ENSEEIHT
Toulouse
2010

Utilisation d'un nouveau type de matériel de sécurité avec IPsec/IKEv2

Aurélien Wailly

France Télécom - Orange Labs R&D
38 avenue du Général Leclerc
Issy-les-moulineaux
Directeur de recherche : Jean-Michel Combes
Rapporteur universitaire : Anas Abou El Kalam

Table des matières

Table des figures

Chapitre 1

Introduction

Le stage de fin d'étude s'est déroulé à France Télécom dans le département sécurité. Une partie des recherches est dédiée à l'exploitation et à l'optimisation d'IPsec par les applications et les nœuds terminaux.

IPsec est un protocole visant à sécuriser les communications IP.

1.1 Présentation de la compagnie

Dans le monde actuel des télécommunications, les compagnies se doivent d'innover pour maintenir leurs positions commerciales. Pour France Télécom - une des plus grandes entreprises de télécommunications - l'innovation est au cœur de la stratégie de développement.

Depuis sa création, France Télécom a été la principale compagnie de télécommunication française. Elle a su garder sa position malgré sa transformation radicale d'opérateur téléphonique traditionnel vers un fournisseur de services complexes. Aujourd'hui, France Télécom compte plus de 182 millions de clients sur les cinq continents. En Europe, France Télécom est le premier fournisseur de services Internet large bande et le troisième en téléphonie.

Une étape importante dans l'histoire de France Télécom est l'acquisition d'Orange - une société de mobile Britannique. En Juin 2006, Orange devient la marque officielle du France Télécom pour Internet, la télévision et les mobiles.

Actuellement, France Télécom est un fournisseur d'accès au travers de quatre plateformes : téléphonie fixe, accès large bande, téléphonie mobile et plus récemment la télévision sur IP. Avec plus de 191 000 employés (dont 45% en dehors de la France) et un chiffre d'affaire de 53 milliards d'euros, France Télécom est un des opérateurs les plus connus et les plus respectés de part le monde.

Plus de 70% des services proposés par France Télécom sont directement issus de la division Recherche et Développement d'Orange Labs.

1.1.1 Orange Labs

Orange Labs forme le réseau d'innovation du groupe France Télécom Orange. En effet, Orange Labs est présent dans 9 pays sur 4 continents. Cette distribution géographique variée permet d'anticiper les progrès technologiques et les changements tout autour de la planète.

Pas moins de 3400 chercheurs font partie de la division Recherche et Développement de France Télécom. Le rôle de cette division est d'innover en anticipant les technologies futures. Avec plus de 8000 brevets, la R&D excèle dans ce domaine.

Afin d'organiser cette recherche et de développer des compétences spécifiques, la division R&D est organisée en différents départements :

- Integrated Residential & Personal Services (SIRP)
- Services to companies (BIZZ)
- Middleware and Advanced Platforms (MAPS)
- Technologies (TECH)
- Core Network (CORE)
- Access Network (RESA)
- International laboratories (ILAB)

Chacune de ces unités est divisée en plusieurs laboratoires, eux-mêmes divisés en unités de recherche. Le but de ce partitionnement est de créer des domaines de compétences très spécifiques. Le stage s'est déroulé au sein de MAPS-STT-NDS (Security and Trusted Transactions - Network and Devices Security).

1.1.2 Middleware and Advanced Platforms Center (MAPS)

Le laboratoire MAPS est responsable de différents aspects :

- Développer des composants middleware opérateur intégrés, des plateformes de services et des terminaux
- Définir un environnement pour l'interopérabilité des plateformes de service.
- Définir une architecture technique adaptable et des outils pour créer des services associés pour diminuer le coût des services et le temps de développement.
- Déployer des services de communication, de production et de contenus sur tous les terminaux et contribuer à leurs déploiements. Il joue un rôle majeur dans tous les processus d'innovation associés aux objectifs de mettre le client au coeur du monde des communications. Un des principaux challenges pour un opérateur de services est le contrôle des plateformes et des terminaux : en fournissant des API de gestion pour les communications de tous les jours, l'évolution des réseaux vers le tout IP a permis le développement de nouveaux services sur l'Internet, pour la maison ou pour l'entreprise. Ces services

sont maintenant basés sur des middlewares fournissant une vue globale des terminaux jusqu'aux serveurs de contenus, en passant par les routeurs. Le développement des technologies a permis d'uniformiser les infrastructures et les logiciels des plateformes, facilitant l'intégration et l'interopérabilité. Ceux-ci ont encouragé la multiplication des terminaux, amélioré les capacités multimédia et augmenté la bande passante des réseaux fixes et mobiles. Les lecteurs externes peuvent maintenant fournir des services à haute valeur ajoutée.

- Dans ce contexte, France Télécom veut contrôler les plateformes de services, les routeurs et gérer leur complexité, améliorer le temps de développement des services, créer des services qui placent le client au coeur de leur monde, développer une synergie et une convergence des services, préserver la position de l'opérateur dans la chaîne des valeurs, simplifier l'offre aux clients, garantir la qualité du service et la sécurité et réduire les coûts d'opération.

La division Recherche & Développement MAPS, emploie plus de 600 personnes sur 5 sites.

1.2 Problématique générale

Le stage s'inscrit dans la problématique de l'utilisation d'un nouveau type de matériel de sécurité pour IKEv2/IPsec, les adresses générées cryptographiquement (CGA).

Ces adresses ont été mises en place dans le contexte de Secure Neighbour Discovery (SEND). Elles lient très fortement l'adresse IPv6 à son utilisateur, rendant impossible l'usurpation d'identité.

La mise en place d'IPsec est difficile et cette lourdeur freine son déploiement dans des situations simples. Afin de remédier à cela, une solution a été imaginée, combinant IKEv2 aux CGA. Cette solution permet d'avoir un système plus souple, robuste et facile à mettre en place comme décrit plus tard dans ce document.

1.3 État de l'art

Des implémentations de IKEv2 et de CGA sont disponibles sur Internet et au cœur du laboratoire. Cependant il faut prendre en compte les différents designs pour qu'ils interopèrent efficacement et simplement.

IKEv2 étant un protocole jeune, peu d'implémentations sont connues. On peut noter les quatres plus actives :

- OpenIKEv2[1]
- StrongSWAN[2]
- IKEv2[3]
- Racoon2[4]

Divers travaux d'anciens stagiaires ont été basés sur StrongSWAN pour son style de programmation et ces performances. De plus, Racoon2 et IKEv2 n'ont pas de communautés très actives. OpenIKEv2 fournit quand à lui une API plutôt simple et qui ne permet pas d'avoir des fonctions avancées comme la gestion d'OCSP ou EAP-TTLS.

Après analyse des différentes solutions, le logiciel StrongSWAN a été retenu pour IPsec et la librairie DoCoMo USA Labs pour les CGA.

1.4 Objectif du stage

Ce stage orienté recherche sera consacré à l'étude des différents aspects d'utilisation des CGA avec IKE, avec une mise en pratique par l'élaboration d'un démonstrateur. Enfin, une solution se basant sur CGA+DNSSEC sera évaluée afin de rendre l'utilisation encore plus simple sans pour autant perdre en robustesse.

Le stage s'est déroulé en plusieurs étapes :

- Approfondissement des connaissances sur IPsec/IKEv2
- Prise en main des CGA
- Etude théorique des atouts et des désavantages de la solution
- Implémentation d'un démonstrateur
- Participation à la rédaction d'un livrable dans le cadre du projet ANR MobiSEND[5]
- Etude théorique et pratique de la solution avec DNSSEC

Bien sûr, ces étapes reflètent le travail relatif au sujet de stage initial. D'autres missions ont été réalisées de façon spontanée mais cela faisait aussi partie du stage : s'intégrer et rester curieux vis-à-vis des autres sujets de l'équipe. C'est d'ailleurs cette ambiance et cet environnement de travail qui m'ont donnés l'envie et la possibilité de continuer en thèse avec cette équipe.

[1] http ://sourceforge.net/projects/openikev2
[2] http ://www.strongswan.org/
[3] http ://sourceforge.net/projects/ikev2
[4] http ://www.racoon2.wide.ad.jp/
[5] http ://www.mobisend.org/

1.5 Remerciements

Je tiens à remercier Jean-Michel COMBES, mon maître de stage, et Anas ABOU EL KALAM pour leurs conseils avisés et leur aide précieuse durant les six mois. Merci aussi à toutes les personnes du laboratoire et aux professeurs de l'école pour leur soutien et leur humour.

Chapitre 2

IPv6

2.1 Présentation générale

IPv6, ou Internet Protocol Version 6 [DH98], est l'aboutissement des travaux menés au sein de l'IETF à la fin des années 1990 pour succéder à IPv4 dont la capacité d'adressage est insuffisante.

Le format de l'en-tête IPv6 est décrit sur la figure 2.1.

Version codé sur 4 bits, il représente le numéro de version du Protocole Internet. Sa valeur est donc égale à 6 (0110 base 2). Ce champ est identique à la pile IPV4, il sert justement à identifier le protocole de couche 3 du modèle OSI. La liste des différent codes est donné figure 2.2.

Classe (Traffic Class) codé sur 8 bits, il définit la priorité du datagramme afin que des noeuds origines et des routeurs transmetteurs puissent identifier et distinguer la classe ou la priorité du paquet IPv6 en question.

Label (Flow Label) codé sur 20 bits, il peut être utilisé par une source pour nommer des séquences de paquets pour lesquels un traitement spécial de la part des routeurs IPv6 est demandé. Ce traitement spécial pourrait être une qualité de service différente du service par défaut ou un service "temps réel".

Longueur (Payload Length) codé sur 16 bits. Le champ "Longueur" de l'en-tête IPv4 indique la longueur des données incluant l'en-tête IPv4. Contrairement à cela, le champ "Longueur" de l'en-tête IPv6 indique le nombre d'octets des données qui suivent cet en-tête IPv6. Il faut noter que les options de l'en-tête IPv6 sont considérées comme de la donnée et font donc partie du calcul du champ "Longueur".

En-tête suivante (Next Header) codé sur 8 bits, il identifie le type de la Data ou de l'option qui se trouve derrière l'en-tête IPv6. Les valeurs employées sont identiques au champ "Protocole" de l'en-tête IPv4.

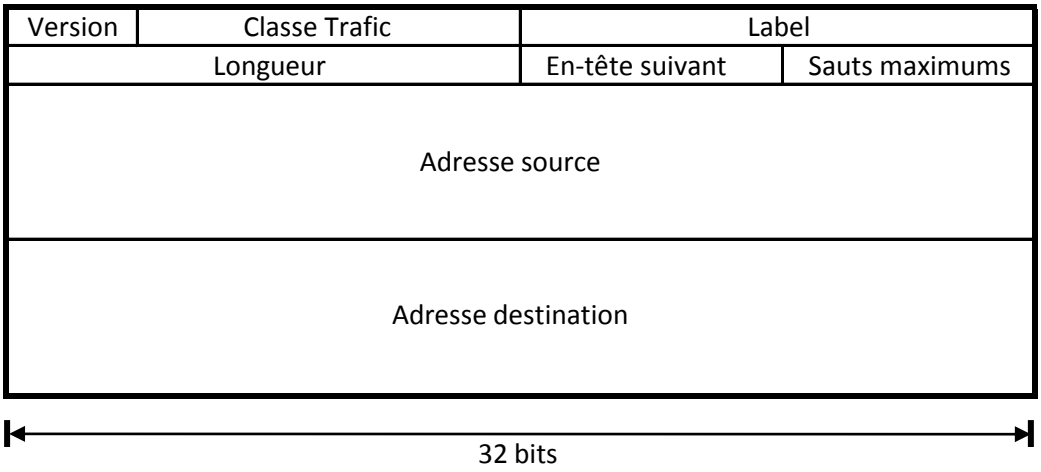

FIG. 2.1 – Format IPv6

Decimal	Mot clef	Version
0-1		Réservé
2-3		Non assigné
4	IP	Internet Protocol
5	ST	ST Datagram Mode
6	IPv6	Internet Protocol version 6
7	TP/IX	TP/IX: The Next Internet
8	PIP	The P Internet Protocol
9	TUBA	TUBA
10-14		Non assigné
15		Réservé

FIG. 2.2 – Liste des versions IP

Saut maximum (Hop Limit) codé sur 8 bits, il indique le nombre de routeurs maximum que le datagramme pourrait traverser. Identiquement au champ "TTL" de l'en-tête IPv4, il est décrémenté de 1 par chaque noeud que le paquet traverse.

Adresse source (Source Address) codé sur 128 bits, il représente l'adresse IP de l'émetteur.

Adresse destination (Destination Address) codé sur 128 bits, il représente l'adresse IP du destinataire.

Chapitre 3

IPsec

IPsec (Internet Protocol Security) [KS05] est une suite protocolaire visant à sécuriser les communications IP via de l'authentification et/ou de chiffrement. Il existe plusieurs types de services permettant de répondre de façon adaptée aux besoins des entreprises, des nomades et des particuliers. Son utilisation la plus connue consiste à encapsuler de l'IP pour créer des tunnels sur un réseau quelconque. Ainsi, il facilite la mise en place de réseaux privés virtuels (aussi appelé Virtual Private Network (VPN).

3.1 Présentation générale

IPsec va gérer un certain nombre de paramètres définis dans des bases de données de politiques de sécurité (Security Policy Database, SPD).

Une association de sécurité IPsec (Security Association, SA) est une structure de données servant à stocker l'ensemble des paramètres de sécurité associés à une communication. L'ensemble des SA est stocké dans la base de données des associations de sécurité (Security Asssociation Database, SAD). Elle est consultée pour définir le comportement à adopter pour les paquets entrants et sortants. Comme une SA est unidirectionnelle, il faut deux SA pour protéger les deux sens d'une communication. Les services de sécurité définis par la SA sont fournis par l'utilisation des protocoles Authentication Header (AH) ou Encapsulating Security Payload (ESP) qui seront expliqués plus tard dans ce document. De plus, une base de données d'autorisation des pairs (Peer Authorization Database, PAD) permet de lier chaque identité à des matériels de sécurité.

Plus précisément, le rôle d'une SA est de spécifier, pour chaque adresse IP avec laquelle IPsec peut communiquer, les informations suivantes :

- le SPI (Security Parameter Index) : l'identifiant de la SA choisi par le récepteur.

- le numéro de séquence : Protége des attaques par rejeu.
- une fenêtre d'anti-rejeu : Un ensemble de bits utilisé pour déterminer si le paquet entrant est un rejeu.
- le dépassement de séquence : Un drapeau signalant que le champ numéro de séquence est dépassé.
- les paramètres d'authentification : Spécifient les algorithmes et les clefs.
- les paramètres de chiffrement : Spécifient les algorithmes et les clefs relatifs au chiffrement.
- la durée de vie de l'association : Intervalle de temps après lequel une SA doit être remplacée ou terminée.
- le mode du protocole IPsec : Tunnel ou transport.
- IP source et destination du tunnel : Les adresses utilisées quand le mode tunnel d'IPsec est utilisé
- la valeur du Path MTU : L'ensemble des path MTU et des variables relatives à l'âge.
- Champ QoS : Informations relatives à l'utilisation avec DiffServ.

Chaque association est identifiée de manière unique à l'aide du SPI, mais certaines implémentations utilisent un triplet composé de :

- l'adresse de destination des paquets
- l'identifiant du protocole de sécurité (AH ou ESP)
- le SPI

Lorsqu'un destinataire reçoit un paquet IPsec, il vérifie qu'une règle de la SPD s'applique au paquet. S'il y en a une, il cherche à quelle SA correspond le paquet reçu. Si cette association de sécurité n'est pas trouvée le paquet est rejeté, sinon il utilise les informations de sécurité pour interpréter le paquet IPSec.

Les services de sécurité sont basés sur des mécanismes cryptographiques. IPsec met en jeu deux protocoles en complément du protocole IP classique : AH et ESP. Ce sont deux types de sécurisations différents même si ESP reprend la plupart des principes d'AH et ajoute notamment des services de confidentialité. Par ailleurs, IPsec offre un service supplémentaire de cryptographie (chiffrement en mode Fast Forward) qui permet de conserver des performances optimales en conservant des paquets de même taille. Néanmoins, ce mode garantie uniquement la confidentialité : L'en-tête IP et la longueur du datagramme restent les mêmes (le champ d'options IP peut être chiffré). En mode transport, AH et ESP fournisse une protection relative au prochain niveau protocolaire ; en mode tunnel, AH et ESP sont appliqués aux paquets IP tunnelisés.

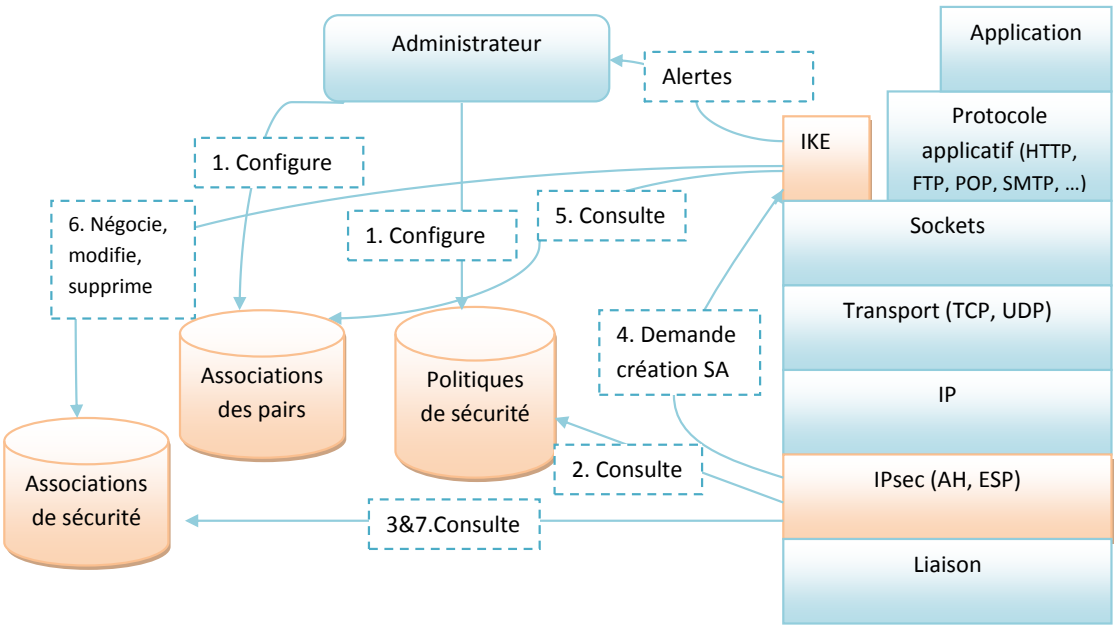

FIG. 3.1 – Fonctionnement d'IPsec

3.2 Les protocoles

De façon plus précise, IPsec utilise deux protocoles pour fournir des services de sécurité, *Authentication Header* et *Encapsulating Security Payload*.

3.2.1 Authentication Header

L'*Authentication Header* [Ken05a] est utilisé pour fournir de l'intégrité, l'authenticité sur l'origine des données et une protection anti-rejeu optionnelle. Une fois l'association de sécurité (SA) est établie, d'autres options pourront être mises en place.

AH permet d'assurer l'authenticité d'autant de champs IP que possible, cependant vu que certains champs ne peuvent être prédits par l'émetteur ils ne seront pas protégés par AH.
Il peut être utilisé seul, combiné avec ESP ou encore de façon imbriquée.

Plusieurs architectures peuvent être mises en place, comme sur les figure 3.2 et 3.3, en particulier.

Mode Transport L'AH est inséré entre l'en-tête IP et la prochaine couche protocolaire en rendant impossible la modification des champs qui ne doivent pas changer.

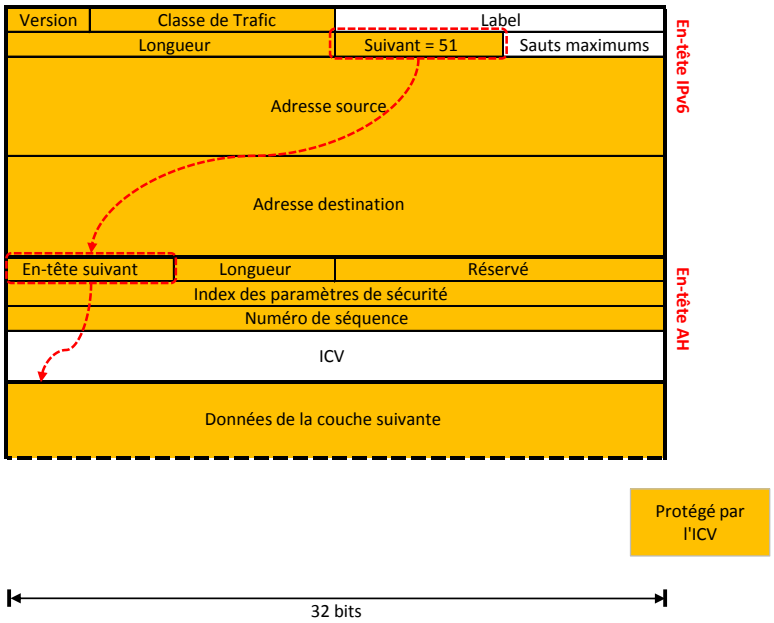

FIG. 3.2 – Modes d'utilisation Transport AH

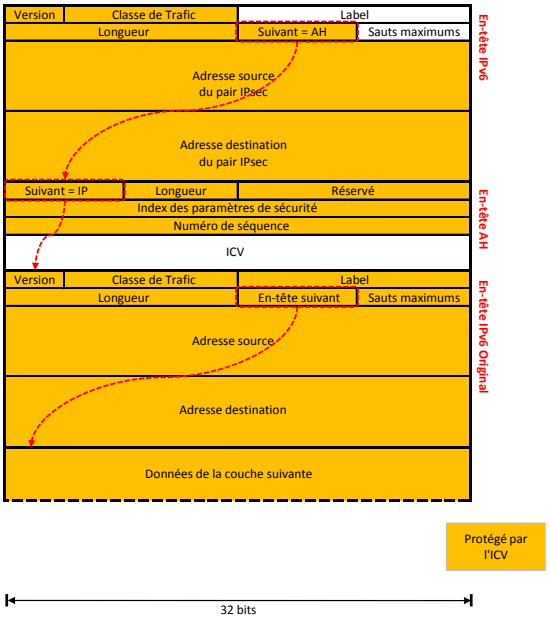

FIG. 3.3 – Modes d'utilisation Tunnel AH

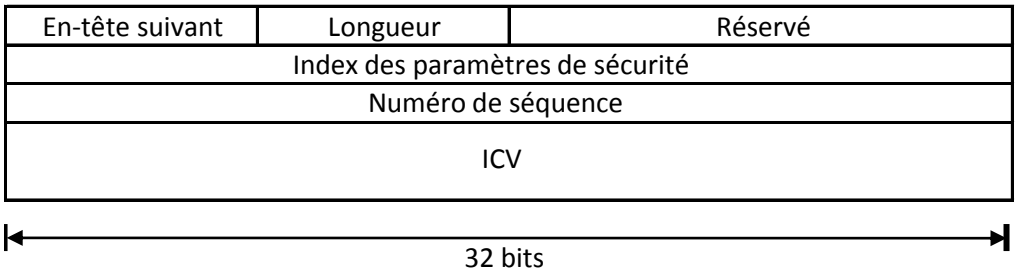

FIG. 3.4 – Format AH

Il faut aussi voir le mode *transport* comme étant une simple sécurisation de la connexion IP, et non pas un VPN.
C'est donc un simple ajout dans le paquet transmis permettant, une fois que l'en-tête AH est validée par le destinataire, de reconstruire le paquet d'origine en enlevant AH et en pointant le champ 'prochain en-tête' vers celui que désignait AH.

Mode Tunnel L'AH authentifie le paquet interne tout entier.

Leurs implémentations varient en fonction du type d'équipement. Un hôte doit pouvoir faire les deux, tandis qu'une passerelle peut ne fournir qu'un seul service de type *Transport*.

Techniquement, le format d'AH est spécifié sur la figure 3.4 où :

Entête suivante (Next Header) codé sur 8 bits, il identifie le type de la Data ou de l'option qui se trouve derrière cette option de l'entête IPv6. Sa valeur doit correspondre à une valeur définie par l'IANA (Internet Assigned Numbers Authority).

Longueur (Payload Len) codé sur 8 bits, il représente la taille de cette Option "AH". L'unité de mesure est les mots de 4 octets avec une valeur minimale de 2.

Réservé (Reserved) codé sur 16 bits, il est prévu pour des besoins futurs, en attendant, sa valeurs doit être positionnée à 0 pour l'émission et il doit être ignorée pour la réception.

SPI (Security Parameters Index) codé sur 32 bits, il permet au destinataire d'identifier l'association de sécurité (SA) avec le datagramme entrant.

Séquence (Sequence Number Field) codé sur 32 bits, il contient le numéro de séquence du SA. Il est incrémenté à chaque datagramme.

ICV (Integrity Check Value-ICV) codé sur N bits, il contient la valeur du résultat d'un procédé cryptographique sur les données. Cela permet d'assurer

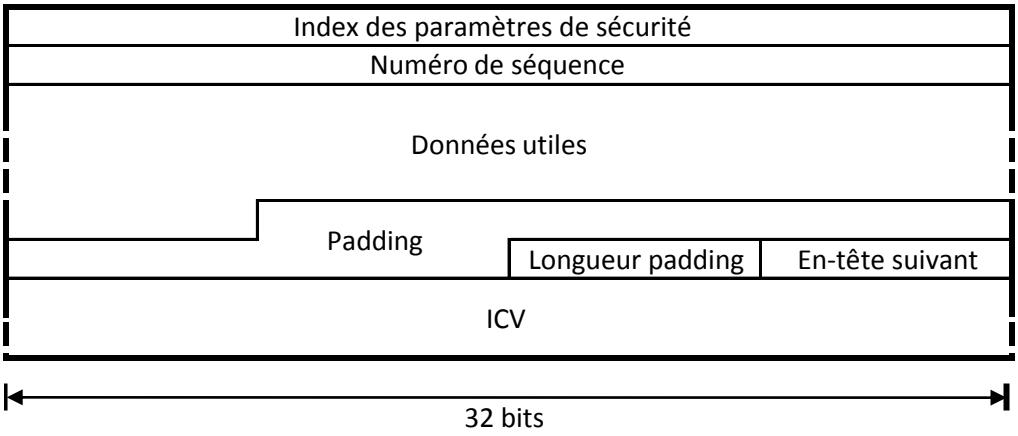

FIG. 3.5 – Format ESP

l'intégrité de celles-ci.

L'ICV est construit à partir du haché calculé par MD5 ou SHA-1 par exemple. Il incorpore un sel permettant aux seules entités le connaissant de pouvoir générer le haché. Ce sel est donc un secret devant être délivré de manière sécurisée, soit par entrée manuelle des utilisateurs physiques, soit par un mécanisme sophistiqué d'échange de clefs (IKEv2) détaillé par la suite.

3.2.2 Encapsulating Security Payload

Le protocole Encapsulating Security Payload [Ken05b] permet d'assurer des services comme la confidentialité, l'authenticité des adresses, l'intégrité, l'anti-rejeu (optionnelle) et une forme de confidentialité des flux de trafics. Tout cela dépend des options choisies au moment de l'établissement de la SA.

Le format utilisé par ESP est décrit sur la figure 3.5.

SPI (Security Parameters Index) Le champ "SPI" est codé sur 32 bits. Il permet au destinataire d'identifier l'association de sécurité (SA) avec le datagramme entrant.

Séquence (Sequence Number) Le champ "Séquence" est codé sur 32 bits. Il contient le numéro de séquence du SA. Il est incrémenté à chaque datagramme.

Données utiles (Payload Data) Le champ "Données utiles" est codé sur N bits. Il contient les données chiffrées.

Remplissage (Padding) Le champ "Remplissage" est codé sur N bits. Il permet de combler l'entête de données non intéressant afin d'obtenir une taille

multiple de 32. La longueur maximum possible est de 255 octets.

Longueur de remplissage (Pad Length) Le champ "Longueur" est codé sur 8 bits. Il représente la taille du champ "Remplissage".

Entête suivant (Next Header) Le champ "Entête suivant" est codé sur 8 bits. Il identifie le type de la Data ou de l'option qui se trouve derrière cette option de l'entête IPv6. Tous les détails de la définition du champ "Entête suivant" sont présents dans le paragraphe 3.5.

ICV (Integrity Check Value) Le champ "ICV" est codé sur N bits. Il contient le haché de l'entête ESP.

Le service d'intégrité prend en compte le SPI, le numéro de séquence, les données et les champs ajoutés par ESP à la fin.

Le service de confidentialité constitue le chiffré à partir des données et du trailer ESP.

Le fait d'*entourer* les données utiles rend la sécurisation plus complexe qu'avec AH où il suffit d'*ajouter* un entête.

Il est possible d'utiliser ESP avec un algorithme de chiffrement à NULL. Cet algorithme combiné avec l'authentification ESP permet de traverser du NAT. Ou inversement, n'utiliser que le chiffrement ESP. Mise à part les cas de test, il ne faut pas que les deux soient simultanément à NULL.

Comme pour AH, ESP peut être utilisé suivant deux modes, décrits sur la figure 3.6 :

Mode Transport Dans le contexte IPv6, ESP est vu comme des données utiles de bout en bout, devant faire apparaître les extensions suivantes :
- Hop-by-Hop
- Routing
- Fragmentation

Pour les implémentations "Bump-in-the-stack" et "Bump-in-the-wire", le support doit être transparent. Il faut donc prévoir des services de ré-assemblage et fragmentation. Les données utiles contiennent uniquement les données relatives à la couche protocolaire suivante.

Mode Tunnel De la même façon qu'AH, seules les adresses des pairs IPsec sont visibles. Les adresses finales de destinations sont chiffrées et visibles seulement après le traitement effectué par les noeuds d'IPsec. Le placement des données relatives à ESP ne change pas du mode *Transport*. Les données utiles contiennent désormais l'ensemble des données relatives à l'entête IP original et à la couche protocolaire suivante. C'est à dire que tout le paquet original est encapsulé.

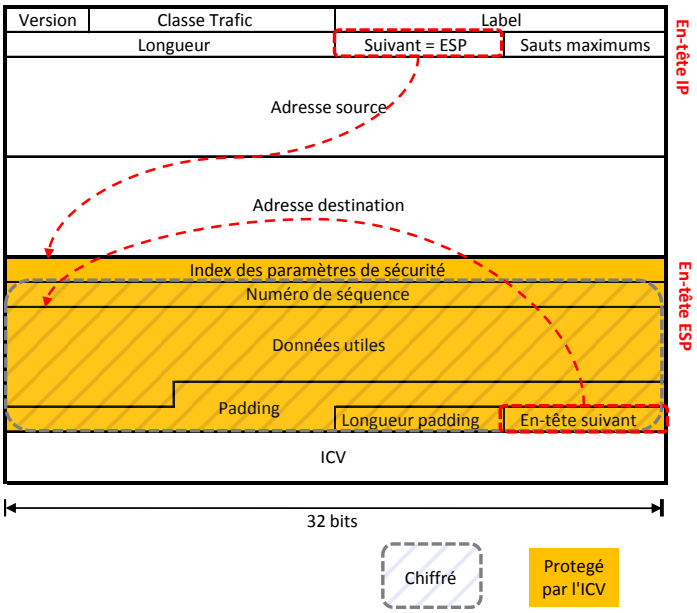

FIG. 3.6 – Modes d'utilisation Transport/Tunnel d'ESP

3.3 Gestion des secrets

3.3.1 Manuelle

L'idée la plus naturelle concernant le partage des clefs entre tous les participants d'une connexion sécurisée est celle de l'échange de clefs physiques. Chaque pair devra alors mettre en place sur son hôte une correspondance entre un pair et la clef qui lui est associée.

Cette technique présente des problèmes majeurs dans une infrastructure sécurisée : comme l'échange du secret dans un milieu non contrôlé, mais aussi le fait que cela ne passe pas à l'échelle.

3.3.2 Internet Key Exchange v2

L'Internet Key Exchange version 2 [Kau05] est une nouvelle version non interopérable avec IKEv1. Une version mise à jour d'IKEv2 est en cours de standardisation, il s'agit d'IKEv2bis[?].

Il s'agit d'un protocole qui permet la négociation des associations de sécurité et de leurs mises à jour, notamment pour l'échange de secrets utilisés comme entrée dans les algorithmes cryptographiques.

Une communication IKE est constituée d'une requête et d'une réponse, aussi

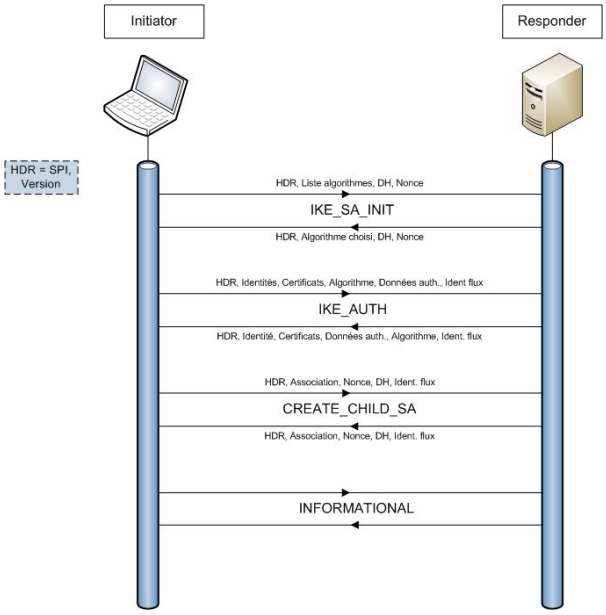

FIG. 3.7 – Echanges IPsec

appelée échange. La figure 3.3.2 décrit ces échanges. Les premiers sont nommés *IKE_SA_INIT* et *IKE_AUTH*, les suivant sont des *CREATE_CHILD* et *INFORMATIONAL*.

L'échange *IKE_SA_INIT* doit être complété avant de pouvoir continuer, tout comme l'échange *IKE_AUTH* doit être fini avant le commencement des suivants.

En résumé, chaque échange négocie :

IKE_SA_INIT les paramètres de sécurité, les nonces, les valeurs pour Diffie-Hellman.

IKE_AUTH les identités, prouvent la connaissance des secrets liés aux identités, mettent en place l'association de sécurité pour le premier CHILD_SA.

CREATE_CHILD crée un CHILD_SA (aussi appelé IPsec SA).

INFORMATIONAL supprime une SA, renvoie des rapports d'erreur, vérifie que le lien existe toujours.

Description de l'échange IKE_SA_INIT

Cet échange va permettre à l'émetteur et au destinataire d'établir des paramètres de confiance pour les échanges à venir. Il est décrit sur la figure 3.3.2

L'émetteur envoie :

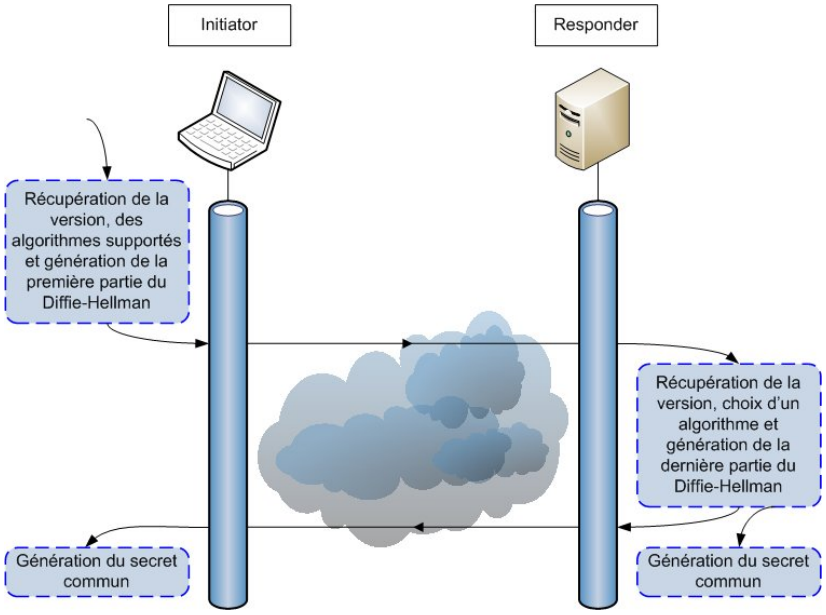

FIG. 3.8 – Echange IKE_SA_INIT

SPI Identifiant l'association de sécurité
Numéro de version Mis à 2
SAi1 Les algorithmes cryptographiques supportés
KEi Les valeurs pour le Diffie-Hellman
Ni Le nonce de l'émetteur

Le destinataire répond :

SPI Identifiant l'association de sécurité
Numéro de version Mis à 2
SAr1 L'algorithme choisi parmi ceux disponibles dans *SAi1*
KEr Complète le Diffie-Hellman
Nr Le nonce du destinataire
CERTREQ Ce champ non obligatoire permet de demander un certificat

A ce niveau là, les deux parties peuvent générer un SKEYSEED dont toutes les clefs utilisées par la suite pour cette SA dérivent. La première pour le chiffrement est communément appelée SK_e pour 'chiffrement', la seconde pour l'intégrité SK_a et celle utilisée pour les fils seront notées SK_d.

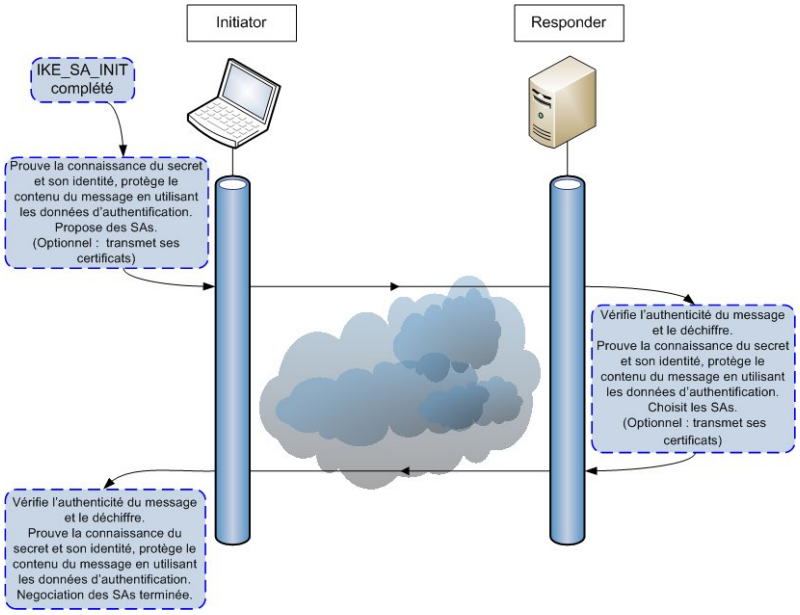

FIG. 3.9 – Echanges IKE_AUTH

Description de l'échange IKE_AUTH

Durant cet échange, chaque pair est authentifié et le premier CHILD_SA est négocié. Il est décrit sur la figure 3.3.2

L'émetteur envoie :

SPI Identifiant l'association de sécurité
Numéro de version Mis à 2
IDi L'identité de l'émetteur
CERT Certificat(s) de l'émetteur (clef publique)
CERTREQ Demande de certificat
IDr L'identité du destinataire
AUTH Contient les données permettant la protection d'intégrité
SAi2 Propositions pour le CHILD_SA
TSi Spécifie la politique de sécurité de l'émetteur pour le CHILD_SA
TSr Spécifie la politique de sécurité du destinataire pour le CHILD_SA.

Les champs en italique sont chiffrés et ont leurs intégrités protégées par SK_e et SK_a. L'émetteur prouve son identité avec les données contenues dans IDi et communique sa connaissance du secret relatif à cette identité.

Le destinataire répond :

SPI Identifiant l'association de sécurité
Numéro de version Mis à 2
IDr L'identité du destinataire
CERT Certificat(s) du destinataire
AUTH Contient les données permettant la protection d'intégrité
SAr2 CHILD_SA choisi
SAi2 Propositions pour le CHILD_SA
TSi Spécifie la politique de sécurité de l'émetteur pour le CHILD_SA
TSr Spécifie la politique de sécurité du destinataire pour le CHILD_SA.

3.3.3 Authentification des IKE_SA

Le secret à partager peut être sous trois formes "traditionnelles" :
- Clef pré-partagées
- Certificats x509
- EAP (Extensible Authentication Protocol)

Cependant, ces méthodes présentent certains problèmes. La publication de la clef pré-partagée rend cette méthode peu sure, tandis que les certificats x509 et l'EAP nécessitent globalement une architecture importante à mettre en oeuvre.

Une utilisation des certificats auto-signés est certainement un bon compromis mais le problème de l'attaque de l'homme du milieu (MITM) reste présent. Une solution alternative peut être utilisée comme nouveau matériel de sécurité, les adresses générées cryptographiquement (Cryptographically Generated Addresses, CGA). La suite décrit comment les CGA peuvent être mises en oeuvre avec IKEv2.

Chapitre 4

Adresses Générées Cryptographiquement

4.1 Présentation générale

L'idée générale des CGA [Aur05] consiste à générer l'identificateur d'interface IPv6 en calculant le haché cryptographique d'une clef publique. La clef privée correspondante peut alors être utilisée pour signer les messages utilisés ensuite par cette adresse.

Pour vérifier l'association entre une clef publique et une adresse, il faut connaitre l'adresse elle-même, la clef publique et la valeur des paramètres auxiliaires.

4.2 Format

Les CGA se basent sur des adresses IPv6, dont les 64 premiers bits représentent le préfixe du sous-réseau et les 64 derniers l'identificateur d'interface (IID).

Un paramètre de sécurité est introduit pour pallier aux attaques de type force brute et à l'évolution de la puissance des machines. Codé sur les 3 bits les plus à gauche de l'IID, c'est un entier non-signé ayant pour valeur :

```
Sec = (IID & 0xe000000000000000) >> 61
```

Une CGA est associée à un ensemble de paramètres constitués d'une clef publique et de paramètres auxiliaires.

Deux hachés seront alors générés en utilisant SHA-1,

Hash1 représente l'IID en ignorant les bits 0, 1, 2, 6 et 7.
Hash2 ces 16*Sec bits les plus à gauche doivent être à zéro.

Leur calcul devra vérifier certaines conditions. En définissant

Mask1 (64 bits) `= 0x1cfffffffffffff`
Mask2 (128 bits) =
`0x0; while(Sec--) Mask2&=(0xffff<<(7-Sec/16))`

Il faudra que :

```
Hash1 & Mask1 == IID & Mask1
Hash2 & Mask2 == 0x0
```

4.3 Paramètres CGA

Les paramètres associés à chaque CGA ont le format défini sur la figure 4.1.

Modifier Ce champ sur 128 bits est utilisé lors de la génération pour ajouter un facteur aléatoire au niveau de l'adresse.

Préfixe de sous-réseaux (Subnet Prefix) utilisé par la CGA.

Collisions (Collision Count) détectées par le DAD (Duplicate Address Detection) [TNJ07]. Ne peuvent dépasser 2.

Clef Publique (Public Key) provient de l'émetteur, elle est de taille variable et formatée dans le standard DER ASN.1 de type SubjectPublicKeyInfo [HPFS02]. L'algorithme utilisé est RSA[1]

Champs d'extension (Extension Fields) permet d'ajouter des champs pour les versions futures.

4.4 Algorithmes

4.4.1 Génération

Trois paramètres sont nécessaires à la génération :

1. Préfixe de sous réseau sur 64 bits
2. Clef publique encodée au format DER ASN.1 du type subjectPublicKeyInfo
3. Paramètre de sécurité entre 0 et 7

[1] Algorithme créé par Rivest, Shamir et Adleman

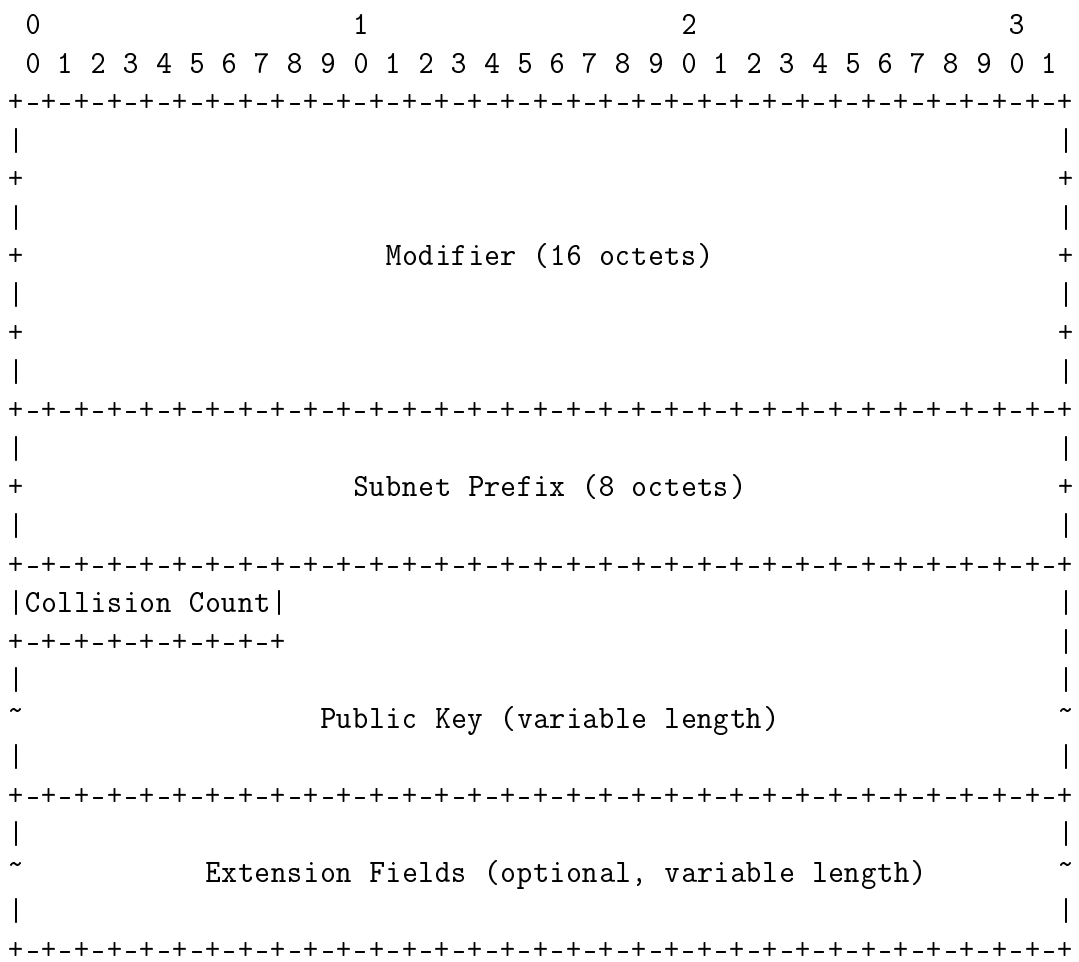

FIG. 4.1 – Format des paramètres CGA

La spécification CGA [Aur05] définit l'algorithme suivant :

1. Set the modifier to a random or pseudo-random 128-bit value.
2. Concatenate from left to right the modifier, 9 zero octets, the encoded public key, and any optional extension fields. Execute the SHA-1 algorithm on the concatenation. Take the 112 leftmost bits of the SHA-1 hash value. The result is Hash2.
3. Compare the 16*Sec leftmost bits of Hash2 with zero. If they are all zero (or if Sec=0), continue with step 4. Otherwise, increment the modifier by one and go back to step 2.
4. Set the 8-bit collision count to zero.
5. Concatenate from left to right the final modifier value, the subnet prefix, the collision count, the encoded public key, and any optional extension fields. Execute the SHA-1 algorithm on the concatenation. Take the 64 leftmost bits of the SHA-1 hash value. The result is Hash1.
6. Form an interface identifier from Hash1 by writing the value of Sec into the three leftmost bits and by setting bits 6 and 7 (i.e., the "u" and "g" bits) to zero.
7. Concatenate the 64-bit subnet prefix and the 64-bit interface identifier to form a 128-bit IPv6 address with the subnet prefix to the left and interface identifier to the right, as in a standard IPv6 address [RFC3513].
8. Perform duplicate address detection if required, as per [RFC3971]. If an address collision is detected, increment the collision count by one and go back to step 5. However, after three collisions, stop and report the error.
9. Form the CGA Parameters data structure by concatenating from left to right the final modifier value, the subnet prefix, the final collision count value, the encoded public key, and any optional extension fields.

Les sorties sont donc une adresse IPv6 et une structure de paramètres CGA.

Le modificateur doit être choisi de façon aléatoire pour rendre les adresses générées avec la même clef sans lien apparent.

Les étapes 2-3 rendent l'attaque par force brute complexe, en effet il faudrait $O(2^{(}16 * Sec))$ itérations ce qui est impossible avec les architectures actuelles.

Certaines astuces permettent de rendre la génération plus rapide, en prenant le même modificateur quand seul le préfixe change par exemple. Cela permet d'économiser la plus lourde étape de calcul.

Un algorithme développé en C est trouvable en appendice A.

4.4.2 Vérification

Deux paramètres sont nécessaires à la génération :

1. Une adresse IPv6
2. Une structure de paramètre CGA

La spécification [Aur05] définit l'algorithme suivant :

```
1. Check that the collision count in the CGA Parameters data
   structure is 0, 1, or 2.  The CGA verification fails if the
   collision count is out of the valid range.
2. Check that the subnet prefix in the CGA Parameters data
   structure is equal to the subnet prefix (i.e., the leftmost
   64 bits) of the address.  The CGA verification fails if the
   prefix values differ.
3. Execute the SHA-1 algorithm on the CGA Parameters data
   structure. Take the 64 leftmost bits of the SHA-1 hash value.
   The result is Hash1.
4. Compare Hash1 with the interface identifier (i.e., the rightmost
   64 bits) of the address.  Differences in the three leftmost bits
   and in bits 6 and 7 (i.e., the "u" and "g" bits) are ignored. If
   the 64-bit values differ (other than in the five ignored bits),
   the CGA verification fails.
5. Read the security parameter Sec from the three leftmost bits of
   the 64-bit interface identifier of the address.  (Sec is an
   unsigned 3-bit integer.)
6. Concatenate from left to right the modifier, 9 zero octets, the
   public key, and any extension fields that follow the public key
   in the CGA Parameters data structure.  Execute the SHA-1
   algorithm on the concatenation.  Take the 112 leftmost bits of
   the SHA-1 hash value.  The result is Hash2.
7. Compare the 16*Sec leftmost bits of Hash2 with zero.  If any one
   of them is not zero, the CGA verification fails.  Otherwise, the
   verification succeeds.  (If Sec=0, the CGA verification never
   fails at this step.)
```

4.5 Problèmes

Il est important de bien comprendre que n'importe qui peut générer des CGA, il est cependant impossible de falsifier une adresse déjà en cours d'utilisation à moins de trouver une collision. Cela ne dépend plus des CGA mais de la résistance à la seconde pré image.

L'utilisation d'une adresse normale ou cryptographique n'est pas clairement distinguée, il est donc possible de diminuer le niveau de sécurité en utilisant l'adresse CGA comme une adresse IPv6 normale.

4.6 Avantages et inconvénients

Comme vu précédemment, le problème des collisions touche tout système utilisant des algorithmes de hachage.

En effet, un attaquant qui génére une paire de clefs valide créera une collision. Il n'est alors plus possible d'assurer la sécurité de la communication.

Les PKI résolvent ce problème grâce aux révocations et permettent à l'autorité de certification (CA) de ne pas valider la clef.

En contrepartie, les CGA sont sans infrastructure ce qui rend leurs déploiements beaucoup plus simples. Le problème de collisions peut être résolu en combinant les CGA avec une résolution de noms sécurisée (DNSSEC).

Chapitre 5

Utilisation des CGA dans IKEv2

L'utilisation des CGA dans IKEv2 est actuellement en cours de normalisation à l'IETF [LMK07] et définit comment exploiter les différents champs de IKEv2 pour les CGA .

5.1 Utilisations des champs IKEv2

5.1.1 Champ ID

Ce champ doit contenir le nom du pair qui va être authentifié dans les données AUTH. Il est naturel d'y mettre l'adresse CGA au format IPv6.

Il serait intéressant d'utiliser les ID_FQDN ici, en complément de DNSSEC. L'implémentation montrera qu'il est difficile de résoudre et vérifier un ID_FQDN contenu dans un champ ID. Nous nous limiterons donc au cas où la résolution se fait avant la création de ce champ, contenant donc uniquement une CGA. Cela implique que le serveur IPsec devra vérifier, via une résolution DNSSEC inverse, que le nom de domaine associé à la CGA est bien défini dans la SAD et la PAD. Faute de temps, nous n'avons pas pu vérifier qu'une résolution DNSSEC inverse était possible. Le serveur IPsec doit donc recevoir comme identité possible les CGA.

5.1.2 Champ Certificate

IKEv2 spécifie que ce champ peut contenir tout type de matériel permettant l'authentification. Il semble donc trivial d'y loger la structure des paramètres CGA.

Il faudra donc choisir un nouveau type de CERT. En attendant la décision de l'IANA, ce type est le 222.

5.1.3 Champ Certificate Request

Ce champ doit contenir le même type CERT que précédemment pour spécifier que l'on souhaite recevoir les paramètres CGA de l'autre pair.

5.1.4 Champ Authentication

Ce champ contient les différents paramètres et permet d'authentifier le propriétaire de l'adresse CGA, il faut y loger une signature digitale du message en utilisant la clef publique contenue dans les données CERT.

5.2 Fonctionnement

Lors du premier échange *IKE_SA_INIT* la seule modification concerne la réponse du destinataire. Il doit transmettre un CERTREQ qui contient le nouveau type 222.

L'échange IKE_AUTH sera construit avec les éléments suivants dans le cas où les deux pairs utilisent les CGA :

SPI
Numéro de version
IDi CGA de l'émetteur
CERT Paramètres CGA
CERTREQ Type CGA (=222)
IDr L'identité du destinataire
AUTH $Signature(Clef publique de CERT)$
SAi2
TSi Politique de sécurité contenant la CGA de l'émetteur
TSr Politique de sécurité contenant la CGA du destinataire

Si un seul des pair utilise les CGA, il sera le seul à envoyer ce type d'IKE_AUTH. L'autre pair peut envoyer n'importe quel type d'IKE_AUTH.

5.3 Atouts et inconvénients

5.3.1 Sans infrastructures

Comme vu précédemment, l'utilisation des CGA ne nécessite pas d'infrastructure car elles sont générées par leurs propriétaires et tous les matériels de sécurité

nécessaires à la vérification sont envoyés. En comparaison, les certificats auto-signés peuvent être générés par n'importe qui. Il est donc impossible de vérifier si l'information contenue dans le certificat est correcte ou pas. L'attaquant peut reproduire exactement le même certificat et changer la valeur de la clé publique par la valeur de sa paire de clé générée. Les CGA sont plus sécurisées vu que l'attaquant doit casser la fonction de hachage et trouver une collision. Ce qui est beaucoup plus difficile que l'usurpation.

De plus, l'ensemble de la vérification est faite par le destinataire du message. Il n'y a donc pas besoin d'une architecture complexe comme pour les certificats X.509 contruits sur une infrasture à clef publique (PKI). Une PKI nécéssite en effet, un certain nombre d'entités comme une autorité de certification (CA) ou une autorité d'enregistrement (RA). Celles-ci introduisent des vecteurs d'attaques potentiels, et une seule d'entre elles peut compromettre la chaîne de vérification toute entière pour au final revenir à des certificats auto-signés.

Même si une CGA permet d'associer de manière forte une adresse et son utilisateur, elle ne peut pas empêcher un utilisateur malveillant de l'utiliser. Ainsi, une adresse compromise par la collision ne peut être révoquée comme c'est le cas dans une infrastructure de clé publique (PKI) par l'autorité de Certification (CA).

5.3.2 Identité

Les CGA, en tant qu'adresses IPv6, ne sont pas facilement mémorisables par les humains (128 encodé en hexadécimal). De plus, le destinataire d'un message ne peut pas savoir directement qui est l'émetteur de la CGA. C'est pourquoi il faut associer les CGA à des noms de domaines (FQDN) enregistrés dans un serveur de noms de domaines (DNS). Cette solution peut cependant introduire des problèmes inhérents au DNS, comme le cache poisoning, et fragiliser le niveau de sécurité assuré par les CGA. Pour pallier à ce problème, les mise à jour DNS et leurs résolutions doivent être sécurisées. Les mise à jours interviennent lorsqu'un utilisateur veut mettre à jour le lien entre sa CGA et son nom domaine dans le serveur DNS. Les résolutions sont utilisées lorsqu'un noeud IPv6 veut connaître la CGA de l'utilisateur associée à un FQDN. Pour sécuriser les échanges de mise à jour, TSIG [Eas06] et SIG(0) [BKN06] peuvent être mis en œuvre. Au niveau des résolutions, la version sécurisée de DNS, DNSSEC [Har06], doit être déployée. Avec de telles protections, un attaquant potentiel doit de nouveau faire face à des problèmes de recherche de collision.

5.3.3 Algorithmes Cryptographiques

Les spécifications des CGA [Aur05] autorisent seulement l'utilisation de deux algorithmes cryptographiques : l'algorithme de clef publique RSA et la fonction de

hachage SHA-1. Au niveau de RSA, cet algorithme est très couteux en terme de ressources et de temps CPU. Il n'est donc pas adapté à des noeuds mobiles ou des capteurs. Au niveau SHA-1, selon les dernièrs résultats de la cryptanalyse, la communauté des experts en sécurité prévoit que cet algorithme soit cassé dans un futur proche et que les collisions soient rapidement trouvées. La communauté travaille déjà sur la prochaine génération d'algorithme SHA-3. Evidemment, il est important de souligner que certains pays peuvent limiter l'utilisation d'algorithmes. En Russie, seul l'algorithme GOST peut être utilisé.

5.3.4 Revocation

Bien que les CGA fournissent une relation forte entre une adresse et son tulisateur, rien n'empêche un attaquant ayant trouvé une collision de l'utiliser : l'adresse est compromise. Sans infrastructure, il est impossible d'utiliser des listes de révocations (CRL) ou un protocole de status des certificats (OCSP) comme pour les certificats X.509. Une CGA ne peut être révoquée. Une solution potentielle est de considérer l'utilisation de DNS (voire DNSSEC directement au vu des problèmes précédents). En effet, si la valeur du champ temps de vie (TTL) de l'enregistrement AAAA du FQDN associée à la CGA est plus courte que le temps nécessaire pour trouver une collision, l'attaque devient limitée. Un attaquant peut cependant essayer de générer des collisions pour un grand nombre de CGA, et donc réduire le temps de calcul, rendant la solution précédente inutile.

Chapitre 6

Implémentation

Les différentes implémentations ont été comparées dans une précédente section. La suite décrira la solution choisie et les modifications apportées.

6.1 StrongSWAN

Il s'agit d'une implémentation de IPsec/IKEv2 récupérée d'un projet à l'abandon nommé FreeSWAN. Il est très utilisé en ce moment et possède une communauté de développeurs très active.

6.1.1 Fonctionnement

Description

IPsec étant assez complexe, StrongSWAN [Ste05] dispose d'une architecture très modulaire. Un démon central appelé starter va initialiser l'environnement et lancer les autres démons décrits dans les différents fichiers de configuration.

Le démon propre à IKEv2 se prénomme `charon`. Il va se greffer à la pile IP pour récupérer les paquets IP et leurs appliquer différentes actions suivant la politique de sécurité établie, à savoir rien, chiffrés ou rejetés. De plus, ce démon va être lancé sous forme de thread où un démon maître va nourrir ces fils avec différentes informations rendant le traitement hautement parallélisé.

Afin d'interagir avec `charon`, un bus de données agissant directement avec l'interface utilisateur est créé. Il est appelé `stroke` et fournit une interface pour chaque démon permettant d'interagir avec la console.

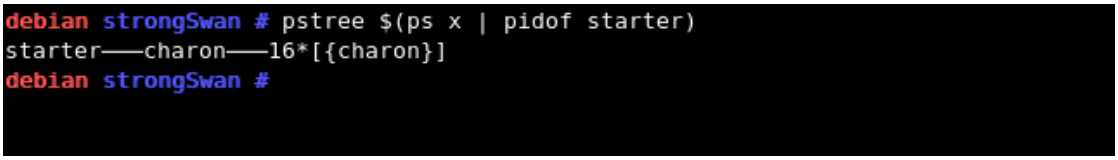

FIG. 6.1 – Dépendances du démon starter

Configuration

Trois fichiers de configurations sont nécessaires au fonctionnement :

ipsec.secrets
: Contient les emplacements absolus ou relatifs des clefs privées.

ipsec.conf
: Contient la liste des options disponibles pour chaque association de sécurité.

strongswan.conf
: Décrit les paramètres que le starter devra prendre en compte lors du lancement des démons.

Utilisation

Une fois la configuration établie et StrongSWAN installé, il ne reste plus qu'à lancer le starter via `ipsec start` (figure 6.2). Celui-ci lance des démons en arrière plan, en attendant que l'utilisateur envoi des commandes via `ipsec`. Cette commande est récupérée par le démon `stroke` prenant une entrée clavier et la transforme en message `stroke\_t` standardisé que tous les démons peuvent interpréter. Une fois que le message est prêt, il est diffusé sur le bus de communication inter démons, où chacun possède une interface d'écoute. Ainsi les démons concernés par le message peuvent effectuer les actions attendues. Ce type d'interaction permet d'avoir un état d'ensemble cohérent grâce à un protocole simple. Par exemple, une des entités devra envoyer `ipsec up <conn-name>` (figure 6.3) pour établir une SA. Une description générale va s'afficher en sortie, et une sortie beaucoup plus détaillée sera écrite dans le fichier `/var/log/daemon.log` par défaut.

6.1.2 Modifications

StrongSWAN peut être agrémenté de plugins, comme par exemple le fait de pouvoir utiliser des certificats x509 qui contiennent uniquement des clefs publiques.

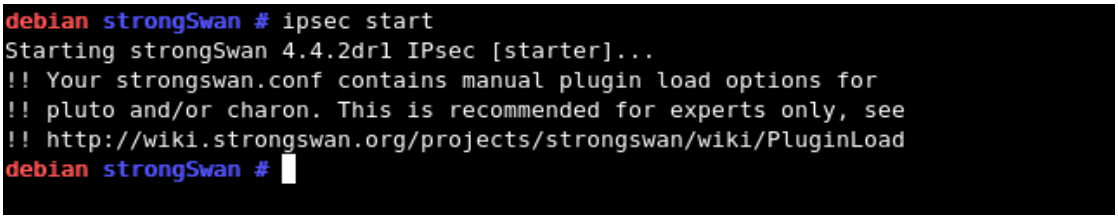

FIG. 6.2 – Démarrage du démon starter

Les fichiers `stroke\_config.c` et `stroke\_config.h` doivent être mis à jour pour prendre en compte de nouveaux types. Ainsi le fichier `ipsec.conf` pourra comprendre :

```
conn v6ss
left=%defaultroute
leftauth=cga
leftid=fec0::241b:73d6:4288:223c
leftcgapm=bige_cga_param.der
right=10.0.0.1
rightid=fec0::2406:7af:6bf6:f143
auto=add
```

Il faut aussi inclure les fichiers de DoCoMo USA Labs dans la chaîne de compilation, en modifiant le `Makefile.am`. La modification devra être prise en compte en lançant une série de commandes à la racine du répertoire StrongSWAN :

```
aclocal4
autoconf
automake
./configure --enable-cga
```

En cas de problème un `autoreconf` peut les résoudre.

Au niveau du code lui même, il a fallu repenser le draft [LMK07] afin de produire quelque chose d'exploitable et facilement maintenable. Pour cela, les paramètres CGA vont être mis sous forme d'un certificat X.509 à l'aide d'un wrapper (enveloppe) comme illustré sur la figure 6.4.

Ce wrapper sera ajouté dynamiquement à l'ensemble des constructeurs de certificats, via un plugin. Ainsi, lorsque le démon `charon` lit le fichier de configuration, il arrive à convertir le fichier .der en certificat valide pour le champ CERT. Pour cela il essaie tous les constructeurs qu'il connait.

D'autres changements interviennent dans les différents fichiers gérant l'ensemble des payloads IKEv2.

```
debian strongswan-cga # ipsec up v6ss
initiating IKE_SA v6ss[1] to fec0::2406:7af:6bf6:f143
generating IKE_SA_INIT request 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) ]
sending packet: from fec0::241b:73d6:4288:223c[500] to fec0::2406:7af:6bf6:f143[
500]
received packet: from fec0::2406:7af:6bf6:f143[500] to fec0::241b:73d6:4288:223c
[500]
parsed IKE_SA_INIT response 0 [ SA KE No N(NATD_S_IP) N(NATD_D_IP) CERTREQ N(MUL
T_AUTH) ]
received cert request for CGA "fec0::41b:73d6:4288:223c"
sending cert request for "fec0::41b:73d6:4288:223c" (as id)
CGA_PLUGIN: cga_auth.build :
CGA_PLUGIN: id=fec0::241b:73d6:4288:223c
CGA_PLUGIN: Found a private key for this id :-)
authentication of 'fec0::241b:73d6:4288:223c' (myself) with RSA signature succes
sful
sending end entity cert "fec0::41b:73d6:4288:223c"
establishing CHILD_SA v6ss
generating IKE_AUTH request 1 [ IDi CERT CERTREQ IDr AUTH SA TSi TSr N(MULT_AUTH
) N(EAP_ONLY) ]
sending packet: from fec0::241b:73d6:4288:223c[500] to fec0::2406:7af:6bf6:f143[
500]
received packet: from fec0::2406:7af:6bf6:f143[500] to fec0::241b:73d6:4288:223c
[500]
parsed IKE_AUTH response 1 [ IDr CERT AUTH SA TSi TSr N(AUTH_LFT) ]
[ike_cert_pre]adding cert to auth<0x80621a8>->
  trying to build cga parameters
  feeding certificate with blob from blob
  Generating CGA
  Converting CGA to string
    string : fec0::406:7af:6bf6:f143
  Feeding certificate subject field
  Feeding certificate key field
Dumping this->ctx => 4 bytes @ 0x80645a8
   0: 79 2F 06 08                                     y/..
Dumping this->ctx->modifier => 16 bytes @ 0x80645e8
   0: EB C3 71 F2 2D CE DA 62 CA DD C9 27 7B 2D 79 FD  ..q.-..b...'{-y.
Dumping this->ctx->prefix => 8 bytes @ 0x80645b0
   0: FE C0 00 00 00 00 00 00                         ........
Dumping this->ctx->key => 162 bytes @ 0x8062f79
   0: 30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01  0..0...*.H......
  16: 05 00 03 81 8D 00 30 81 89 02 81 81 00 CB B8 E9  ......0.........
  32: F8 D8 26 A9 69 61 FF CB 3E CF 55 28 B9 04 6A EE  ..&.ia..>.U(..j.
  48: 82 A5 27 34 22 8A 2C 57 8C C8 7E 75 D0 F8 FC A3  ..'4",W..~u....
  64: 0D F4 73 0D 8E 00 28 3A 6E 0D E8 F7 D7 99 F7 97  ..s...(:n.......
  80: B2 10 17 A9 23 72 05 32 B8 9A 64 B3 27 2B 5C 70  ....#r.2..d.'+\p
  96: C6 4B 8D 6D EE BD AC 7A 85 C8 60 3E CF 2C 59 86  .K.m...z..`>.,Y.
 112: 7A 4B DF 85 DA 55 86 D8 7C 82 B6 BE 75 5A C1 7F  zK...U..|...uZ..
 128: 99 CF 3F F7 52 75 BE EE 52 CD B0 24 77 C7 8A 17  ..?.Ru..R..$w...
 144: FE F2 02 6C F1 88 2D 71 48 63 3A F3 6F 02 03 01  ...l..-qHc:.o...
 160: 00 01                                           ..
Dumping this->ctx->addr => 16 bytes @ 0x80645c0
   0: FE C0 00 00 00 00 00 04 06 07 AF 6B F6 F1 43  ...........k..C
Dumping this->ctx->sec => 1 bytes @ 0x80645d4
   0: 00                                              .
[ike_cert_pre]try_get_cert ok
[ike_auth->process_i] Get_id returned fec0::2406:7af:6bf6:f143
[ike_auth->process_i] other_id is fec0::2406:7af:6bf6:f143
In Process
CGA owned by id fec0::2406:7af:6bf6:f143
authentication of 'fec0::2406:7af:6bf6:f143' with RSA signature successful
IKE_SA v6ss[1] established between fec0::241b:73d6:4288:223c[fec0::241b:73d6:428
8:223c]...fec0::2406:7af:6bf6:f143[fec0::2406:7af:6bf6:f143]
scheduling reauthentication in 10020s
maximum IKE_SA lifetime 10560s
```

FIG. 6.3 – Mise en place d'une association de sécurité

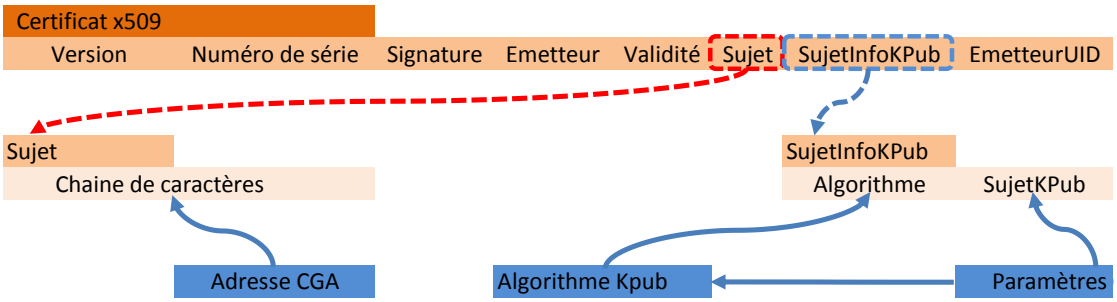

FIG. 6.4 – Transformation d'un certificat X.509 en enveloppe pour CGA

Fichier modifié	Description
libstrongswan/	
-> plugins/cga/	Ajout de la librairie DoCoMo
-> cga_cert.c	Wrapper X.509
-> cga_plugin.c	Ajout du constructeur
-> credentials/certificates/certificate.h	Nouveau type de cert.
libcharon/	
-> plugins/stroke/stroke_config.c	Prend en compte l'auth. cga
-> sa/tasks/ike_cert_pre.c	Importe le certificat dans le CERT.
-> sa/authenticators/	
-> cga_authenticator.c	Définit les actions build et process.
-> cga_authenticator.h	Offre de nouveaux create().
-> authenticator.c	Nouveau type d'auth.
-> encoding/payloads/	
-> cert_payload.c	Prend en charge le nouveau codage.
-> cert_payload.h	Nouveau type de codage des certificats.
-> certreq_payload.c	Demande le certificat pour CGA

Le fichier `certificate.h` (figure 6.5) permet de donner une valeur au nouveau type de certificat. Celle-ci permettra d'identifier le certificat en tant qu'enveloppe CGA.

Le fichier `stroke_config` lit les fichiers de configuration et configure l'ensemble des paramètres nécessaires aux démons. Ce fichier doit être modifié pour que le type d'authentification "cga" soit reconnu comme étant valide.

Afin de mettre en place l'ensemble des payloads IKEv2, StrongSWAN génère un ensemble de tâches dans une file d'attente. Il faut donc intervenir pendant ce processus pour que le certificat de type CERT_CGA soit reconnu et ajouté. La tâche prenant en charge ce processus est `ike_cert_pre` (*pre* comme preprocessing)

```
/**
 * Kind of a certificate_t
 */
enum certificate_type_t {
    /** just any certificate */
    CERT_ANY,
    /** X.509 certificate */
    CERT_X509,
    ...
    /** Cheated certificate to accept CGA parameters */
    CERT_CGA,
};
```

FIG. 6.5 – Modification du fichier certificate.h

sur la figure 6.6. Les modifications sont décrites sur la figure 6.7.

Pour ce qui est de l'authentification à proprement parler, elle est gérée dans une instance de type `authenticator\_t`. Tout type d'authentification doit se greffer au fichier `authenticator.c/h` pour y ajouter son type, son constructeur et son processeur. Notre instance se nomme cga_authenticator et permet de construire le payload AUTH pour les CGA en implémentant les commandes `build\_i/r` et `process\_i/r` (voir figure 6.8). Celles-ci énumèreront directement sur les payloads de type CERT et CERTREQ modifiés pour construire et consulter leurs contenus. En effet, CERTREQ doit retourner le nouveau type CERT_CGA au niveau de l'encodage ; CERT doit retourner le certificat au format CERT_CGA si l'encodage est défini comme étant ENC_CGA.

6.2 Wireshark

La vérification du bon déroulement de IKEv2 peut se faire via un analyseur de protocoles, Wireshark[1]. Ce dernier va permettre de déchiffrer à la volée l'ensemble des paquets transitant sur les interfaces réseau.

6.2.1 Fonctionnement

Les divers protocoles sont reconnus en fonction du port qu'ils utilisent. IKEv2 utilise les ports 500 et 4500, Wireshark essaye donc de décoder ces paquets en appelant la fonction définie dans le fichier "isakmp". Le nom peut être trompeur, mais les développeurs ont simplement créé un plugin qui supporte les deux versions de IKE.

[1] http ://www.wireshark.org

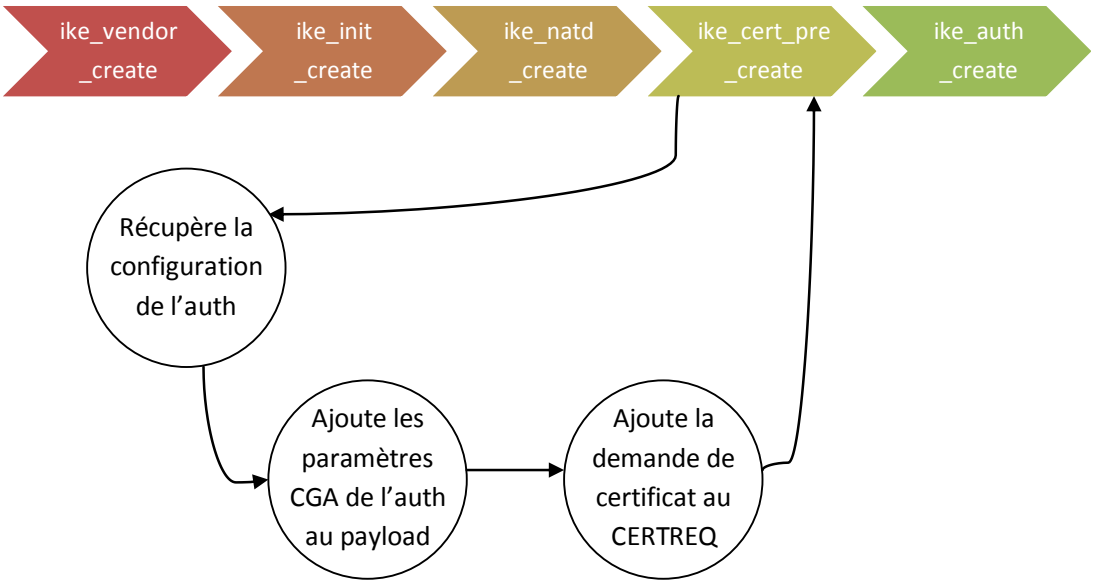

FIG. 6.6 – Description de la modification dans la file de tâche

```
case CERT_CGA:
    enumerator = certreq->create_keyid_enumerator(certreq);
    while (enumerator->enumerate(enumerator, &keyid))
    {
        identification_t *id;
        certificate_t *cert;

        id = identification_create_from_encoding(ID_IPV6_ADDR, keyid);
        cert = lib->credmgr->get_cert(lib->credmgr,
                                                    CERT_CGA, KEY_ANY, id, TRUE);

        if (cert)
        {
            DBG1(DBG_IKE, "received cert request for CGA \"%Y\"",
                     cert->get_subject(cert));
            auth->add(auth, AUTH_RULE_SUBJECT_CERT, cert);
        }
        else
        {
            DBG1(DBG_IKE, "received cert request for CGA unknown "
                          "with keyid %Y", id);
        }
        id->destroy(id);
    }
    enumerator->destroy(enumerator);
    break;
```

FIG. 6.7 – Modifications du fichier ike_cert_pre.c

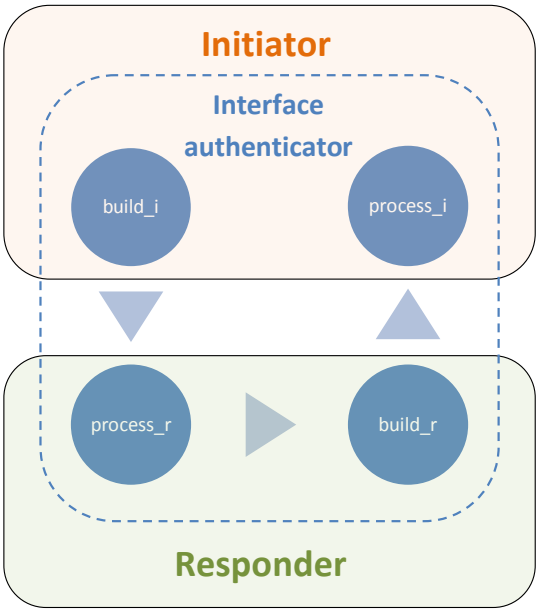

FIG. 6.8 – Fonctionnement d'un authenticator

6.2.2 Modifications

La version de Wireshark devant simplement prouver la cohérence entre l'implémentation et la spécification IKE avec CGA [?], une fonction de déchiffrement des certificats vers la librairie de DoCoMo USA Labs au travers d'un wrapper a été rédigée. Il a aussi fallu ajouter le nouveau type d'encodage pour le certificat, et un désassembleur hexadécimal pour interagir avec l'interface graphique. Cela à permis d'avoir un programme fonctionnel rapidement.

6.2.3 Résultats

Le paramétrage du plugin se fait après que la connexion IPsec soit établie et que l'on peut connaître le SPI, et les clefs d'authentification et de chiffrement (figure 6.9 et figure 6.10).

Après configuration, la dissection des paquets nous a permis de valider que les champs contenaient exactement ce qui était attendu figure 6.11.

6.3 BIND

Le logiciel BIND [HV10] est le serveur DNS le plus utilisé dans le monde. Il supporte DNSSEC et peut servir de serveur cache.

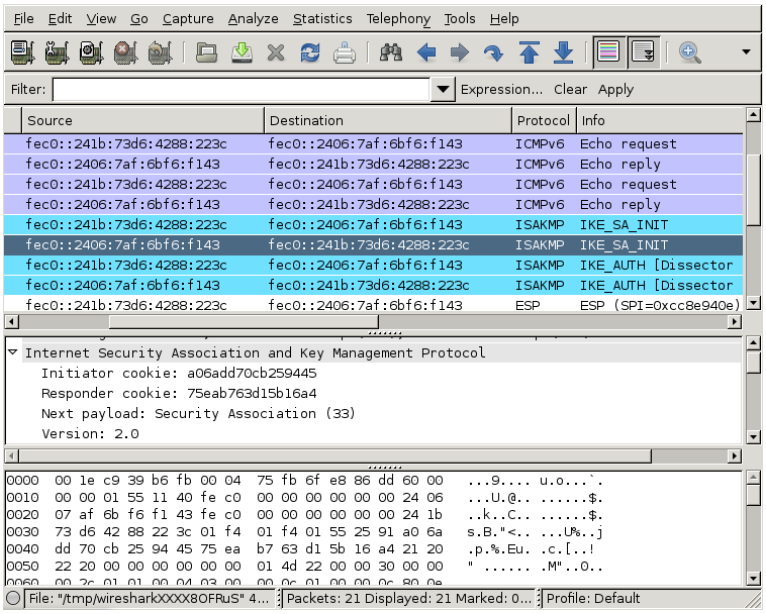

FIG. 6.9 – Récupération des cookies IKEv2 (SPI)

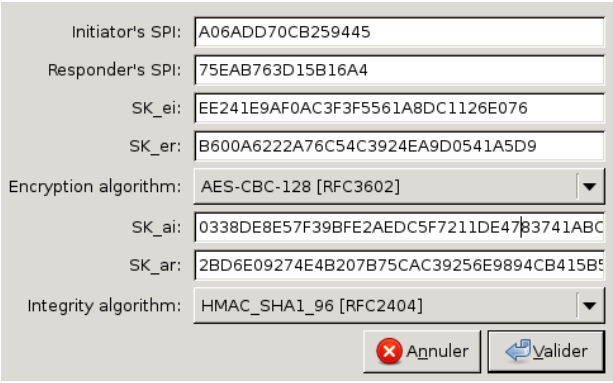

FIG. 6.10 – Passage des clefs de chiffrement et d'authentification

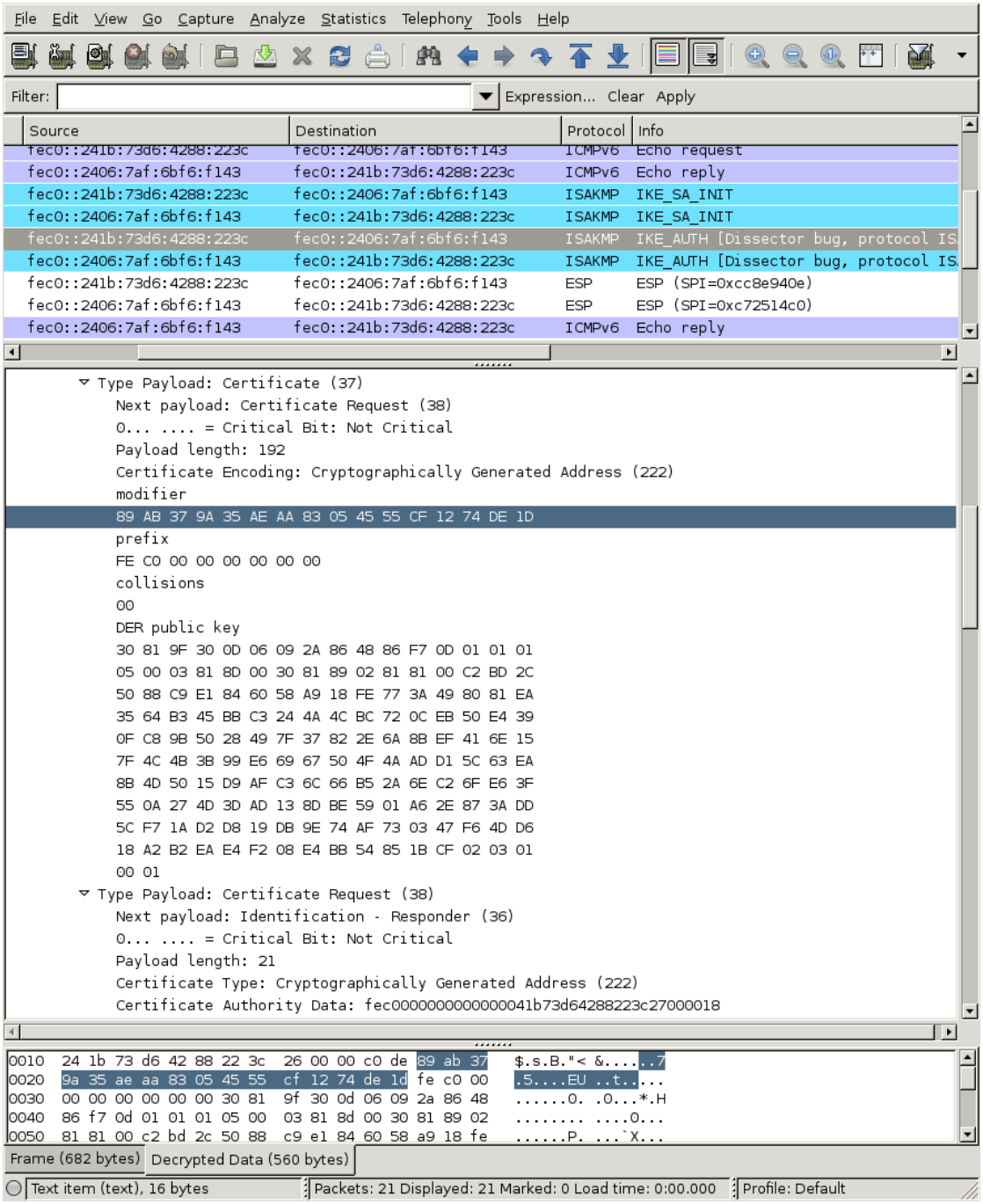

FIG. 6.11 – Déchiffrement des payloads contenant les paramètres CGA

```
$TTL 86400
y0da.com.       IN      SOA     ns1.y0da.com. admin.y0da.com. (
                                                        2006081401
                                                        28800
                                                        3600
                                                        604800
                                                        38400
 )

y0da.com.       IN      NS              ns1.y0da.com.

smalle.y0da.com.          IN      AAAA    fec0::2406:7af:6bf6:f143
bige.y0da.com.            IN      AAAA    fec0::241b:73d6:4288:223c
ns1.y0da.com.             IN      AAAA    fec0::3
```

FIG. 6.12 – Fichier de zone BIND pour y0da.com

6.3.1 Fonctionnement

Le principe du DNSSEC est d'utiliser un serveur cache de confiance qui résout des requêtes DNSSEC pour trouver la CGA attachée au FQDN.

Afin de rester dans une politique de confiance efficace, le serveur cache doit être placé directement sur la machine cliente. Ainsi le DNS local pointe vers la machine hôte locale.

```
/etc/resolv.conf
nameserver ::1
```

La résolution se fait donc via deux modules indépendants des CGA dans IKEv2. StrongSWAN demande au système l'IP qui se cache derrière et BIND s'occupe de résoudre le FQDN demandé par l'appel système.

BIND commence par demander aux root-servers qui gèrent la première sous zone de manière sécurisée en vérifiant les signatures, puis déroule la chaîne jusqu'au domaine attendu. Il construit donc une chaîne de confiance, et renvoie au système soit l'IP, soit un code d'erreur. StrongSWAN récupère ce code d'erreur et arrête la mise en place de la connexion.

6.3.2 Configuration

La configuration de BIND est assez simple, mais non triviale.

Il faut créer une zone, par exemple y0da.com. Le serveur DNS sera ns1.y0da.com et les deux entités IPsec respectivement smalle.y0da.com et bige.smalle.com. Un exemple est donné figure 6.12.

La prochaine étape consiste à signer cette zone avec des paires de clefs générées aléatoirement. BIND fournit des outils comme dnssec-keygen et dnssec-signzone. Les clef générées sont respectivement nommées Zone Signing Key (ZSK) et Key

```
; This is a zone-signing key, keyid 61167, for y0da.com.
; Created: Fri Jun 18 16:45:59 2010
; Publish: Fri Jun 18 16:45:59 2010
; Activate: Fri Jun 18 16:45:59 2010
y0da.com. IN DNSKEY 256 3 5 AwEAAbdGNgwYHJHdUG/bX/xzkgIZrshpaiHpfKdR8ge
6GiqPH5eFbdth 3pF09M2ikKQ0FLbI3JuyPCqLnvijQlggkRHO2bdP9/5cUcIbcQhdZQXY
wf6wUQiklZ/oClE644L8no7PflomCqwvClBomH9YR7vWthoSfEA+Zo9i h+iZFdCz
; This is a key-signing key, keyid 62023, for y0da.com.
; Created: Fri Jun 18 16:46:25 2010
; Publish: Fri Jun 18 16:46:25 2010
; Activate: Fri Jun 18 16:46:25 2010
y0da.com. IN DNSKEY 257 3 5 AwEAAZ8TyjVsmY875MYJwDewqrXcVfPhA2F9MunS9Vi
rd7ybpv/Z4H9G LQuBB7sgR7KsXLsLjIPf4Lg3W7T/FEjj3PUtPG/h8ywlvMMocCcHLeSd
vimmBEIg43sYvGzRLUzE2Gk1U8cLEAVnoitG77yNvl+NF0RM2IQt5v35 DNqS+uG9
```

FIG. 6.13 – Extension fichier de zone BIND signé pour y0da.com

Signing Key (KSK). La ZSK permet de signer les données de la zone, tandis que la KSK signe la ZSK pour fournir un point d'entrée sécurisé sur la zone. Il suffit donc d'ajouter les parties publiques des deux nouvelles clefs dans les champs DNSKEY du fichier de zone pour qu'il puisse être vérifié par le serveur cache. Évidemment, il faudra transmettre les parties publiques des clefs qui devront être considérées comme étant de confiance. Dès qu'une correspondance est trouvée entre une clef publique obtenue avec DNSSEC et une clef publique de confiance, cette clef publique est considérée comme "trust anchor" de la chaîne de confiance.

Une fois que les clefs sont générées et ajoutées au fichier de zone, il faut le recharger pour que BIND le prenne en compte. Le serveur BIND considérera la résolution de nom de domaines sécurisés DNSSEC via l'ajout de deux lignes spécifiques dans le fichier de configuration. La prise en charge du protocole DNSSEC s'ajoute par `dnssec-enable yes`, tandis que la vérification des différents domaines de la chaîne de confiance est définie grâce à la ligne `dnssec-validation yes`.

Le serveur cache local n'aura plus qu'à vérifier que la signature correspond bien à la signature avec la clef qu'il connait.

6.4 Conclusion

Les CGA sont donc faciles à mettre en œuvre si on en croit le draft [LMK07] et la théorie, mais la pratique montre que n'importe quel changement mineur dans la structure interne d'un protocole peut très vite se transformer en réel challenge. Il a fallu trois mois de travail pour mettre en place ces différentes implémentations et configurations et trois machines pour les exécuter. Cependant, quelques erreurs restent présentes.

DNSSEC se lie facilement aux CGA et cela peu importe les implémentations. Seul importe la façon dont le système gère le DNS. Linux s'avére adapté du moment

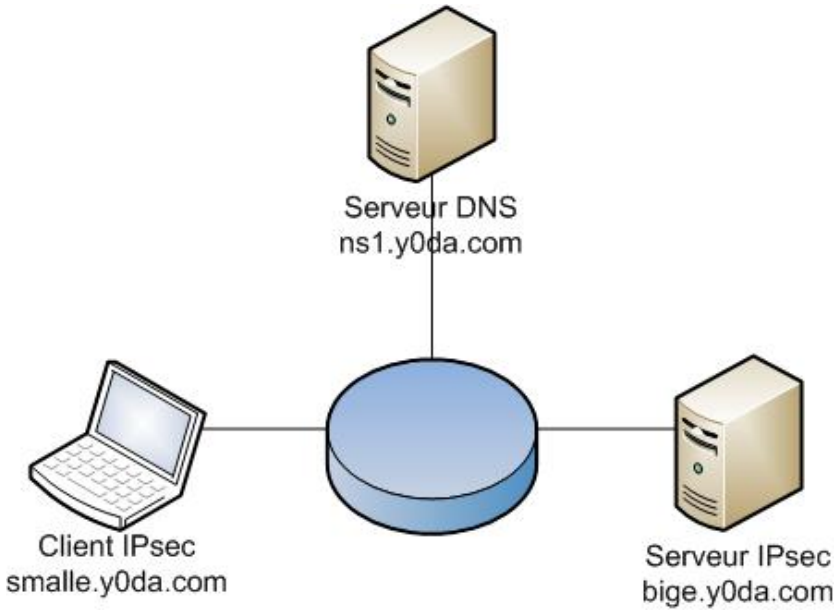

FIG. 6.14 – Schéma du démonstrateur

que son serveur DNS est localisable dans `/etc/resolv.conf`.

Le démonstrateur final fournit donc quelque chose qui se rapproche d'une PKI en étant plus légère (voir figure 6.14). Son fonctionnement est expliqué figure 6.15, il résume de façon simple les différentes étapes d'une communication.

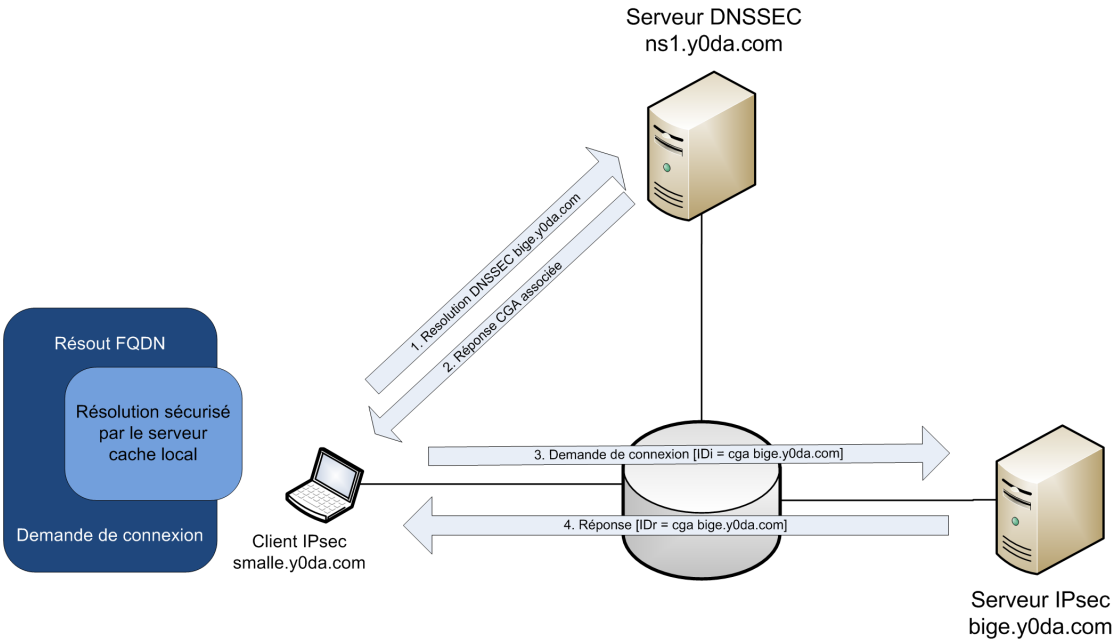

FIG. 6.15 – Étapes clefs d'une connexion utilisant la solution

Chapitre 7

Conclusion

Ce stage m'a permis d'éveiller ma curiosité dans de nombreux domaines. Grâce au soutien de toute l'équipe de chercheurs très compétente, j'ai pu très facilement m'intégrer et me familiariser avec le domaine de la recherche et du développement. L'éventail des compétences est conséquent à Orange Labs, et l'aspect recherche permet de ne pas rester figé sur un seul sujet.

7.1 Bilan technique

Les compétences que j'ai pu développer ont touchés les réseaux, les systèmes, la sécurité et les mobiles. Des sujets dont certaines problématiques ne sont pas évidentes et pourtant très utiles. Ce sont ces discussions et débats intéressants avec toute l'équipe d'Orange Labs qui m'ont permis d'approfondir mes capacités dans des sujets aussi divers et qui restent toujours à la pointe de l'actualité.

7.2 Avis personnel

L'ambiance au sein du laboratoire est très agréable. Cette ambiance et l'environnement de travail m'ont poussé à signer pour une thèse au sein d'Orange Labs.

Annexe A

Génération CGA en C

Liste des fichiers :

Programme principal

```
1  /*
2   * Routine de calcul de cga
3   */
4
5  #include "generate_cga.h"
6
7  void usage(char* arg)
8  {
9      fprintf(stdout, "CGA Generator usage :\n");
10     fprintf(stdout, "%s <CGA.DER>\n", arg);
11 }
12
13 int main(int argc, char *argv[])
14 {
15     if (argc < 2)
16     {
17         usage(argv[0]);
18         return 0;
19     }
20
21     mpz_t prefix64;
22     mpz_init_set_ui(prefix64, 0xfe80);
23     mpz_mul_2exp(prefix64, prefix64, 64-2*8);
24
25     mpz_t publicKey;
26     mpz_init(publicKey);
27
28     readDERFile(publicKey, argv[1]);
29
```

```
30      mpz_t cgaAddress;
31      mpz_init(cgaAddress);
32      cga_parameters_t cgaParams;
33
34      generateCGA_mpz(cgaAddress, &cgaParams, prefix64, publicKey, 1)↩
            ;
35
36      gmp_printf("Generated CGA :\n\t%Zx\n", cgaAddress);
37      fprintf(stdout, "Generated CGA_PARAMETERS : \n");
38      hexdump(&cgaParams, sizeof(cgaParams));
39
40      mpz_clear(cgaAddress);
41      mpz_clear(publicKey);
42      mpz_clear(prefix64);
43
44      return 0;
45  }
```

Librairie

```
1   #ifdef DEBUG
2   #define DEBUGGMP debug_gmp
3   #else
4   #define DEBUGGMP voidfunc
5   #endif
6
7   #include "libcga.h"
8
9   void debug_gmp(char* sentence, mpz_t value)
10  {
11      gmp_printf(sentence);
12      gmp_printf("\n%Zx\n(%d bytes)\n",
13          value,
14          mpz_sizeinbase(value, 16)/2);
15          //strlen(mpz_get_str(NULL, 16, value))/2);
16  }
17
18  void randomMpz(mpz_t value, int nbits)
19  {
20      gmp_randstate_t randstate;
21      gmp_randinit_mt(randstate);
22
23      //TODO Use /dev/random bits
24      unsigned long seed = time(NULL);
25      gmp_randseed_ui(randstate, seed);
26
27      mpz_urandomb(value, randstate, nbits);
28  }
```

```
29
30 void readDERFile(mpz_t keyOut, char* fileIn)
31 {
32     FILE* pFile;
33
34     if ((pFile = fopen(fileIn, "rb")) == NULL)
35     {
36         perror("Error opening key");
37     }
38
39     u_char c;
40
41     while (!feof(pFile))
42     {
43         c = fgetc(pFile);
44         mpz_mul_2exp(keyOut, keyOut, 8);
45         mpz_add_ui(keyOut, keyOut, c);
46     }
47
48     mpz_div_2exp(keyOut, keyOut, 8);
49
50     fclose(pFile);
51 }
52
53 void concatenate(mpz_t concatenated, mpz_t modifier, mpz_t ↩
       publicKey)
54 {
55     mpz_set(concatenated, modifier);
56     mpz_mul_2exp(concatenated, concatenated, 8*9);
57     mpz_mul_2exp(concatenated, concatenated, mpz_sizeinbase(↩
           publicKey, 16)*4);
58     mpz_add(concatenated, concatenated, publicKey);
59 }
60
61 void concatenateFinal(mpz_t result, mpz_t modifier, mpz_t prefix64,↩
       mpz_t collisionCount, mpz_t publicKey)
62 {
63     mpz_set(result, modifier);
64     mpz_mul_2exp(result, result, mpz_sizeinbase(prefix64, 16)*4);
65     mpz_mul_2exp(result, result, mpz_sizeinbase(collisionCount, 16)↩
           *4);
66     mpz_mul_2exp(result, result, mpz_sizeinbase(publicKey, 16)*4);
67     mpz_add(result, result, publicKey);
68 }
69
70
71 void concatenateGen(mpz_t result, ...)
72 {
73     va_list args;
```

```
74      va_start(args, result);
75
76      mpz_set_ui(result, 0);
77
78      mpz_t *value = va_arg(args, mpz_t *);
79
80      while (value != NULL)
81      {
82          //DEBUGGMP("Result: ", result);
83          //DEBUGGMP("va_arg: ", *va_arg(args, mpz_t *));
84
85          mpz_mul_2exp(result, result,
86                  mpz_sizeinbase(*value, 16)*4);
87          mpz_add(result, result, *value);
88          value = va_arg(args, mpz_t *);
89      }
90
91      va_end(args);
92  }
93
94  u_char checkSecZeroBits(const u_char *hash, u_char secParam)
95  {
96      int bit = 0;
97
98      // hash[xx] contains 8 bits, lets match for 2*secParam
99      // TODO Faster calculus
100     // TODO return !(hash>>(160-16*secParam) & 0xffff);
101     while (bit < 2*secParam)
102     {
103         if (hash[bit++] != 0)
104         {
105             return 0;
106         }
107     }
108
109     return 1;
110 }
111
112 void generateCGA_mpz(mpz_t cgaAddress, cga_parameters_t *cgaParam, ↩
        mpz_t prefix64, mpz_t publicKey, u_char secParam)
113 {
114     // Usefull when padding needed as far as we can't define a ↩
            mpz_t size
115     mpz_t zeroPad;
116     mpz_init_set_ui(zeroPad, 0);
117
118     // 1-Set the modifier to a random or pseudo-random 128-bit ↩
            value.
119     // 128-bit modifier
```

```
120     mpz_t modifier;
121     mpz_init(modifier);
122 
123     // For testing purpose [RFC3972] modifier value used.
124     mpz_set_str(modifier,
125                 "89a8 a8b2 e858 d8b8 f263 3f44 d2d4 ce9a", 16);
126     //randomMpz(modifier, 128);
127 
128     DEBUGGMP("Random :", modifier);
129 
130     // 2-Concatenate from left to right the modifier, 9 zero octets↩
        , the
131     // encoded public key, and any optional extension fields.
132     DEBUGGMP("PublicKey :", publicKey);
133 
134     mpz_t concatenated;
135     mpz_init(concatenated);
136 
137     u_char validHash;
138 
139     do
140     {
141         // Adding 18 times zeroPad because zeroPad will be ↩
            represented as
142         // 4 bits.
143         concatenateGen(concatenated, modifier,
144                        zeroPad, zeroPad,
145                        zeroPad, zeroPad,
146                        zeroPad, zeroPad,
147                        zeroPad, zeroPad,
148                        zeroPad, zeroPad,
149                        zeroPad, zeroPad,
150                        zeroPad, zeroPad,
151                        zeroPad, zeroPad,
152                        zeroPad, zeroPad,
153                        publicKey, NULL);
154         DEBUGGMP("Concatenated :", concatenated);
155 
156         // TODO Optionnal Fields
157 
158         // Execute the SHA-1 algorithm on the concatenation.
159         //SHA1Context sha;
160         //SHA1Hash(&sha, concatenated);
161         size_t countExport;
162         u_char *cArray =
163             mpz_export(NULL, &countExport, 1, 1,
164                        0, 0, concatenated);
165         u_char *concatHashSHA1 =
166             SHA1(cArray, countExport, NULL);
```

```
167
168         if (checkSecZeroBits(concatHashSHA1, 1))
169         {
170             validHash = 1;
171         }
172         else
173         {
174             validHash = 0;
175             mpz_add_ui(modifier, modifier, 1);
176             DEBUGGMP("Added 1 to modifier :", modifier);
177             hexdump(concatHashSHA1, 160/8);
178         }
179     } while (!validHash);
180
181     // 4-Set the collision count to zero
182     mpz_t collisionCount;
183     mpz_init_set_ui(collisionCount, 0);
184
185     // 5-Concatenate final modifier value, subnet prefix, collision↩
          count
186     // encoded public key, and any optional extension fields.
187
188     concatenateGen(concatenated, modifier, prefix64, zeroPad,
189                    collisionCount, publicKey, NULL);
190
191     DEBUGGMP("Final concat. :", concatenated);
192
193     // Execute the SHA-1 algorithm on the concatenation.
194     size_t countExport;
195     u_char *cArray =
196         mpz_export(NULL, &countExport, 1, 1, 0, 0, concatenated);
197     u_char *finalHashSHA1 =
198         SHA1(cArray, countExport, NULL);
199
200     hexdump(finalHashSHA1, 160/8);
201
202     // 6-Form an interface identifier from final hash by writing ↩
          the value
203     // of secParam into the three leftmost bits and by setting bits↩
          6 and 7
204     // (ie. "u" and "g") to zero in the 64 first bytes of hash.
205     // TODO Endianness
206     finalHashSHA1[0] =
207         ((secParam & 0x7) << (sizeof(u_char)*8 - 3)) |
208         (finalHashSHA1[0] & 0x1c) | 0;
209
210     mpz_import(concatenated, 64/8, 1, sizeof(u_char), 0, 0, ↩
          finalHashSHA1);
211
```

```
212     DEBUGGMP("Final interface identifier: ", concatenated);
213
214     // 7-Concatenate the 64-bit subnet prefix and the 64-bit ↩
            interface
215     // identifier to form a 128-bit IPv6 address.
216     concatenateGen(cgaAddress, prefix64, concatenated, NULL);
217
218     // 8-Perform duplicate address detection. If collision is ↩
            detected
219     // increment the collision count by one and go back to step 5.
220     // However after three collision, stop and report error.
221     //TODO
222
223     // 9- Form the CGA Parameters data structure.
224     if ((mpz_export(cgaParam->modifier, NULL, 1, 1, 0, 0, modifier)↩
            ) == NULL)
225     {
226         perror("Error while exporting modifier to cga_parameters_t"↩
                );
227     }
228
229     if ((mpz_export(cgaParam->subnetPrefix, NULL, 1, 1, 0, 0, ↩
            prefix64)) == NULL)
230     {
231         perror("Error while exporting prefix to cga_parameters_t");
232     }
233
234     if ((mpz_export(&cgaParam->collisionCount, NULL, 1, 1, 0, 0, ↩
            collisionCount)) == NULL)
235     {
236         perror("Error while exporting collisionCount to ↩
                cga_parameters_t");
237     }
238
239     if ((cgaParam->publicKey = mpz_export(NULL, NULL, 1, 1, 0, 0, ↩
            publicKey)) == NULL)
240     {
241         perror("Error while exporting publicKey to cga_parameters_t↩
                ");
242     }
243
244     // TODO Extension Fields
245     //if ((mpz_export(cgaParam->modifier, NULL, 1, 1, 0, 0, ↩
            modifier)) == NULL)
246     //{
247     //  perror("Error while exporting modifier to cga_parameters_t↩
            ");
248     //}
249
```

```
250     mpz_clear(zeroPad);
251     mpz_clear(modifier);
252     mpz_clear(concatenated);
253     mpz_clear(collisionCount);
254 }
```

Bibliographie

[Aur05] T. Aura. Cryptographically Generated Addresses (CGA). RFC 3972 (Proposed Standard), March 2005. Updated by RFCs 4581, 4982.

[BKN06] M. Bellare, T. Kohno, and C. Namprempre. The Secure Shell (SSH) Transport Layer Encryption Modes. RFC 4344 (Proposed Standard), January 2006.

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722.

[Eas06] D. Eastlake 3rd. Domain Name System (DNS) Case Insensitivity Clarification. RFC 4343 (Proposed Standard), January 2006.

[Har06] B. Harris. Improved Arcfour Modes for the Secure Shell (SSH) Transport Layer Protocol. RFC 4345 (Proposed Standard), January 2006.

[HPFS02] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280 (Proposed Standard), April 2002. Obsoleted by RFC 5280, updated by RFCs 4325, 4630.

[HV10] Bob Halley and Paul Vixie. *Berkeley Internet Name Domain (BIND)*, 2010. `https://www.isc.org/software/bind`.

[Kau05] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), December 2005. Updated by RFC 5282.

[Ken05a] S. Kent. IP Authentication Header. RFC 4302 (Proposed Standard), December 2005.

[Ken05b] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303 (Proposed Standard), December 2005.

[KS05] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005.

[LMK07] J. Laganier, G. Montenegro, and A. Kukec. Using IKE with IPv6 Cryptographically Generated Addresses. Internet-Draft draft-laganier-ike-ipv6-cga-02, Internet Engineering Task Force, July 2007. Obsolete.

[Ste05] Andreas Steffen. *StrongSWAN*, 2005. `http://www.strongswan.org`.

[TNJ07] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862 (Draft Standard), September 2007.