)}$ = $(\mathcal{K}^{(m)}, \mathcal{E}^{(m)}, \mathcal{D}^{(m)})$ algorithms with m independent keys as as follows:

- $\mathcal{K}^{(m)}$ chooses m random keys.
- $\mathcal{E}^{(m)}_{a_1,a_2,..,a_m}(\sigma_1,...,\sigma_m,x)$ is defined as:

```
let $x = x_1x_2...x_n$
for $i = 1,...,n$ do $y_i = f_{a_1}(\sigma_1 + i) \oplus ... \oplus f_{a_m}(\sigma_m + i) \oplus x_i$.
return $(\sigma_1...\sigma_m, y_1y_2...y_n)$.
for $j = 1,...,m$ do $\sigma_j \leftarrow \sigma_j + n$
```

- $\mathcal{D}^{(m)}_{a_1,...,a_m}(\sigma_1,...,\sigma_m,y)$ is defined as:

```
let $y = y_1y_2...y_n$
for $i = 1,...,n$ do $x_i = f_{a_1}(\sigma_1 + i) \oplus ... \oplus f_{a_m}(\sigma_m + i) \oplus y_i$
return $x = x_1x_2...x_n$.
```

We note immediately that $\mathcal{E}_a = \mathcal{E}^{(1)}_a$ and $\mathcal{D}_a = \mathcal{D}^{(1)}_a$. This form of multiple encryption has the following interesting properties:

Fact 8.1 *For any permutation π of $\{1,...,m\}$ we have*

$$\mathcal{E}_{a_{\pi(1)},...,a_{\pi(m)}}(\sigma_{\pi(1)},...,\sigma_{\pi(1)},x) = \mathcal{E}_{a_1,a_2,..,a_m}(\sigma_1,...,\sigma_m,x)$$

and

$$\mathcal{D}_{a_{\pi(1)},...,a_{\pi(m)}}(\sigma_{\pi(1)},...,\sigma_{\pi(1)},y) = \mathcal{D}_{a_1,a_2,..,a_m}(\sigma_1,...,\sigma_m,y)$$

This is a natural consequence of the commutativity of the XOR binary operation.

Fact 8.2 *Given a message x if we compute $\{\sigma_1,...,\sigma_{(m-1)},y\} \leftarrow \mathcal{E}^{(m-1)}_{a_1,...,a_{(m-1)}}(\sigma_1,...,\sigma_{(m-1)},x)$ and $\{\sigma,z\} \leftarrow \mathcal{E}^{(1)}_a(\sigma,y)$ then we have $\{\sigma_1,...,\sigma_{(m-1)},\sigma,z\} = \mathcal{E}^{(m)}_{a_1,...,a_{(m-1)},a}(\sigma_1,...,\sigma_{(m-1)},\sigma,y)$. A similar result holds for $\mathcal{D}^{(m-1)}$ and $\mathcal{D}^{(1)}$.*

Fact 8.3 *When a message is encrypted with m keys as described above it is at least as secure as anyone of the individual encryptions.*

Proof. (This is a classical result). We will say that an adversary A with access to a "encryption device" or oracle representing an encryption scheme can $(t, q, \mu; \epsilon)$-break that algorithm if A succeeds in breaking the encryption scheme with a probability greater than ϵ by asking at most q oracle queries totaling at most μ bytes and spending at most t units of time (where an oracle query itself costs one unit of time). The property above follows from a simple simulation argument: If an adversary A, which has access to a "encryption device" or oracle representing an $XORC^{(m)}$-encryption, $(t, q, \mu; \epsilon)$ breaks that scheme, then we can design an adversary B_A, with access to a *1*-encryption oracle, which can $(t' = t + (m-1)q, q, \mu; \epsilon)$-break one of the XORC encryptions by simulating the $m-1$ other encryptions with random keys.

8.3 l-Layer Encryption Trees.

8.3.1 Construction

As described in section 1.1.2, we call a tree T a *singular leaf* tree if each leaf in T has a distinct unique parent. We define a function $Depth(N)$ which for a node N returns its depth in the tree, where $Depth(root) = 0$, and we define the function $Parent(N, l)$ which returns the l^{th} parent of node N if it exists or $\emptyset$ otherwise.

We call *"l-layer tree"* the association of a multicast singular leaf tree network with a set of cryptographic transformations designed to protect the distributed data with a varying set of l layers of encryption. We associate a set of keys to the tree to perform XORC encryptions as described previously, taking advantage of the commutative nature of the encryption scheme. Let T be a singular leaf tree with n intermediaries. We associate a set of $n + l$ different encryption keys $[K_1, ..., K_{l+n}]$ to the tree as follows:

root: The root receives the encryption keys $[K_1, ..., K_l]$.

intermediaries: The n intermediaries receive each a distinct key from the set $[K_{l+1}, ..., K_{l+n}]$. For example if we number the intermediary arbitrarily from 1 to n we can associate key K_{l+i} to intermediary number i. We call this key the intermediary's *encryption* key. Each intermediary N receives a secondary key K', which we will call *decryption* key, as follows:

```
if $Depth(N) < l$ then $K' \leftarrow K_{Depth(N)}$.
else $K' \leftarrow$ (the encryption key of $Parent(N, l)$)
```

leaves: The leaves each receive l keys. To clarify the notation we will call these keys $X_1, ..., X_l$, A leaf N receives these keys as follows:

```
for $i = 1, ..., l$ do
(1)  if $Parent(N, i) = \emptyset$ then $X_{(l-i+1)} \leftarrow K_{Depth(N)+i-1}$
(2)  else $X_{(l-i+1)} \leftarrow$ (the encryption key of $Parent(N, l)$)
```

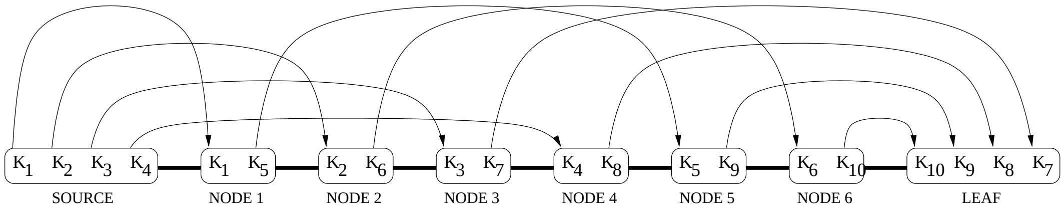

Figure 8.1: Key distribution on a single path in a 4 layer tree.

Line (1) shows that if the leaf does not have an i^{th} parent then it gets one of the encryption keys used by the root and line (2) shows that if it does have an i^{th} parent then it gets the encryption key of that parent.

This key assignment may seem somewhat complex but in fact it's governed by two simple principles:

- Each intermediary gets its own encryption key and the encryption key of its l^{th} parent.
- Each leaf gets all the encryption key of its parents of degree l down to 1.

The complexity only appears in the algorithm for nodes or leaves that are not deep enough in the tree to fully apply the previous two rules. In such a case, the otherwise missing keys are taken from the root. If we focus our attention on a single path of the tree extending from the root to a leaf, we can see that each key used on a node in a path is used once as an encryption key and once as a decryption key, as illustrated on figure 8.1.

An example of our key assignment algorithm is shown on figure 8.2 for a 4 layer tree.

8.3.2 Data Distribution

Once the tree is constructed, its components operate as follows:

root: The root or source encrypts a message M by computing $(\sigma_1, ..., \sigma_l, C) = \mathcal{E}^{(L)}_{K_1,...,K_l}(\sigma_1, ..., \sigma_l, M)$ and sends the result to its children nodes in the tree.

intermediaries: Each intermediary N receives an encrypted message $(\sigma_1, ..., \sigma_l, C)$. The intermediary N performs the following operations:

1. Suppress a layer of encryption: $C' \leftarrow \mathcal{D}^{(1)}(\sigma_1, C)$.
2. Add a new layer of encryption: $(\tau, C'') \leftarrow \mathcal{E}^{(1)}(\tau, C')$ where τ is the internal counter of N.
3. Let $\tau_L \leftarrow \tau$ and $\tau_i \leftarrow \sigma_{(i+1)}$ for $i = 1, ..., (l-1)$. Send $(\tau_1, \tau_2, ..., \tau_l, C'')$ to the children nodes.

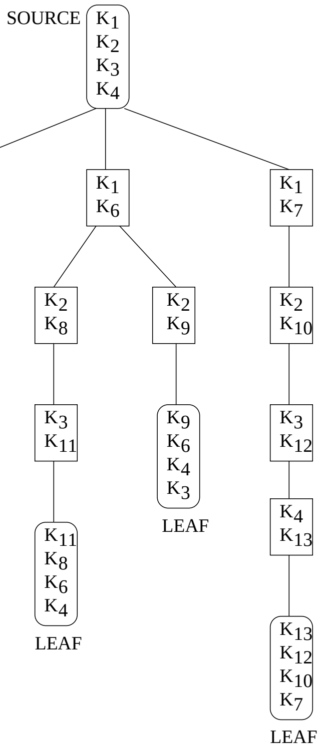

Figure 8.2: Key distribution on a 4 layer tree.

leaves: The leaves receive an encrypted message $(\sigma_1, \sigma_2, ..., \sigma_l, C)$ that they decrypt by computing $M = \mathcal{D}^{(L)}_{X_1,...,X_l}(\sigma_1, ..., \sigma_l, C)$.

If we recall the construction of our tree, we see that the keys are distributed to make the above algorithm work: each key used to encrypt the data is used later as a decryption key. The source and the leaves perform an l-encryption and an l-decryption respectively. Intermediaries use Fact 8.2 to first transform an l-encryption to a $(l-1)$ encryption, then using the same property, they transform the $(l-1)$-encryption back into an l-encryption. The combination of Fact 8.1 and 8.2 allows us to decrypt a layer regardless of the order in which the encryptions were done.

As an example, we will examine how our data distribution algorithm is applied on the path of the 4-layer tree of figure 8.1, where C^r denotes the encryption of M at stage r in the algorithm:

Source: Computes and sends $(\sigma_1, \sigma_2, \sigma_3, \sigma_4, C^0) \leftarrow \mathcal{E}^{(m)}_{K_1,K_2,K_3,K_4}(\sigma_1, \sigma_2, \sigma_3, \sigma_4, M)$

Node 1: Suppresses a layer $(\sigma_2, \sigma_3, \sigma_4, C^1) \leftarrow \mathcal{D}^{(1)}_{K_1}(\sigma_1, C^0)$.

Then it computes and sends $(\sigma_2, \sigma_3, \sigma_4, \sigma_5, C^2) \leftarrow \mathcal{E}^{(1)}_{K_5}(\sigma_5, C^1)$.

Node 2: Suppresses a layer $(\sigma_3, \sigma_4, \sigma_5, C^3) \leftarrow \mathcal{D}^{(1)}_{K_2}(\sigma_2, C^2)$.

Then it computes and sends $(\sigma_3, \sigma_4, \sigma_5, \sigma_6, C^4) \leftarrow \mathcal{E}^{(1)}_{K_6}(\sigma_6, C^3)$.

Node 3: Suppresses a layer $(\sigma_4, \sigma_5, \sigma_6, C^5) \leftarrow \mathcal{D}^{(1)}_{K_3}(\sigma_3, C^4)$.
Then it computes and sends $(\sigma_4, \sigma_5, \sigma_6, \sigma_7, C^6) \leftarrow \mathcal{E}^{(1)}_{K_7}(\sigma_7, C^5)$.

Node 4: Suppresses a layer $(\sigma_5, \sigma_6, \sigma_7, C^7) \leftarrow \mathcal{D}^{(1)}_{K_4}(\sigma_4, C^6)$.
Then it computes and sends $(\sigma_5, \sigma_6, \sigma_7, \sigma_8, C^8) \leftarrow \mathcal{E}^{(1)}_{K_8}(\sigma_8, C^7)$.

Node 5: Suppresses a layer $(\sigma_6, \sigma_7, \sigma_8, C^9) \leftarrow \mathcal{D}^{(1)}_{K_5}(\sigma_5, C^8)$.
Then it computes and sends $(\sigma_2, \sigma_3, \sigma_4, \sigma_5, C^{10}) \leftarrow \mathcal{E}^{(1)}_{K_9}(\sigma_9, C^9)$.

Node 6: Suppresses a layer $(\sigma_7, \sigma_8, \sigma_9, C^{11}) \leftarrow \mathcal{D}^{(1)}_{K_6}(\sigma_6, C^{10})$.
Then it computes and sends $(\sigma_7, \sigma_8, \sigma_9, \sigma_{10}, C^{12}) \leftarrow \mathcal{E}^{(1)}_{K_{10}}(\sigma_{10}, C^{11})$.

Leaf: Decrypts the message $M \leftarrow \mathcal{D}^{(m)}_{K_7,K_8,K_9,K_{10}}(\sigma_7, \sigma_8, \sigma_9, \sigma_{10}, C^{12})$.

8.3.3 Membership Management

After describing how members access the multicast content in the previous section, we will now turn our attention to the addition and removal of members in our framework.

Add: When a recipient M wants to be added to the group, he contacts a membership manager (MM) with an authenticated secure channel. If M is allowed to access the group, then there are 2 possible scenarios:

1. M is already physically in an existing leaf F: the MM sends an authenticated secure message to the parent intermediary P of F, to change the encryption key K of P to a new value K'. Then the new key K' is sent to all the members of the same leaf and to the new member M.
2. M is not in an existing leaf: the tree is expanded to create a new leaf for M. The corresponding keys are distributed to the new intermediaries and M.

Remove: When a member needs to be removed from the group, the MM sends an authenticated secure message to the parent intermediary P of F, to change the encryption key K of P to a new value K'. Then the new key K' is sent to all the members of the same leaf except M. If the leaf is empty because the last member left, than after a certain delay, we may remove unused intermediaries from the tree.

The leaf holds l keys and needs all of them to access the data. Thus, changing just one of them when we add or remove a member provides us with forward and backward secrecy (req. 5.2).

Distributed Membership Management.

This scheme also lends itself to a certain form of decentralized key management. The tree of intermediary elements can be managed by a hierarchy of membership managers. It follows from our construction in section 8.3 that an individual membership manager only needs to know l extra keys to manage a subtree on its own. More precisely, if a membership manager is selected to manage a subtree consisting of an intermediary N and all its descendants in the tree, then it needs to know the set $Y_1, ..., Y_l$ of keys defined as follows:

```
for $i = 1, ..., l$ do
    if $Parent(N, i) = \emptyset$ then $Y_i \leftarrow K_{Depth(N)+i-1}$
    else $Y_i \leftarrow$ (the encryption key of $Parent(N, i)$)
```

In turn a membership manager may delegate the management of some of its own subtrees to several other membership managers.

8.4 Security Requirements

8.4.1 Encryption

To discuss the security of our construction, we will first look at *one-layer* trees before we study the general case. One layer trees are conceptually very simple, since they only use the original XORC encryption algorithm. The source has a key K_1 and uses it to encrypt data to be sent to its children. The intermediaries decrypt the data with the key K_j that their parents used to encrypt the data and use their own key K_i to encrypt the data again for their children. The leaves use a single key to access the data. A one-layer tree is quite similar to the IOLUS framework [Mit97], and it shares one of the drawbacks of that framework: each intermediary is trusted to access the cleartext data. For now however, let's examine the security of a one-layer tree while making the hypothesis that the intermediary elements are secure.

The individual links are secured by the XORC encryption algorithm. In our framework, an adversary has the ability to observe several links and thus the same message encrypted under different keys. We can even imagine that the adversary may modify or input new messages at different points in the tree to try to break the security of the system. In a recent work evaluating the security of public key cryptosystems in the multiuser setting [BBM00], Bellare et al. have shown essentially that if a public key cryptosystem is secure in the sense of indistinguishability, then it implies the security of the cryptosystem in the multiuser setting, where related messages are encrypted with different keys. We refer the reader to their work for further details [BBM00]. Though their work was targeted at public key cryptosystems, their results can be applied to the private key setting, and since the XORC encryption is secure in the sense of "indistinguishability" under chosen plaintext attacks[BDJR97], we can assert the security of the whole tree by using the results of [BBM00].

Now for l-layer trees, property 8.3 tells us that they are at least as secure as a 1-layer tree if no intermediary is compromised. But, the advantage of a l-layer tree is that it remains secure even if some nodes are compromised, more precisely:

Proposition 8.4 Let $B = B_1, ..., B_p$ define a set of p compromised intermediaries in a L-layer tree. The tree remains secure as long as there exists a constant $c \in \{0, ..., l-1\}$ such that $Depth(B_i) \neq c \bmod l$ for all $i \in \{1, ..., p\}$.

Proof. This property derives from the arrangement of the keys in the tree. Let $B = B_1, ..., B_p$ define a set of compromised intermediaries in an l-layer tree T such that there exists a constant $c \in \{0, ..., l-1\}$ verifying $Depth(B_i) \bmod l \neq c$ for all $i \in \{1, ..., p\}$. From the tree T we can extract a subtree $\overline{T}$ iteratively, as follows:

Notations:

Let N_0 define the root of T, let $\{N_1, ..., N_n\}$ denote the intermediaries of T and let $\mathcal{L}_{k>0}$ denote the leafs of T.

Similarly, let $\overline{N}_0$ define the root of $\overline{T}$, let $\{\overline{N}_1, ..., \overline{N}_q\}$ define the intermediary nodes of $\overline{T}$, and let $\overline{\mathcal{L}}$ denote the leafs of $\overline{T}$.

Construction:

```
$\overline{N}_0 \leftarrow N_0$
Select $\{N_1, ..., N_q\}$, the set of intermediaries $N_i$ of $T$ which verify $Depth(N_i) = c \bmod l$.
$\{\overline{N}_1, ..., \overline{N}_q\} \leftarrow \{N_1, ..., N_q\}$
for $i = 1, ..., q$ do
  if $Depth(N_i) = c$ then
      connect $\overline{N}_i$ to $\overline{N}_0$ in $\overline{T}$.
      let $\overline{\mathcal{L}}$ be the concatenation of all leaves $\mathcal{L}_k \in T$ such that $Depth(\mathcal{L}_k) \leq c$.
      if $\overline{\mathcal{L}} \neq \emptyset$ then connect $\overline{\mathcal{L}}$ to $\overline{N}_0$ in $\overline{T}$.
  else
      let $N_j = Parent(N_i, l)$.
      connect $\overline{N}_j$ to $\overline{N}_i$ in $\overline{T}$.
      let $\overline{\mathcal{L}}$ be the concatenation of all leaves $\mathcal{L}_k \in T$ such that
            $(Parent(\mathcal{L}_k, r) = N_i$ and $j \leq l)$.
      if $\overline{\mathcal{L}} \neq \emptyset$ then connect $\overline{\mathcal{L}}$ to $\overline{N}_i$ in $\overline{T}$.
```

The tree $\overline{T}$ represents a 1-layer tree such that none of its intermediaries $\{\overline{N}_1, ..., \overline{N}_q\}$ hold a key in common with any of those distributed to the compromised set B. Thus since there exists an independent 1-layer tree between the root and the leaves, the encryption of data in the tree remains secure (req. 5.1). ◇

Corollary 8.5 An obvious implication of this property is that an l-layer tree can at least withstand the compromise of any set of less than l intermediaries.

8.4.2 Containment

In singular parent trees, no leaf shares its direct parent with another leaf, thus each leaf holds at least one key that is not known by any other leaf. This key is the encryption key used by the parent intermediary node of the leaf. Thus if $\mathcal{L}$ is a leaf, then no collusion of any group of leaves $\{\mathcal{L}_1, ..., \mathcal{L}_p | \mathcal{L}_i \neq \mathcal{L}, i \in \{1, ..., p\}\}$ can break the encryption of data received in the leaf $\mathcal{L}$ (req. 5.3). Moreover, an adversary in a leaf $\mathcal{L}$ who compromises the keys in a set of leaves $\{\mathcal{L}_1, ..., \mathcal{L}_p | \mathcal{L}_i \neq \mathcal{L}, i \in \{1, ..., p\}\}$

cannot use this information to access the data in his own leaf. Thus having a *singular leaf* tree is a sufficient condition to provide containment (req. 5.4).

There is no containment within a leaf, all the recipients that are physically in the same leaf use the same key to access the data, thus exposure of keying material in one leaf allows other members of the same leaf to access the data. However, unless there is a form of hardware access control installed directly on each recipient, providing containment in within a leaf is very hard: ultimately, it's difficult to stop or even detect if a member rebroadcasts decrypted data to other local recipients that are not members themselves.

8.4.3 Hybrid attacks

This framework may face more complex attacks that are a combination of both the compromise of leaves and intermediary elements:

Membership extensions:

If a member M in a leaf $\mathcal{L}$ takes full control of the direct parent P of $\mathcal{L}$, it can monitor key changes in P. If the membership manager decides to remove M from the group, it will change the key K held by P and send the new updated key K' to the other remaining members in $\mathcal{L}$ as well as P. As a consequence the removed member M will still be able to stay in the group because it will learn the new value K' from P. This means that we lose forward secrecy. Recovering from such a compromise requires a key change in the parent P' of the compromised node P, which in turn requires all the leaves that have P' as an ancestor to be updated.

Containment failures:

Assume that two leafs $\mathcal{L}_1$ and $\mathcal{L}_2$ of same depth in the tree share a common ancestor node P in the tree which verifies $|Depth(\mathcal{L}_1) - Depth(P)| \leq l$. In that situation, the members in $\mathcal{L}_1$ and the members in $\mathcal{L}_2$ have $k < l$ decryption keys in common. If a member M_1 in $\mathcal{L}_1$ compromises the $(l - k)$ first parents of $\mathcal{L}_2$ than M_1 will know enough information to generate the set of l keys used in $\mathcal{L}_2$, by combining the k common keys with the $l - k$ compromised keys. This attacks breaks the containment property of the scheme, for two leafs that are at the same depth in the tree.

8.5 Scalability Requirements

The processing load supported by each entity in the tree is not proportional to the group size. For the leaves and the root it depends on the parameter l, which defines the number of layers in the tree, while intermediaries always perform a single decryption and a single encryption regardless of the number of layers. Thus this framework offers processing scalability (req. 5.6).

When a member is removed or added to the group, the key change remains local and only affects a leaf at a time. The number of elements in a leaf is not a scalability itself itself, because we can simply create more branches in the tree to cope with leaves that get too large. As in the construction of the previous chapter, membership robustness issues (req. 5.7) are limited to a subgroup rather than the whole group.

8.6 Reducing Expansion

In the XORC^m scheme, the encryption of a message results in an expansion of $m.|\sigma_i|$ bytes where $|\sigma_i|$ represents the size in bytes of the state value. We could use a single state chosen by the source and common to all layers of encryption as well as all elements in the tree. In other words we would rewrite the encryption algorithm as follows:

```
$\mathcal{E}^{*(m)}_{a_1,a_2,..,a_m}(\sigma, x)$:
split $x$ as $x_1, ..., x_n$
**for** $i = 1, ..., n$ **do** $y_i = f_{a_1}(\sigma + i) \oplus ... \oplus f_{a_m}(\sigma + i) \oplus x_i$.
**return** $(\sigma, y_1 y_2 ... y_n)$
$\sigma \leftarrow \sigma + n$
```

The intermediaries would use the same σ to both encryption and decryption operations. The algorithm would be simplified and the ciphertext size would be independent of the number of layers in the tree. In such a case, however, proving the security of the scheme is an open problem since the intermediaries are now stateless and cannot be modeled as independent encrypting devices, which was a requirement of the security proof found in [BBM00] upon which we relied for our scheme.

8.7 Conclusion

As we did in the previous chapter, we used intermediary elements in the network to constructed a scalable framework for multicast access control based on symmetric cryptographic transforms. This framework offers interesting properties such as containment, and limited trust in the intermediary elements of the network. It shows some vulnerabilities to what we called hybrid attacks when both members and intermediary elements in the network are compromised, in particular what we called the "*leaf parent*" node.

Non trusted re-encryption trees are an interesting approach to multicast confidentiality. However, there remains some challenges for their deployment. First, we need to evaluate which nodes should become active security intermediaries and how to deal with routing or node failures. Second, we need to establish a well defined method to distribute the security parameters to these active intermediaries.

Chapter 9

Conclusion

9.1 The Complexity of Multicast Security

In this dissertation, we analyzed and suggested solutions for the two most important security issues in large scale multicast applications: authentication and confidentiality. The set of requirements that we established for multicast authentication in Chapter 2 and for multicast confidentiality in Chapter 5 are clearly more complex than their unicast counterparts.

In this thesis we highlighted several reasons to this additional complexity. First, the participants in a multicast security protocol change dynamically through time. Adapting to dynamic groups adds certain requirements, such as *joinability* for authentication or *backward and forward secrecy* for confidentiality. Second, we have a different notion of trust in the participating entities of multicast security protocols. In a good multicast authentication scheme, though the source will share some security parameters with the recipients, these recipients should not have the ability to forge packets that seem to originate from the source. Similarly, we introduced *containment* in multicast confidentiality to limit the impact of potential key exposure, because we cannot trust the recipients with the same strength as the source or the membership managers. More generally, the fundamental source of additional complexity in multicast security protocols can be found each time the need of one or few participants is opposed to the rest of the participants in the protocol. For authentication, the source is opposed to the rest of the recipients, which are treated as potential adversaries. For confidentiality, added and removed members are in opposition with the rest of the group. Properties such as containment, joinability, backward and forward secrecy can all be viewed as resulting from the opposition between an individual entity and the rest of the group.

9.2 Maturity of Multicast Security Solutions

Authentication

Multicast authentication is essentially a one-way protocol where a source provides information to a set of recipients for verification. All practical protocols we presented in this thesis rely partly on a digital signature verification mechanism which requires the recipients to acquire the public key of the source. Besides this minimal setup, the recipients don't communicate with any additional entity or back with the source in order to authenticate packets. For this reason, multicast authentication protocol design and specification seems like a less complex problem than for confidentiality. As a consequence, we believe that standards for multicast authentication will appear sooner than for confidentiality.

Confidentiality

Multicast confidentiality is a more complex issue, because it involves membership managers that communicate on a bidirectional channel with members in different situations. Note that specific membership management issues are rarely mentioned in multicast confidentiality frameworks, however we believe that these issues would deserve to be explored in future research. For example, in large commercial applications, distributed membership management will become a needed feature. One reason is scalability. Indeed, in a very large multicast group, a single membership manager serving all requests, would quickly become a bottleneck. A second reason relates to laws and regulations. Different countries have different commercial laws and sometimes different cryptography regulations. It makes sense to distribute several membership managers according to these constrains. Re-encryption trees are well suited for distributed membership management, and the MARKS subscription approach is even better. However, it seems that a construction such as LKH is inherently hard to distribute.

Someone who wishes to implement multicast confidentiality as described in this dissertation is faced with a dilemma. If he uses approaches that share a common key among members, without any form of containment, then piracy related issues will greatly hinder the use of such schemes in large groups. On the other hand the schemes that provide containment, require the use of intermediary elements in the network, which has a deployment cost. Note that the new approach that we suggested, which uses only semi-trusted intermediary elements in the network, partially reduces that cost because it does not require the content provider to own the whole infrastructure, but allows it to be shared among several providers.

Because of all these issues, it is not clear yet what type of solution will emerge for confidentiality in large scale multicast applications. In fact it is quite possible that some solutions will rely on a combination of both re-encryption trees for inter-domain communications in combination with a LKH approach in local domains.

9.3 Future Directions

In recent years, content providers have shown a strong concern in protecting the content they distribute beyond the notion confidentiality we described in this thesis. Recall for example, the highly publicized lawsuit[USDC99] that opposed the RIAA to Napster, the mp3 audio download service. Issues such as copy protection and copyright have become paramount for content distributors. Architectures that provide these additional services typically rely on several cooperating tamper-proof elements that provide end to end protection to the content from the source directly to the video or audio playback hardware (see for example, CPRM, `http://www.4Centity.com/tech/cprm/`). Future commercial multicast confidentiality frameworks will potentially rely on such tamper-proof hardware platforms.

The non-negligible level of piracy that affects the Digital Video Broadcasting industry which uses tamper-proof hardware to provide access control is one of many examples that show that these technologies have their own source of challenges. Developing and extending these technologies in the context of the Internet to provide new security services for multicast applications presents a potentially strong area of research. We believe that some of the ideas we introduced in this thesis, such as containment and the use of untrusted elements to perform security transforms in a network will find good applications in this extended context.

Bibliography

[ABR98] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHAES: An encryption scheme based on the diffie-hellman problem. *Submited to IEEE P1363a*, 1998.

[Aut] Internet Assignement Authority. Internet multicast addresses. http://www.iana.org/assignments/multicast-addresses.

[Bal97] A. Ballardie. Core based trees (CBT version 2) multicast routing. Request For Comment 2189, September 1997.

[BBM00] M. Bellare, A. Boldyreva, and Silvio Micali. Public-key encryption in a multiuser setting: Security proofs and improvements. In *Eurocrypt 2000*, volume LNCS 1807, pages 259–274. Springer Verlag, 2000.

[BC93] J. Bolot and H. Crépin. Analysis and control of audio packet loss over packet-switched networks. Technical report, INRIA, 1993.

[BCK96] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. *Lecture Notes in CS*, 1109:1–15, 1996.

[BD95] M. V. D. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In Alfredo De Santis, editor, *Advances in Cryptology - EuroCrypt '94*, pages 275–286, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science Volume 950.

[BDF01] Dan Boneh, Glenn Durfee, and Matt Franklin. Lower bounds for multicast message authentication. In *Theory and Application of Cryptographic Techniques*, pages 437–452, 2001.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *IEEE Symposium on Foundations of Computer Science*, pages 394–403, 1997.

[BFPT99] Jean-Chrysostome Bolot, Sacha Fosse-Parisis, and Donald F. Towsley. Adaptive FEC-based error control for internet telephony. In *INFOCOM (3)*, pages 1453–1460, 1999.

[BLMR98] John Byers, Michael Luby, Michael Mitzenmacher, and Ashu Rege. A digital fountain approach to reliable distribution of bulk data. In *proceedings of ACM SIGCOMM '98*, September 1998.

[Bon99] Boneh. Twenty years of attacks on the RSA cryptosystem. *NOTICES: Notices of the American Mathematical Society*, 46, 1999.

[BR95] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology - EuroCrypt '94*, pages 92–111, Berlin, 1995. Springer-Verlag. Lecture Notes in Computer Science Volume 950.

[Bri99] Bob Briscoe. MARKS: Zero side-effect multicast key management using arbitrarily revealed key sequences. In *First International Workshop on Networked Group Communication*, November 1999.

[BSUB98] M. Borella, D. Swider, S. Uludag, and G. Brewster. Internet packet loss: Measurement and implications for end-to-end QoS. In *International Conference on Parallel Processing*, August 1998.

[CEK$^+$99] Isabella Chang, Robert Engel, Dilip Kandlur, Dimitrios Pendarakis, and Debanjan Saha. Key management for secure internet multicast using boolean function minimization techniques. In *Proceedings IEEE Infocomm'99*, volume 2, pages 689–698, March 1999.

[CGI$^+$99] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of IEEE Infocom'99*, 1999.

[CVSP98] G. Caronni, M. Valdvogel, D. Sun, and B. Plattner. Efficient security for large and dynamic multicast groups. In *Proceedings of IEEE WET-ICE'98*, 1998.

[CW93] K.W. Campbell and M.J. Wiener. DES is not a group. In *In Advances in Cryptology - Crypto '92*, pages 512–520. Springer-Verlag, 1993.

[CWSP98] Germano Caronni, Marcel Waldvogel, Dan Sun, and Berhardt Plattner. Efficient security for large and dynamic multicast groups. In *IEEE 7th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '98)*, 1998.

[Dee89] Steve E. Deering. RFC 1112: Host extensions for IP multicasting, Aug 1989.

[Dee91] Steve E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, 1991.

[DFFT95] Martin Dyer, Trevor Fenner, Alan Frieze, and Andrew Thomason. On key storage in secure networks. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 8(4), 1995.

[EFH$^+$97] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast-sparse mode (PIM-SM): Protocol specification. Request For Comment 2117, July 1997.

[EGM96] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[Elg84] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Crytology - CRYPTO'84, Santa Barbara, California, USA*, 1984.

[Fen97] W. Fenner. Internet group management protocol, version 2. Request For Comments 2236, November 1997. see also draft-ietf-idmr-igmpv3-and-routing-01.txt for IGMP v3.

[FLYV92] V. Fuller, T. Li, J. Yu, and K. Varadhan. Supernetting: an address assignment and aggregation strategy, 1992.

[FN93] A. Fiat and M. Naor. Broadcast encryption. In Douglas R. Stinson, editor, *Advances in Cryptology - Crypto '93*, pages 480–491, Berlin, 1993. Springer-Verlag. Lecture Notes in Computer Science Volume 773.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *CRYPTO*, pages 276–288, 1984.

[GM01] P. Golle and N. Modadugo. Streamed authentication in the presence of random packet loss. In *to appear in NDSS 2001.*, 2001.

[GR97] Rosario Gennaro and Pankaj Rohatgi. How to sign digital streams. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - Proceedings of CRYPTO 97*, volume 1294 of *Lecture Notes in Computer Science*, pages 180–197, Santa Barbara, California, USA, August 1997. Springer Verlag.

[GS00] C. M. Grinstead and J. L. Snell. *Introduction to Probability.* McGraw Hill, 2000.

[HC99] H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proceedings of ACM SIGCOMM'99*, Harvard University, September 1999. ACM SIGCOMM.

[HCD00] Thomas Hardjono, Brad Cain, and Naganand Doraswamy. A framework for group key management for multicast security. Internet-Draft, work in progress, February 2000.

[Häg02] Olle Häggström. *Finite Markov Chains and Algorithmic Applications.* Cambridge, June 2002.

[HK89] Lein Harn and Thomas Kiesler. Authenticated group key distribution scheme for a large distributed network. In *Symposium on Security and Privacy*, 1989.

[HM97] H. Harney and C. Muckenhirn. Group key management protocol (GKMP) specification. RFC 2093, July 1997.

[HPS98] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. *Lecture Notes in Computer Sciences*, 1423:267–??, 1998.

[KRS85] B. S. Kaliski, R. L. Rivest, and A. T. Sherman. Is the Data Encryption Standard a group? In *Advances in Crytology - CRYPTO'85, Santa Barbara, California, USA*, 1985.

[LMS$^+$97] Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical loss-resilient codes. In *ACM Symposium on Theory of Computing*, pages 150–159, 1997.

[LRW00] H. Lipmaa, P. Rogaway, and D. Wagner. Comments to NIST concerning AES modes of operation: CTR-Mode encryption. In *NIST First Modes of Operation Workshop*, Baltimore, Maryland, USA, October 20 2000.

[LYGL01] Xiaozhou Steve Li, Yang Richard Yang, Mohamed G. Gouda, and Simon S. Lam. Batch rekeying for secure group communications. In *Tenth International World Wide Web conference*, pages 525–534, 2001.

[Mer89] Ralph C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435, pages 218–238. Springer-Verlag, August 1989.

[Mit97] Suvo Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM'97 (September 14-18, 1997, Cannes, France)*, 1997.

[Moy94] John Moy. Multicast extensions to OSPF. Request For Comment 1584, March 1994.

[MP99] Refik Molva and Alain Pannetrat. Scalable multicast security in dynamic groups. In *Proceedings of the 6th ACM conference on Computer and Communications Security*, pages 101–112, Singapore, November 1999. Association for Computing Machinery.

[MP00] Refik Molva and Alain Pannetrat. Scalable multicast security with dynamic recipient groups. *ACM Transactions on Information and System Security*, 3(3):3, 2000.

[MS97] T. Maufer and C. Semeria. Introduction to IP multicast routing. Internet-Draft, July 1997. draft-ietf-mboned-intro-multicast-03.txt.

[MS98] David A. McGrew and Alan T. Sherman. Key establishment in large dynamic groups using one-way function trees. Technical report, TIS Labs at Network Associates, Inc., Glenwood, MD, 1998.

[MS01] Sara Miner and Jessica Staddon. Graph-based authentication of digital streams. In *2001 IEEE Symposium on Security and Privacy*, May 2001.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Nat95] National Institute of Standards and Technology. Secure hash standard, 1995.

[NBT98] Jörg Nonnenmacher, Ernst W. Biersack, and Don Towsley. Parity-based loss recovery for reliable multicast transmission. *IEEE/ACM Transactions on Networking*, 6(4):349–361, 1998.

[Obr98] K. Obraczka. Multicast transport protocols: a survey and taxonomy. *IEEE Communications Magazine*, 36(1):94–102, jan 1998.

[oST01] National Institute of Standards and Technology. Advanced Encryption Standard, 2001.

[Pax99] Vern Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[PB99] R. Poovendran and John S. Baras. An information theoretic analysis of rooted-tree based secure multicast key distribution schemes. In *CRYPTO*, pages 624–638, 1999.

[PCTS00] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, May 2000.

[PM02a] Alain Pannetrat and Refik Molva. Authenticating real time packet streams and multicasts. In *The Seventh IEEE Symposium on Computers and Communications*, Taormina, Italy, July 2002.

[PM02b] Alain Pannetrat and Refik Molva. Multiple layer encryption for multicast groups. In *the sixth IFIP Communications and Multimedia Security Conference*, Portoroz, Slovenia, September 2002.

[PST01] Adrian Perrig, Dawn Song, and Doug Tygar. ELK, a new protocol for efficient large-group key distribution. In *IEEE Symposium on Security and Privacy*, May 2001.

[Rab89] Michael O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[RC98] Phillip Rogaway and Don Coppersmith. A software-optimized encryption algorithm. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 11(4):273–287, Fall 1998.

[Riv92] R.L. Rivest. The MD5 message-digest algorithm, April 1992.

[Riv95] R.L. Rivest. The RC5 encryption algorithm. In In B. Preneel, editor, *Fast Software Encryption*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96. Springer Verlag, 1995.

[Riz97] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACMCCR: Computer Communication Review*, 27, 1997.

[Roh99] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *Proceedings of the 6th ACM conference on Computer and Communications Security*, pages 93–100, Singapore, November 1999. Association for Computing Machinery.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21(2):120-126*, 21(2):120–126, 1978.

[RSA99] RSA Security Inc. PKCS-1 v2.1: RSA cryptography standard, 1999.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. Request for Comments 1889, January 1996.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[STW96] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM Conference on Communications Security (March 14-16, 1996, New Delhi, India)*, 1996.

[STW98] Michael Steiner, Gene Tsudik, and Michael Waidner. CLIQUES: A new approach to group key agreement. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, pages 380–387, Amsterdam, May 1998. IEEECSP.

[Tur94] T. Turletti. The INRIA videoconferencing system *ivs*. *ConneXions - The Interoperability Report Journal*, 8(10):20–24, October 1994.

[USDC99] Northern District of California United States District Court. RIAA v. Napster, court's 512(a) ruling, 1999.

[WGL98] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. In *ACM SIGCOMM 1998*, pages 68–79, 1998.

[WHA98] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key management for multicast: Issues and architectures. Internet draft, Network working group, september 1998, 1998.

[WL99] Chung Kei Wong and Simon S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Transactions on Networking*, 7(4):502–513, 1999.

[WL00] C. Wong and S. Lam. Keystone: a group key management system. In *Proceedings of International Conference in Telecommunications*, 2000.

[WP98] D. Waitzman and C. Partridge. Distance vector multicast routing protocol. Request For Comments 1075, November 1998.

[YKT96a] M. Yajnick, J. Kurose, and D. Tosley. Packet loss correlation in the Mbone multicast network. Technical Report 96-32., UMCASS CMPSCI, 1996.

[YKT96b] M. Yajnick, J. Kurose, and D. Tosley. Packet loss correlation in the mbone multicast network. In *IEEE Global Internet Conference*, London, November 1996.

[YMKT99] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and modeling of the temporal dependence in packet loss. In *IEEE INFOCOM*, New York, March 1999.

Curriculum Vitae

Name:	Alain Pannetrat
Date of birth:	July 20, 1974.
Nationality:	French

1998-2002	Ph.D. Student and Research Assistant University of Nice in Sophia-Antipolis Institut Eurecom, Corporate Communications Departement Sophia-Antipolis
1997-1998	M.Sc. in Computer Science, DEA Networks and Distributed Systems University of Nice in Sophia-Antipolis
1995-1997	B.Sc. in Computer Science, Licence & Maîtrise Faculty of Science, University of Poitiers, France
1993-1995	University Degree in Mathematics, DEUG MIAS Faculty of Science, University of Poitiers, France

Publications

- Refik Molva, Alain Pannetrat, Scalable Multicast Security in Dynamic Groups. *Proceedings of the 6th ACM Conference on Computer and Communications Security, November 1999, Singapore.*

- Refik Molva, Alain Pannetrat, Scalable Multicast Security with Dynamic Recipient Groups. *ACM Transactions on Information and System Security, 3, August 2000.*

- Sergio Loureiro, Refik Molva, Alain Pannetrat, Secure data collection with updates. *Electronic Commerce Research Journal, 1/2:119-130, February/March 2001.*

- Alain Pannetrat, Refik Molva, Authenticating Real Time Packet Streams and Multicasts. *The Seventh IEEE Symposium on Computers and Communications, Taormina, Italy ,July 2002.*

- ALAIN PANNETRAT, REFIK MOLVA, Multiple Layer Encryption for Multicast. *The sixth IFIP Communications and Multimedia Security Conference, Portoroz, Slovenia, September 2002.*
- REFIK MOLVA, ALAIN PANNETRAT, Network Security in the Multicast Framework. *Tutorial for the IFIP Technical Committee on Communication Systems (TC6), Networking 2002 Conference. May 19-24 2002, Pisa - ITALY.*
- ALAIN PANNETRAT, REFIK MOLVA, Real Time Multicast Packet Authentication. *In submission*

Résumé

IP-Multicast est un mécanisme qui permet à une source de distribuer des données à un ensemble presque illimité de clients sur Internet. Ce mécanisme semble particulièrement bien adapté pour des applications de distribution commerciale de contenu à grande échelle comme, par exemple, des chaînes à péage, la diffusion de valeurs boursières ou la mise à jour de logiciels... Force est de constater que ces applications n'ont pas vu le jour comme on aurait pourtant pu l'imaginer. Un frein majeur au développement de ces applications est le manque de sécurité des communications en multicast. Il est clair qu'une distribution de contenu commercial nécessite de fournir des mécanismes qui restreignent l'accès au contenu distribué en multicast aux seuls clients légitimes de l'application. D'autre part, dans de nombreux scénarios le client doit pouvoir s'assurer de l'origine des données reçues et le producteur de contenu veut garantir cette origine afin de se protéger contre les risques d'une usurpation d'identité. Dans le cadre d'un réseau ouvert comme Internet, on fait appel à des techniques cryptographiques éprouvées pour résoudre ces problèmes entre deux entités communicantes. Mais pour des raisons de facteur d'échelle et parfois même de sécurité, ces techniques ne peuvent pas s'étendre au multicast. Cette thèse se focalise donc sur la fourniture de services de confidentialité et d'authentification spécifiquement pour les applications multicast à grande échelle.

Cette dissertation se divise donc en deux volets orthogonaux mais complémentaires : l'authentification et la confidentialité. Dans chacun de ces volets nous proposons d'abord une analyse détaillée de ces problèmes et nous mettons en valeur certains aspects nouveaux ou négligés qui sont spécifiques au multicast. Ensuite nous présentons les principales solutions existantes, en analysant leurs avantages et leurs limites. Nous terminons par nos propres solutions originales, en mettant en valeur les avantages qu'elles offrent par rapport aux solutions précédentes.

Abstract

IP-Multicast is a mechanism that allows a source to transmit packets to an almost unlimited number of recipients over the Internet. This mechanism would seem to be particularly well suited for large scale commercial content distribution, such as, for example, pay-TV, stock quote distribution, or software updates. However, a large scale deployment of any of these applications remains to be seen. One of the major reasons that has hindered the deployment of such applications is the lack of security protocols for multicast communications. Clearly, in many cases, the distribution of content with a commercial value requires the use of mechanisms that restricts access to the content solely to the legitimate recipients. Moreover, in many scenarios the recipient needs to ascertain the origin of the multicast content he receives and the content provider will also want to provide such a guaranty to protect himself from the potentially devastating effect of being impersonated by a third party. In an open network such as the Internet, well studied and reliable cryptographic techniques are used to provide this type of security in two party protocols. However, for scalability and sometimes even for security reasons, these techniques cannot easily be extended to the multicast setting.

The goal of this thesis is thus to study and provide basic security services designed specifically for large scale multicast applications. This dissertation is divided in two orthogonal but complementary themes: authentication and confidentiality. For each theme, we start with a detailed analysis of the problem, while highlighting new or neglected aspects of security that are specific to multicast. Then, we review existing solutions, analyzing their advantages and their limitations. Finally, we provide our own original solutions, highlighting the advantages they offer over previous proposals.