Amélioration des adresses CGA et du protocole SEND pour un meilleur support de la mobilité et de nouveaux services de sécurité

Tony Cheneau

Ecole Doctorale EDITE

Thèse présentée pour l'obtention du diplôme de Docteur de Télécom & Management SudParis

Doctorat conjoint Télécom & Management SudParis et Université Pierre et Marie Curie

Spécialité : Informatique

Par M Tony Cheneau

Amélioration des adresses CGA et du protocole SEND pour un meilleur support de la mobilité et de nouveaux services de sécurité

Soutenue le 07 janvier 2011 devant le jury composé de

Abdelmadjid Bouabdallah	Professeur UTC, Compiègne	Rapporteur
César Viho	Professeur Université de Rennes I	Rapporteur
Sébastien Tixeuil	Professeur Université Pierre et Marie Curie, Paris 6	Examinateur
Claude Castelluccia	Directeur de recherche Equipe PLANETE, INRIA Rhône Alpes	Examinateur
Olivier Heen	Ingénieur de recherche Technicolor	Examinateur
Jean-Michel Combes	Ingénieur de recherche Orange Lab	Examinateur
Maryline Laurent	Professeur Télécom & Management SudParis	Directrice de thèse

Thèse n°2011TELE0002

Thèse

pour obtenir le grade de

docteur de l'Université Pierre et Marie Curie

Amélioration des adresses CGA et du protocole SEND pour un meilleur support de la mobilité et de nouveaux services de sécurité

Thèse défendue par :
Tony Cheneau

Sous la direction de :
Prof. Maryline Laurent

Discipline : Informatique
École Doctorale : École Doctorale d'Informatique, Télécommunications et Électronique de Paris (EDITE)

Composition du jury :

Maryline Laurent	Directrice de thèse
Abdelmadjid Bouabdallah	Rapporteur
César Viho	Rapporteur
Sébastien Tixeuil	Examinateur
Claude Castelluccia	Examinateur
Olivier Heen	Examinateur
Jean-Michel Combes	Examinateur

Abstract

Originally designed to protect the *Neighbor Discovery Protocol* (NDP) (part of the IPv6 protocol suite), the Cryptographically Generated Addresses (CGA) and the Secure Neighbor Discovery (SEND) protocol now need to be adapted to the context of Mobility and extended to new functionalities. The term "Mobility" encompasses many aspects, among them : node mobility (*Mobile IPv6*, MIPv6), router mobility (*Network Mobility*, NEMO) and network-based mobility management (*Proxy Mobile IPv6*, PMIPv6). Numerous changes need to be operated on the SEND protocol in order to comply with the Mobility : the cryptographic operations need to be adapted to operate on low power mobile nodes, the incompatibilities between the address sharing model of the mobile protocol and the address protections offered by SEND need to be fixed, etc.

In a first part of this thesis, we present the *Neighbor Discovery Protocol* and introduce different threats it is vulnerable to. Then, we present the CGA addresses and the SEND protocol. The former enables a node to bind itself a public key. The latter generates a signature for each NDP messages with the associated private key in order to prove the authenticity of the sender. These mechanisms are completed with others extra protections, like anti-replay mechanisms (timestamps, nonce, etc.). We then study the limits of these protections and detail a denial of service (DoS) attack that we discovered during our state of the art. In order to reduce the computations on mobile node and to enhanced performance of the CGA addresses and the SEND protocol, we propose to use the Elliptic Curve Cryptography (ECC) within the SEND protocol instead of RSA. We then evaluate the performance of this proposal. Finally, we extend the CGA addresses and the SEND protocol so they become crypto agile, that is : the protocol is independent of the cryptographic algorithms it relies on. While this work currently enables user to switch to ECC in the SEND protocol, its long term purpose is to enable any cryptographic algorithm.

In a second part of this thesis, we fix incompatibilities between the SEND protocol and the different mobility protocols. To be more precise, SEND prevents third party nodes to emit authenticated NDP messages on behalf of a node, while the mobility protocols rely on third party nodes to send or modify NDP signaling messages when the node is away from its home link. After introducing the context and a presentation of the existing solutions, we introduce and detail our new extension to the CGA addresses named "Multiple key CGA". These addresses can contain multiple keys and thus authorise multiple nodes to share an address. In order to prove the feasibility of this proposal, we developed a new implementation of the CGA addresses and the SEND protocol named "NDprotector". This implementation allows us to prove the validity of each of our proposals.

The third and last part of this thesis depicts multiple derived usages of the CGA

addresses and the SEND protocol. We first present the state of the art of the existing solutions. Then we introduce two of our proposals. Firstly, we present a new address generation mechanism based on the CGA addresses which provides anonymity to the nodes by preventing attackers to bind multiple connexions together. Secondly, we detail a key broadcast mechanism that operates in MANET networks. This proposal is named Trustful Authentication and Key Exchange Scheme (TAKES). Finally, the security of this proposal is formally validated through the use of the BAN logic.

We conclude by reminding that the NDP needs to be modified in order to handle Mobility. Within our different work (performance enhancements of the CGA addresses and the SEND protocol, cryptographic algorithm selection mechanism, address sharing, ...) we proved that these modifications are possibles. We also showed that the CGA addresses can be extended to new usages where no security infrastructure is present. All these proposals have been practically validated by our NDprotector implementation.

Keywords : Neighbor Discovery Protocol, IPv6, Cryptographically Generated Addresses, Secure Neighbor Discovery Protocol, Mobility, MIPv6, PMIPv6, performance study, anycast, Proxy Neighbor Discovery.

Résumé

À l'origine conçus pour protéger le protocole de Découverte de Voisins (*Neighbor Discovery Protocol*, NDP) en IPv6, les adresses générées de manière cryptographique (*Cryptographically Generated Addresses*, CGA) et le protocole SEND (*Secure Neighbor Discovery*) doivent maintenant s'adapter au contexte de mobilité et à ses nouvelles fonctionnalités. Cette mobilité revêt de nombreuses formes : mobilité du nœud (*Mobile IPv6*, MIPv6), mobilité des routeurs (*Network Mobility*, NEMO) ou encore mobilité gérée par le réseau (*Proxy Mobile IPv6*, PMIPv6). Pour ce faire, de nombreux changements doivent être opérés dans le protocole SEND : les opérations cryptographiques doivent être allégées pour les terminaux à faible capacité de calcul, les incompatibilités entre le partage d'adresse dans les protocoles de mobilité et le mécanisme de protection d'adresse de SEND doivent être corrigées, etc.

Dans une première partie de cette thèse, nous présentons le protocole de Découverte de Voisins et introduisons les différents vecteurs d'attaques qui le menacent. Nous présentons ensuite les adresses CGA et le protocole de sécurité SEND. Les premières permettent de lier une clé publique à un nœud tandis que le second génère une signature sur chaque message NDP avec la clé privée associée pour prouver l'authenticité de l'émetteur de ces messages. À ce mécanisme, viennent s'ajouter plusieurs protections supplémentaires, comme la protection contre les attaques par rejeux (utilisation d'un horodatage des messages, *nonce*, etc.). Ensuite, nous étudions les limites de ces protections et détaillons une attaque de type déni de service qui a été trouvée lors de l'étude du protocole SEND. Afin de réduire le coût en calcul sur les terminaux mobiles et d'améliorer les performances des adresses CGA et du protocole SEND, nous proposons alors de remplacer l'utilisation des clés RSA dans CGA et SEND par l'utilisation de la cryptographie basée sur les courbes elliptiques (*Elliptic Curve Cryptography*, ECC). Pour ce faire, dans une première étape, nous comparons les différentes durées dépendantes des calculs cryptographiques dans l'utilisation des adresses CGA et du protocole SEND et évaluons les gains de l'utilisation d'ECC sur l'algorithme RSA. Dans une seconde étape, nous modifions les CGA et le protocole SEND afin qu'ils soient indépendants du type d'algorithme cryptographique utilisé. Ce travail permet non seulement d'utiliser des clés ECC dans le protocole SEND, mais offre également une possibilité de transition vers de nouveaux algorithmes cryptographiques dans le futur.

Dans une deuxième partie de la thèse, nous tentons de résoudre une incompatibilité entre le protocole SEND et les différents protocoles de mobilité. En effet, ce premier impose que seul le nœud possédant l'adresse soit autorisé à émettre des messages NDP tandis que les seconds reposent sur l'utilisation d'un nœud tiers pour effectuer l'envoi ou la modification de messages NDP quand le nœud mobile n'est plus présent sur le lien. Après avoir présenté plus en détail la problématique et les solutions existantes,

nous introduisons et proposons une extension des CGA que nous nommons “CGA multi-clés”. Celles-ci sont capables de contenir plusieurs clés publiques et donc autorisent plusieurs nœuds à partager une adresse. Afin de prouver la faisabilité de ces différentes propositions, nous avons développé une nouvelle implémentation des CGA et du protocole SEND appelée NDprotector. Cette implémentation nous permet de valider par la pratique chacune de nos propositions.

La troisième et dernière partie de la thèse dépeint plusieurs utilisations dérivées des adresses CGA et du protocole SEND. Nous faisons d'abord un état de l'art des solutions existantes. Puis nous proposons deux de nos contributions. Nous présentons d'une part un nouveau mécanisme de génération d'adresses basé sur les CGA qui permet à un nœud d'obtenir l'anonymat (en empêchant un attaquant de lier plusieurs connexions entre elles). D'autre part, nous détaillons un mécanisme de diffusion de clé publique nommé TAKES. La sécurité du mécanisme est validée de façon formelle en utilisant la logique BAN.

Nous concluons en insistant sur le fait que le protocole de Découverte de Voisins doit nécessairement être modifié pour gérer les différentes formes de mobilité. Au travers de nos travaux (amélioration et étude des performances des adresses CGA et du protocole SEND, mécanisme de sélection de l'algorithme cryptographique, partage des adresses, ...) nous avons montré que ces modifications sont possibles. Nous avons aussi démontré que les CGA peuvent être étendues à de nouveaux usages dans les réseaux où aucune infrastructure assurant la sécurité n'est disponible. Tous ces résultats ont été validés sur notre implémentation NDprotector.

Mots clés : Neighbor Discovery Protocol, IPv6, Cryptographically Generated Addresses, protocole de découverte de voisins sécurisé, mobilité, MIPv6, PMIPv6, étude de performance, anycast, Proxy Neighbor Discovery.

Remerciements

J'ai souvent eu l'occasion de lire ces longues lignes de remerciements en début de documents. Aujourd'hui alors que je commence à écrire moi aussi ces mots destinés à ceux qui m'ont aidés tout au long de la thèse, je pense qu'il convient de m'excuser : je sais par avance que j'oublierai de citer explicitement quelques-uns d'entre vous, même si je sais pourtant que votre présence aura été nécessaire pour transformer ces trois ans d'études en une expérience unique et enrichissante. Alors, si vous ne trouvez pas votre nom dans cette page, sachez que l'oubli n'est que sur le papier.

En trois ans passés à l'Institut Télécom SudParis, j'ai en effet eu l'occasion de faire de nombreuses connaissances intéressantes.

Je remercie mes différents collègues de bureaux, Chakib, Providence, Wassim, Aymen, Andrei, pour la bonne humeur et les discussions (souvent techniques, jamais ennuyeuses) qu'ils y ont apportées.

Je pense aussi à ceux avec lesquels j'ai partagé de nombreux repas, conversations dans les couloirs et voyage dans les transports, Javier, Anis, Pétros, Jordi, Marc, Sophie et Amel.

Je tiens également à remercier mes amis de l'industrie, Jean-Michel, Michaela, Sean, Daniel qui m'ont grandement aidé lors de mon travail sur la *Signature Algorithm Agility* dans SEND et m'ont apporté leur expertise technique et leurs conseils.

Pour finir, je tiens à remercier les personnes qui ont fait en sorte que le travail de cette thèse soit possible. Merci à Maryline, pour m'avoir donné la chance de faire cette thèse, m'avoir supporté et aidé à travers les différentes étapes. Merci à Hakima, pour ses conseils toujours pertinents. Merci à mes parents, pour m'avoir poussé à faire des études et toujours accompagné. Et pour finir, merci à ma femme Julie, pour son indéfectible soutien et ses encouragements permanents.

Table des matières

1 Introduction de la thèse

1.1 Contexte de la thèse

Lorsque le protocole IPv4 fut conçu par l'IETF (*Internet Engineering Task Force*) en 1980, un tel engouement pour l'Internet n'avait pas été prévu et l'encodage des adresses sur 32 bits semblait suffisant. L'incroyable croissance de l'Internet n'a cessé de puiser dans cette réserve de plusieurs milliards d'adresses et même si la création de nouveaux mécanismes comme le NAT (*Network Address Translation*) a temporairement étendu la durée de vie de l'IPv4, il semble aujourd'hui que le protocole IPv4 vit ses dernières années. Les pronostiques les plus optimistes estiment que la pénurie d'adresses IPv4 sera atteinte d'ici 2012. Cela indique que la transition vers le protocole IPv6, qui offre un nombre bien plus important d'adresses, est désormais inéluctable.

Les spécifications du protocole IPv6 ont été publiées en 1998 et depuis, tout un écosystème de protocoles liés à l'IPv6 a fleuri. Toutefois, le protocole IPv6 n'a pas eu le succès escompté. En effet, la phase de transition a été retardée par une inertie des acteurs du domaine réseau qui ne souhaitent pas modifier leur architecture et préfèrent rester sur des protocoles éprouvés. Cette tendance a été confortée depuis que les protocoles initialement développés par l'IPv6 et répondant à des besoins précis de l'industrie ont été (retro-)portés vers l'IPv4. Parmi ces protocoles, l'on trouve le protocole de sécurité IPsec ou encore le protocole de Mobilité *Mobile IP*.

De nos jours, l'on voit apparaître les premiers déploiements de l'IPv6 : expérimentations chez le fournisseur Internet français Free ou de la compagnie Google pour son moteur de recherche. Ces déploiements ont permis de mettre en avant des problèmes de jeunesse du protocole IPv6 : problèmes d'implémentation et parfois, plus grave, problèmes de spécifications. Ainsi, de manière paradoxale, si le protocole IPv6 est présent depuis des années, il n'en est qu'à ses premiers balbutiements et doit encore faire ses preuves auprès de la communauté du réseau.

En dépit de cette apparente jeunesse, beaucoup de problèmes fondamentaux du protocole IPv4 ont été identifiés et résolus dans la version 6. Il s'agit là de changements bien connus, tels que l'encodage des adresses, maintenant sur 128 bits et qui comme nous l'avons dit offre un nombre d'adresses plus que suffisant (3.4×10^{38} adresses), mais également de l'ajout de fonctionnalités, telles que l'*Autoconfiguration d'Adresse Sans État*. Ce mécanisme, à l'image du protocole DHCP (*Dynamic Host Configuration Protocol*) en IPv4, permet à un nœud IPv6 d'obtenir une adresse IPv6 topologiquement valide sans intervention de la part de son utilisateur. L'originalité de ce mécanisme est de ne pas nécessiter la présence de serveur central. Il est basé sur le protocole de Découverte de Voisins (*Neighbor Discovery Protocol*, NDP), un dérivé de l'ARP (*Address Resolution Protocol*), qui permet, entre autres, d'assurer la correspondance entre une

adresse IPv6 et une adresse de couche liaison (par ex. adresse MAC). Néanmoins, le NDP souffre des problèmes de sécurité de son ancêtre ARP (usurpation d'adresses, etc.) et afin de les parer, il se dote d'une couche de sécurité nommée SEND (*Secure Neighbor Discovery*). Le protocole SEND permet d'assurer la protection des messages utilisés dans le NDP via l'usage d'adresses IPv6 générées de manière cryptographique (*Cryptographically Generated Addresses*, CGA). Ces adresses permettent de lier une clé publique à une adresse IPv6. Par la suite, l'ajout d'une signature réalisée à partir de la clé privée associée à cette clé publique à un message émis par le nœud permet de prouver l'origine du message. Appliqué au NDP, cela permet de prouver également la possession de l'adresse IPv6. Cette preuve de possession est d'autant plus importante à l'heure où l'on s'interroge sur l'utilisation de l'adresse IP comme donnée à caractère personnel.

En marge, de nombreux types d'équipements légers (téléphones intelligents, capteurs, etc.) se mettent à utiliser le protocole IPv6. Une de nos premières contributions vise à diminuer la complexité de calcul liée à la protection cryptographique offerte par le protocole SEND afin qu'il devienne utilisable sur ces appareils. Cette approche est double : d'une part nous proposons et évaluons l'introduction d'un nouvel algorithme cryptographique plus léger pour améliorer les performances du protocole SEND, d'autre part nous modifions le protocole pour que n'importe quel algorithme (présent et à venir) puisse être utilisé.

Par la suite, nous nous intéressons à définir de nouveaux scénarios d'utilisation pour les CGA. En effet, celles-ci permettent de prouver l'origine des messages sans besoin d'infrastructure de sécurité extérieure pour assurer cette preuve. Ainsi, il nous est possible de définir de nouvelles architectures de sécurité dans les environnements où aucune infrastructure de sécurité classique (PKI, tierce partie de confiance , etc.) n'est disponible.

1.2 Contributions de la thèse

À travers cette thèse, nous avons analysé les différents aspects des adresses CGA et du protocole SEND (Chapitre 2 à Chapitre 3). Nous avons ensuite exploré plusieurs axes afin d'étendre leurs usages (fin du Chapitre 3 au Chapitre 9). Finalement, le Chapitre 10 contient notre conclusion et présente les axes de recherches restants à explorer.

1.2.1 Découverte d'une faille dans le protocole de sécurité SEND

Dans le Chapitre 3, nous avons étudié les spécifications du protocole de sécurité SEND. Ce protocole ajoute de nouvelles options au protocole de Découverte de Voisins afin de protéger ses messages. Durant cette étude, nous nous sommes intéressés à la présence de failles au sein même de l'implémentation. Nous avons découvert un manque de spécifications autour du mécanisme de protection contre le rejeu de messages. Ainsi, il est possible de rejouer les messages émis durant la procédure de Détection d'Adresse Dupliquée (*Duplicate Address Detection*, DAD) du NDP. Dans la pratique, cette at-

taque locale empêche tout nouveau nœud joignant un réseau attaqué d'obtenir une adresse via le mécanisme d'*Autoconfiguration d'Adresse Sans État* et donc de pouvoir communiquer. Nous proposons les modifications à apporter aux spécifications du protocole SEND afin de se prémunir contre cette attaque.

1.2.2 Étude et amélioration des performances des adresses CGA et du protocole SEND

Avec la forte croissance de l'utilisation de nœuds mobiles à faible capacité de calculs (téléphones intelligents, *netbooks*, capteurs, etc.), il convient de ré-évaluer les contraintes en terme de performances imposées aux protocoles de sécurité. Nous nous y attachons dans le Chapitre 4. Ce point est d'autant plus préoccupant que les adresses CGA et le protocole SEND sont actuellement limités aux usages de l'algorithme de signature RSA et de la fonction de hachage SHA-1.

Dans un premier temps, nous faisons le postulat que l'utilisation de la cryptographie basée sur les courbes elliptiques (Elliptic Curve Cryptography, ECC) permet d'améliorer de manière significative les performances des adresses CGA et du protocole SEND. Nous comparons alors les performances mesurées lors de l'utilisation des adresses CGA et du protocole SEND avec les algorithmes RSA et ECC. Nous constatons alors que le gain en performance est réel dans les deux scénarios où nous avons effectué nos tests : sur un ordinateur de bureau de faible puissance et sur un *tablet PC*.

Dans un deuxième temps, nous évaluons les conséquences des récentes attaques sur SHA-1 et la transition vers de nouvelles fonctions de hachage. Nous mettons en place une batterie de tests pour la nouvelle génération de fonctions (SHA-256, SHA-512, TIGER2, etc.) et effectuons un classement de celles-ci en fonction de leur comportement.

Dans un troisième et dernier temps, nous proposons l'utilisation de techniques GPGPU (*General-Purpose Graphical Processing Units*) afin d'accélérer la génération d'adresses CGA. Nous concluons qu'il est possible de tirer partie de la puissance de calcul d'une carte graphique afin d'accélérer les temps de calculs d'adresses CGA.

1.2.3 Extensions des adresses CGA et du protocole SEND pour ajouter le support de nouveaux algorithmes cryptographiques

Notre contribution précédente met en lumière le besoin de migrer le protocole SEND vers de nouveaux algorithmes cryptographiques. Nous proposons dans cette contribution (Chapitre 5) les modifications nécessaires au protocole SEND et aux adresses CGA pour gérer de nouveaux algorithmes. Ce travail, nommé *Signature Algorithm Agility*, permet d'anticiper toutes futures migrations et met en place les différents mécanismes nécessaires pour une transition douce vers de nouveaux algorithmes. Nous nous intéressons également à l'interopérabilité entre les nœuds implémentant différents types d'algorithmes cryptographiques et proposons des outils pour faciliter leur communication (introduction du rôle de notaire).

Cette contribution a été proposée à la normalisation dans le groupe de travail lié à la maintenance des adresses CGA et du protocole SEND à l'IETF.

1.2.4 Extensions des adresses CGA et du protocole SEND pour améliorer le support de la Mobilité et des adresses Anycast

Le protocole de Mobilité *Mobile IPv6* (MIPv6) repose sur l'utilisation de la signalisation du protocole de Découverte de Voisins afin de simuler la présence du nœud mobile sur son lien mère lorsque celui-ci y est absent. Par ailleurs, le protocole *Proxy Mobile IPv6* (PMIPv6) se base sur l'assignation d'adresses anycast (c.-à-d. des adresses partagées entre plusieurs nœuds) sur les routeurs pour offrir au nœud mobile une adresse de routeur identique sur les différents points d'accès du réseau. Toutefois, ces utilisations sont incompatibles avec les mécanismes de sécurité de SEND qui permettent au seul et unique possesseur de l'adresse d'émettre des messages à partir de celle-ci. Dans cette contribution (Chapitre 7), nous proposons une utilisation dérivée du mécanisme de *Signature Algorithm Agility* pour partager la possession de l'adresse entre plusieurs nœuds.

1.2.5 Implémentation des adresses CGA et du protocole SEND

Dans le Chapitre 6, nous présentons une implémentation des adresses CGA et du protocole SEND que nous avons développé. Celle-ci, nommée NDprotector, a pour principal objectif de servir de terrain d'expérimentation pour chacune de nos propositions. Ainsi, nous y intégrons nos précédentes contributions, comme la *Signature Algorithm Agility* et l'utilisation des courbes elliptiques dans le protocole SEND.

1.2.6 Mécanisme de génération d'adresse IPv6 offrant l'anonymat

Dans cette contribution (Chapitre 8), nous constatons qu'il n'existe pas de mécanisme de génération d'adresses dans le protocole IPv6 permettant à la fois d'assurer la protection de l'adresse (contre les attaques) et d'offrir l'anonymat. Cet anonymat se traduit dans le monde IP par une impossibilité pour un attaquant de corréler deux actions effectuées par un même nœud. À cet effet, nous introduisons alors deux solutions : les *CGA Éphémères Pseudo-Anonymes* et *CGA Éphémères Anonymes*. La première offre l'anonymat à la couche 3, en générant des adresses CGA dont la durée de vie est liée à la durée de vie de la communication (par ex. une session TCP). Le second offre l'anonymat à la couche 2 et à la couche 3 et étend le concept d'identifiant à durée de vie limitée à la couche 2. Nous implémentons la première proposition dans NDprotector afin d'évaluer l'impact de celle-ci sur le temps d'établissement de nouvelles connexions.

1.2.7 Mécanisme de diffusion de clés publiques à plusieurs sauts dans les réseaux MANET

Dans cette dernière contribution (Chapitre 9), nous définissons un mécanisme de diffusion de clés publiques à plusieurs sauts nommé TAKES (pour *Trustful Authentication and Key Exchange Scheme*). Nous souhaitons permettre aux participants de réunions de petite ou moyenne importance (par ex. conférence) de démarrer des services de sécurité là où une infrastructure de réseau ad-hoc ne le permet pas (absence

d'accès Interet, et donc d'infrastructure PKI, etc.). Par ailleurs, nous ajoutons une dimension supplémentaire à notre solution : celle-ci doit être assez simple à mettre en place pour qu'un utilisateur non-initié au domaine de la sécurité soit en mesure de l'utiliser. Finalement, la validité de cette contribution a été vérifiée de manière formelle en appliquant la logique BAN au format des messages que nous avons défini.

2 Le protocole de Découverte de Voisins

Dans ce chapitre, nous présentons le protocole de Découverte de Voisins IPv6. En Section 2.1, nous situons le rôle du protocole de Découverte de Voisins dans la suite de protocoles IPv6. Puis, dans les Sections 2.2, 2.3 et 2.4, nous décrivons les fonctionnalités du protocole ainsi que les adresses spécifiques et les messages permettant de les mettre en œuvre. Finalement, comme nous le montrons Section 2.5, ce protocole est vulnérable à de nombreuses attaques.

2.1 Place du protocole de Découverte de Voisins dans le protocole IPv6

Il y a maintenant plus de 10 ans, la suite de protocoles IPv6 [DH98] a vu le jour. Elle a été initialement conçue en prévision de la pénurie d'adresses IPv4 que nous connaissons aujourd'hui et afin de corriger les défauts de conception du protocole IPv4. Depuis, le protocole IPv6 a été étendu et accueille de nouvelles fonctionnalités telles que *Mobile IP* [JPA04] et *Network Mobility* [DWPT05].

Le changement majeur de la version 6 est l'augmentation de la taille des adresses. Le passage de 32 bits à 128 bits a plusieurs intérêts : il permet d'augmenter le nombre de nœuds connectés en simultané à l'Internet, supprime le besoin du mécanisme de traduction d'adresse (*Network Address Translation*, NAT) qui nuit aux communications de bout en bout et réduit la taille des tables de routage (grâce à une allocation par zone géographique des blocs d'adresses).

Au sein de la suite de protocoles IPv6, le protocole de Découverte de Voisins [NNSS07] (*Neighbor Discovery Protocol*, NDP), opérant au premier saut, contient nombre de fonctionnalités indispensables au bon fonctionnement des réseaux IPv6. Parmi celles-ci, on pourra noter la fonctionnalité de résolution d'adresse (*Address Resolution*), qui correspond à ce qu'est ARP [Plu82] (*Address Resolution Protocol*) pour le protocole IPv4, et l'*Autoconfiguration d'Adresse Sans État* (*Stateless Adddress Autoconfiguration*, SLAAC), facilitant l'assignation des adresses et la configuration des nœuds au sein d'un sous-réseau.

2.2 Fonctionnalités du protocole de Découverte de Voisins

Dans le protocole NDP, le terme “voisin” désigne un nœud qui se trouve à un saut (c.-à-d. sur le lien) et, plus naturellement, le terme“voisinage” désigne l'ensemble des voisins d'un nœud.

Dans ce protocole, et plus particulièrement dans le protocole IPv6, il existe une classification des nœuds en deux types. D'une part, les nœuds qui assurent une fonctionnalité de routage et de transfert de paquets, nommés “routeurs”. D'autre part, les nœuds qui n'assurent pas ces fonctions, “hôtes”. Cette séparation entre routeurs et hôtes est omniprésente dans le protocole de Découverte de Voisins. En effet, selon le type de nœuds, l'action à effectuer pour réaliser une fonctionnalité sera différente.

Le NDP propose neuf fonctionnalités limitées au premier saut :

- la *redirection* permet à un routeur d'indiquer à un hôte que son correspondant est en réalité sur le même lien (c.-à-d. un voisin) afin que le trafic lui soit directement destiné. Cela est utile dans la pratique lorsque deux voisins sont sur un même lien physique et communiquent au travers d'un routeur ;
- la *résolution d'adresse* permet à chaque nœud, à l'image d'ARP [Plu82] en IPv4, d'établir la correspondance entre une adresse IPv6 et une adresse de couche liaison (par ex. l'adresse MAC pour le protocole Ethernet). Par la suite, afin d'éviter d'avoir à réitérer la procédure à chaque paquet émis, la correspondance *adresse IP* $\leftrightarrow$ *adresse couche liaison* est stockée en interne par le nœud dans une table nommée *cache de voisinage* (*Neighbor Cache*, NC) ;
- après avoir effectué la *résolution d'adresse*, un nœud détermine l'état de connectivité d'un voisin via la procédure de *détection de voisins non joignables* (*Neighbor Unreachability Detection*, NUD). Lorsque le nœud et son voisin communiquent, les couches de niveau supérieur suffisent à indiquer la présence du voisin sur le lien. Néanmoins, dans l'éventualité où aucun message n'est transmis entre ces deux nœuds pendant un certain laps de temps, le nœud effectuera des procédures de *résolution d'adresse* pour s'assurer de la présence de son voisin ;
- la *découverte de routeurs* permet à un hôte de découvrir les routeurs qui sont situés à un saut (c.-à-d. sur le lien) et de construire sa liste de routeurs par défaut ;
- la *sélection du prochain saut* est ensuite réalisée en se basant sur la liste de routeurs par défaut. Par la suite, le nœud choisit parmi ces routeurs celui auquel il transmettra son trafic afin d'atteindre les destinations pour lesquelles il ne connaît aucune route ;
- la *découverte de préfixes* permet à un hôte d'apprendre les préfixes sous-réseau disponibles sur le lien. Le préfixe détermine l'état dans lequel seront les adresses qui en seront dérivées. Comme le montre la Figure 2.1, elles peuvent être : “préférées” (de nouvelles communications peuvent utiliser ces adresses comme adresse source), “dépréciées” (les communications en cours peuvent toujours utiliser ces adresses, mais pas les nouvelles communications), et “invalides” (aucune communication ne doit plus utiliser l'adresse). Ces trois états sont très utiles pour la renumérotation des réseaux. À noter qu'il est possible que plus d'un préfixe sous-réseau soit valide

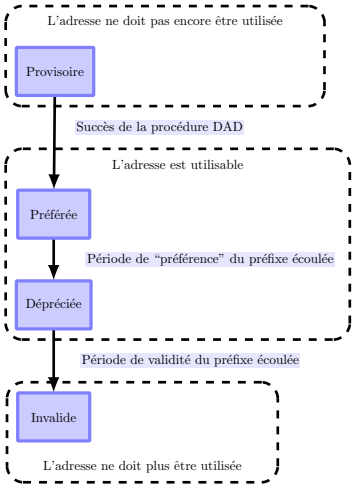


FIGURE 2.1: Cycle de vie d'une adresse construite par l'*Autoconfiguration d'Adresse Sans État*

pour un même lien ;

- la *découverte de paramètres* permet au nœud d'apprendre les informations relatives au sous-réseau. Ainsi, la *découverte de paramètres* permet l'apprentissage de la MTU (Maximum Transmission Unit) appliquée au lien ;
- l'*Autoconfiguration d'Adresse Sans État* (*Stateless Address Autoconfiguration*, SLAAC), décrit dans un document séparé [TNJ07], permet au nœud de construire une adresse IPv6 globale à partir d'un préfixe sous-réseau appris lors de la *découverte de préfixes* et de l'assigner à l'interface où le préfixe a été découvert. La partie identifiant d'interface, qui compose les 64 bits de poids faible de l'adresse, quant à elle, est dérivée à partir de l'adresse de couche liaison. Pour le protocole Ethernet, l'identifiant d'interface est réalisé en dérivant l'identifiant EUI-64 correspondant à l'adresse MAC généralement codée sur 48 bits (cf. Fig. 2.2) ;
- finalement, la *détection d'adresse dupliquée* (*Duplicate Address Detection*, DAD), similaire à la technique de "gratuitous ARP" en IPv4, s'assure que l'*Autoconfiguration d'Adresse Sans État* (Fig. 2.2) n'a pas généré d'adresse IPv6 en double. Si le nœud détecte une adresse dupliquée durant l'*Autoconfiguration d'Adresse Sans État* (ou durant tout mécanisme de construction d'adresse et même en cas d'assignation manuelle), il n'utilise alors pas l'adresse. Durant la procédure, l'adresse n'est pas assignée à une interface et est dite "provisoire". En cas de succès de la procédure, l'adresse passe en état "préférée" (comme indiqué Fig 2.1). Il faut toutefois faire attention, car le mécanisme est assez simple (émission d'une requête à destination de l'adresse dont il veut vérifier l'unicité) et ne permet pas de détecter une adresse dupliquée dans certains cas. Notamment, lors de la fusion de réseaux ou lorsque deux nœuds se trouvant initialement sur des liens séparés se retrouvent réunis, la procédure DAD est inadaptée.

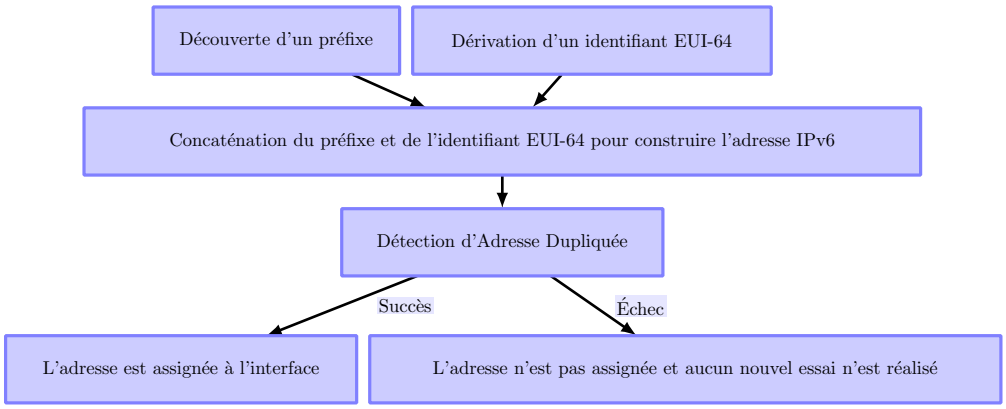


FIGURE 2.2: Construction d'une adresse IPv6 via le mécanisme d'*Autoconfiguration d'Adresse Sans État*

Dans la pratique, ces fonctionnalités sont implémentées dans le protocole de Découverte de Voisins grâce à des messages de type ICMPv6 [CDG06], un protocole de couche transport défini au-dessus du protocole IPv6 qui permet de remonter des informations et des erreurs en provenance du réseau.

2.3 Différents types d'adresses IPv6

Le NDP définit de nouveaux messages ICMPv6 afin d'implémenter les fonctionnalités présentées précédemment. Ces messages peuvent être envoyés à trois types d'adresses différentes : une adresse unicast (qui ne désigne qu'un seul nœud), une adresse anycast (une même adresse est partagée entre plusieurs nœuds) et une adresse multicast (l'adresse est en réalité un groupe de nœud). Ces trois modèles de communication sont décrits plus en détail dans le document [HD06].

De la même façon, il existe un classement parmi les adresses unicast en fonction de leur portée :

- les adresses lien-local, dont le préfixe est `fe80::/64`[1], ont une portée limitée au lien (premier saut). Elles ne sont pas routables, et permettent principalement de s'adresser aux machines du voisinage quand aucun préfixe sous-réseau n'est disponible sur le lien (c.-à-d. aucun routeur n'est connecté) ;
- les *Unique Local Address* (ULA, définies dans [HH05]), dont le préfixe est `fc00::/7`, ont pour portée un site. Elles permettent la mise en place de réseaux privés en IPv6 ;
- les adresses unicast globales ont, comme leur nom l'indique, une porte globale et sont routables. Ce sont des adresses publiques.

1. Cette notation indique que les 64 bits les plus à gauche (de poids fort) de l'adresse font partie du préfixe.

Dans le cadre du protocole de Découverte de Voisins, trois groupes multicast ayant une portée limitée au lien sont également utilisés. Ceux-ci sont définis dans le document [HD06] et sont les suivants :

- l'adresse multicast "tous les nœuds" (*all-node multicast address*). Ce groupe est composé de tous les nœuds. Dans le NDP, il devient possible de communiquer avec ses voisins grâce à ce groupe. L'adresse IPv6 de ce groupe est `ff02::01` ;
- l'adresse multicast "tous les routeurs" (*all-router multicast address*). Ce groupe est composé de tous les routeurs. Cette adresse est utilisée lors de la *découverte de routeurs*, quand un hôte souhaite communiquer à tous les routeurs sur le lien. L'adresse IPv6 de ce groupe est `ff02::02` ;
- l'adresse multicast "du nœud sollicité" (*solicited-node multicast address*) désigne une adresse de groupe particulière correspondant aux nœuds possédant les mêmes 24 bits les plus à droite (de poids faible) dans leur adresse IPv6. Dans la pratique, cette adresse permet de recevoir des messages lorsque le nœud n'a pas assigné d'adresse sur son interface, comme c'est le cas, durant la procédure de *Détection d'Adresse Dupliquée*. L'adresse IPv6 de chaque groupe est formée en concaténant le préfixe `ff02:0:0:0:0:1:ff00::` et les 24 bits de poids faible de l'adresse IPv6 du nœud.

Afin d'écouter ces groupes, le nœud doit au préalable émettre une requête de *Multicast Listener Discovery* (MLD) [DFH99].

2.4 Messages du protocole de Découverte de Voisins

Les différentes fonctionnalités du protocole de Découverte de Voisins sont réalisées par cinq types de messages ICMPv6 différents :

- le message *Neighbor Solicitation* (NS), dont le format est présenté Figure 2.3, permet à un nœud de demander l'adresse de couche liaison d'un voisin et ainsi d'initier la procédure de *résolution d'adresse*. Bien que le champ "adresse de destination" de l'en-tête IPv6 indique le nœud auquel s'adresse la requête, le champ "adresse cible" permet, là où le champ "adresse de destination" peut rester imprécis et pointer une adresse multicast, de désigner l'adresse qui doit être résolue. Ce message peut lui-même contenir, sous forme d'option, l'adresse de couche liaison du nœud, permettant ainsi au destinataire du message d'apprendre l'adresse couche liaison de l'émetteur. Il permet de réaliser la fonctionnalité de *résolution d'adresse* (et par conséquent de NUD) ainsi que la procédure DAD ;
- le message *Neighbor Advertisement* (NA) est le plus souvent la réponse à une requête *Neighbor Solicitation*. En recevant cette réponse, l'initiateur de la requête apprend l'adresse de couche liaison du nœud, complétant ainsi le plus souvent l'échange de messages nécessaire à la procédure de *résolution d'adresse*. Il est aussi possible, comme dans le cas d'un changement de carte réseau, que le nœud émette spontanément un message NA à destination de l'adresse multicast "*All Nodes*" afin d'informer ses voisins d'un changement d'adresse couche liaison ;
- le message *Router Solicitation* (RS) est la requête qui permet à un hôte de déclencher le message d'envoi de *Router Advertisement* de la part des routeurs.

Le plus souvent, cette requête est destinée à l'adresse multicast "*All Routers*" (à destination de tous les routeurs). Afin d'optimiser la durée de la *découverte de routeurs, de préfixes et de paramètres* et de l'*Autoconfiguration d'Adresse Sans État* , cette requête est envoyée dès qu'un nœud s'attache à un lien ;

– le message *Router Advertisement* (RA), dont le format est présenté Figure 2.4, est la réponse à la requête RS. Il permet d'indiquer à un ou plusieurs hôtes (quand la réponse est à destination de l'adresse multicast "*All Nodes*) quels sont les préfixes et paramètres utilisés sur le lien. Ainsi, le temps durant lequel un nœud pense que son correspondant est toujours joignable (entre deux messages durant la procédure de NUD) est contenu par le champ *Durée de joignabilité*. Le champ "durée de vie du routeur" indique, quand la valeur est non-nulle, que le routeur peut être utilisé pour construire la liste des routeurs par défaut et la durée de validité de l'entrée correspondante. Ce message permet de réaliser la *découverte de routeurs, de préfixes et de paramètres*, ainsi que l'*Autoconfiguration d'Adresse Sans État* . À l'image du message NA, un routeur peut émettre un message RA de manière spontanée, afin d'annoncer un changement de configuration dans le sous-réseau (annonce d'un nouveau préfixe sous-réseau, etc). Toutefois, il est plus fréquent qu'un routeur émette de manière régulière ses annonces RA afin d'informer les hôtes de la prolongation de validité de préfixes annoncés ;
– le message *redirect* a pour unique fonction d'implémenter la fonctionnalité de *redirection*. Il est émis par un routeur à destination du nœud dont le trafic doit être redirigé.

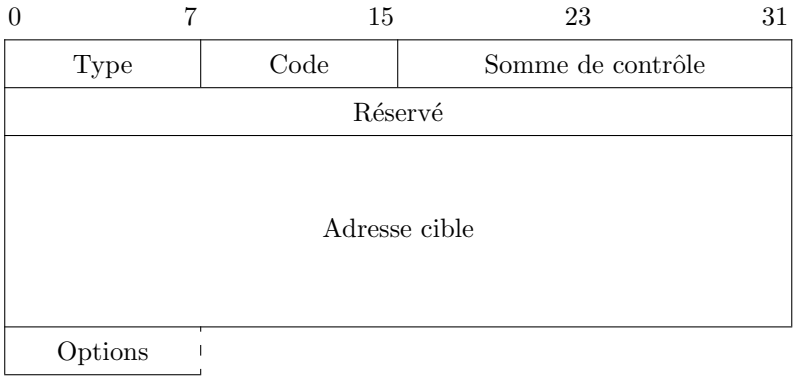


FIGURE 2.3: Format du message ICMPv6 de type *Neighbor Solicitation*

À noter que pour limiter la portée de messages du NDP, la valeur du champ *nombre de sauts* (*hop limit*) de l'entête IPv6 est fixée à 255 (valeur maximale). Dans une utilisation normale de ce champ, sa valeur est décrémentée à chaque saut et le paquet IP est détruit quand la valeur atteint 0. Ce mécanisme permet d'éviter qu'un paquet ne circule indéfiniment en cas de boucles. Pour les messages du protocole NDP, ce mécanisme est légèrement modifié (comme décrit dans le document [GHM+07]) afin

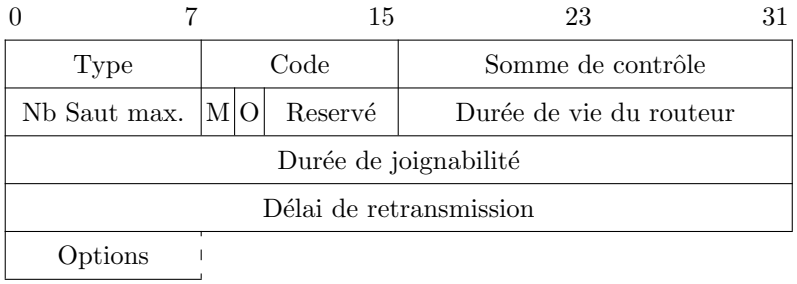


FIGURE 2.4: Format du message ICMPv6 de type *Router Advertisement*

que les messages dont la valeur *nombre de sauts* n'est pas à 255 soient détruits. Ainsi, il est assuré que les messages du NDP ne dépassent pas le premier saut.

Comme nous venons de le voir, la signification de ces messages est fonction des options qui les accompagnent :

- les options *adresse couche liaison de la source* (*Source Link-Layer Address option*, SLLAO) et *adresse couche liaison de la cible* (*Target Link-Layer Address option*, TLLAO) permettent de transporter, respectivement, l'adresse de la couche liaison de l'émetteur dans les messages NS et l'adresse de la couche liaison du nœud cible (c.-à-d. celui qui émet le message) dans les messages NA. Cette option est nécessaire à la fonction de *résolution d'adresse* ;
- l'option *information du préfixe* (*Prefix Information*, PI) transporte un préfixe sous-réseau et indique la durée de validité et de "préférence" de ce dernier. Ainsi, lorsque cette option est jointe aux messages RA, elle permet aux hôtes utilisant l'*Autoconfiguration d'Adresse Sans État* pour configurer leurs adresses de maintenir celles-ci en état "préféré" ou au contraire de les placer en état "déprécié" ou "invalide" (cf. Fig. 2.1). L'option indique aussi si le préfixe annoncé est "sur le lien", c'est-à-dire que les hôtes apprenant ce préfixe pourront communiquer directement entre eux. Un drapeau permet également de désactiver l'*Autoconfiguration d'Adresse Sans État* pour ce préfixe. Cette option est nécessaire pour réaliser la *découverte de préfixes* et l'*Autoconfiguration d'Adresse Sans État* ;
- l'option *MTU* peut être jointe aux messages RA et permet de spécifier la valeur de la MTU quand celle-ci n'est pas connue sur le réseau. Cette option réalise la fonctionnalité de *découverte de paramètres*.

Ainsi, pour effectuer la procédure de *résolution d'adresse*, le nœud initiateur envoie un message NS contenant l'option *adresse couche liaison de la source* à destination de son correspondant. À la réception du message, celui-ci répond avec un message NA contenant l'option *adresse couche liaison de la cible* à destination de l'initiateur. À la fin de cet échange, les deux nœuds ont échangé leur adresse de couche liaison.

Un message RA comprenant une *option MTU* et une option *information du préfixe* à destination de l'adresse multicast "tous les nœuds" permettra d'informer les nœuds sur les préfixes en cours.

La fonctionnalité de *Détection d'Adresse Dupliquée* est, elle, réalisée par des messages NS. Le nœud dont l'adresse est dite "provisoire" joint l'adresse multicast "du nœud sollicité" et l'adresse multicast "tous les nœuds" et envoie un ou plusieurs messages NS (sans option) à destination de l'adresse multicast du nœud sollicité. Joindre les groupes multicast permet de recevoir les messages des autres procédures DAD en cours pour cette adresse d'une part et de recevoir les messages émis par un nœud qui posséderait déjà l'adresse d'autre part. Afin d'amorcer la procédure DAD, le nœud envoie un message NS avec pour adresse source l'adresse non-spécifiée (::) et comme adresse cible l'adresse multicast du nœud sollicité. Comme ce message ne fait pas partie d'une procédure de *résolution d'adresse*, aucune option de type *adresse couche liaison de la source* n'est jointe au message. Si une réponse de type NS ou NA est reçue, l'initiateur de la procédure conclut qu'il existe un autre nœud effectuant la procédure de DAD pour la même adresse (le message de réponse est de type NS) ou qu'un autre nœud utilise déjà l'adresse (le message est de type NA). La procédure DAD échoue et l'adresse n'est pas assignée à une interface. Quand la procédure DAD a été déclenchée à l'issue de l'*Autoconfiguration d'Adresse Sans État*, aucune autre tentative de création d'adresse n'est effectuée (puisque la partie identifiant d'interface de l'adresse est dérivée d'une valeur fixe). Si, en revanche, après un certain délai, aucun message n'est reçu, alors la procédure réussit et l'adresse est assignée à l'interface. Par défaut, la valeur de ce délai est fixée à une seconde.

2.5 Attaques sur le protocole de Découverte de Voisins

À l'origine, le protocole NDP ne dispose d'aucune forme de protection. Pourtant, une utilisation malicieuse des messages du NDP permet de réaliser de nombreuses attaques au niveau du lien. Cette situation a motivé un effort au sein de l'IETF pour sécuriser le protocole de Découverte de Voisins. Elle a, dans un premier temps, produit la rédaction du document [NKN04] qui recense les différentes attaques qui existent sur le NDP.

Ce document explique les modèles de confiance dans le NDP et catégorise les différentes attaques. Elles sont majoritairement réparties dans les attaques qui ciblent l'aspect "découverte de routeurs" (qui implique un nœud "routeur") et l'aspect "découverte de voisins" (où le rôle même du nœud n'a pas d'influence sur l'attaque). Bien que la majorité de ces attaques soit très simple à reproduire, elles suffisent à perturber fortement un réseau. En effet, le manque de protection, notamment cryptographique, sur les messages du NDP, autorise un attaquant à modifier, rejouer et même créer des messages NDP.

Ici, afin d'illustrer nos propos, tout en restant concis, nous décrivons seulement trois de ces attaques, laissant de côté d'autres attaques toutes aussi intéressantes, telles que l'usurpation de message Redirect.

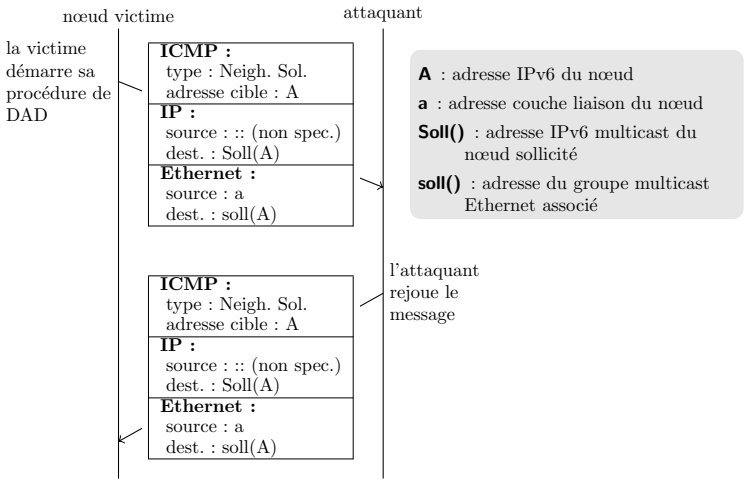


FIGURE 2.5: Attaque DoS sur la Détection d'Adresse Dupliquée

2.5.1 Attaque DoS sur la Détection d'Adresse Dupliquée

La Détection d'Adresse Dupliquée, nécessaire au bon fonctionnement de l'*Auto-configuration d'Adresse Sans État* et pour l'assignation d'une adresse en général, est vulnérable à une attaque de type Déni de Service (*Denial of Service*, DoS). En effet, comme nous l'avons vu dans les Sections 2.2 et 2.4, un nœud réalisant la procédure de DAD écoute les messages qui pourraient venir d'un autre nœud utilisant déjà l'adresse ou réalisant lui même un DAD sur cette adresse. La réception de l'un ou l'autre de ces messages suffit à faire échouer la procédure de DAD et donc l'assignation de l'adresse.

La tâche de l'attaquant consiste à écouter les messages des procédures DAD. Ces messages sont aisément identifiables, car ce sont des messages NS dont l'adresse source est l'adresse non spécifiée (::) et l'adresse destination est l'adresse multicast du nœud sollicité. Le champ *adresse de la cible* quant à lui informe l'attaquant sur l'adresse "provisoire" pour laquelle la victime effectue la procédure de DAD. Il ne reste plus, ensuite, à l'attaquant, qu'à envoyer, dans un délai d'une seconde (valeur par défaut), un message *Neighbor Advertisement* avec comme adresse source l'adresse "provisoire" de la victime et comme adresse destination l'adresse multicast "tous les nœuds". L'attaquant peut aussi simuler la présence d'un autre nœud effectuant une procédure DAD pour la même adresse. Pour se faire, l'attaquant rejoue le message NS reçu de la victime, comme illustré dans la Figure 2.5. Dans les deux cas, la procédure DAD de la victime échouera.

Si l'attaquant est présent sur un lien avant qu'un nouveau nœud ne joigne le réseau, il pourra potentiellement empêcher ce dernier d'obtenir une adresse, et donc de communiquer sur le réseau.

2.5.2 Suppression du routeur par défaut

L'attaque dite de "suppression du routeur par défaut" consiste à vider la liste de routeurs par défaut d'un ou plusieurs nœuds sur le lien. Quand cette liste devient vide sur un nœud, celui-ci ne peut plus envoyer de messages vers les destinations pour lesquelles il n'a aucune route.

L'attaquant dispose de plusieurs méthodes pour mener à bien cette attaque. La plus directe est de rendre injoignable le routeur par défaut (en le saturant avec une attaque de type Déni de Services, par exemple). Une approche plus élégante consiste à émettre de faux messages *Router Advertisement* dont le champ "durée de vie du routeur" est égal à 0. Cela indique au nœud que ce routeur n'offre pas/plus la fonction de routeur par défaut, et supprime alors l'entrée correspondante dans la liste de routeurs par défaut. Néanmoins, les messages RA du vrai routeur sont envoyés régulièrement et peuvent recréer des entrées dans la liste de routeur par défaut. Il suffit alors pour l'attaquant de continuer d'émettre de faux RA régulièrement également.

2.5.3 Usurpation de l'adresse d'un nœud

L'usurpation de l'adresse d'un nœud est une attaque qui permet de prendre le contrôle des communications d'une victime. Pour réaliser cette attaque, l'attaquant doit contrôler le contenu du cache de voisinage d'un nœud victime afin de changer l'association adresse couche liaison/adresse IPv6. Il a à sa disposition pour réaliser cette tâche deux messages : le message NS et le message NA. L'envoi de l'un ou l'autre des messages avec l'option "adresse couche liaison de la source" ou l'option "adresse couche liaison de la cible" correspondante permet d'écraser une entrée précédente (c.-à-d. légitime) contenue dans le cache de voisinage de la victime. Ainsi, l'attaquant écrase dans le cache de la victime l'adresse couche liaison de ses voisins pour y écrire sa propre adresse, lui assurant ainsi de recevoir les communications de la victime qui étaient auparavant destinées à l'adresse du cache écrasée.

Comme pour l'attaque précédente, l'attaquant doit veiller à envoyer ces messages régulièrement afin d'éviter qu'un message légitime d'un voisin ne mette à jour la victime.

2.6 Synthèse du chapitre

Ce chapitre a présenté le protocole de Découverte de Voisins et ses fonctionnalités. Nous avons montré que ce protocole revêt un aspect primordial pour les communications au dessus du protocole IPv6. Néanmoins, comme l'indique la Section 2.5, le manque de protection du protocole permet de nombreuses attaques. Plus encore, la majorité d'entre elles provoque une perte de service et certaines autorisent même l'usurpation de l'adresse d'un voisin. C'est à partir de ce constat que le travail sur le protocole de Découverte de Voisins Sécurisée a été lancé.

3 Le protocole SEND et les adresses CGA

Lors de la spécification initiale du protocole de Découverte de Voisins, le texte suggérait que l'utilisation d'IPsec pourrait assurer la sécurité du protocole. Par la suite, il est apparu que dans le cas de nœuds ne disposant pas encore d'adresse, il n'existe pas de solutions proposant la gestion de clés dynamiques afin d'établir une session IPsec, ce qui remet en question la sécurité du protocole de Découverte de Voisins.

La première étape pour sécuriser le protocole de Découverte de Voisins, présentée dans le chapitre 2, a été de référencer dans le document [NKN04] les différents vecteurs d'attaques du protocole de Découverte de Voisins. La seconde étape pour améliorer la sécurité du NDP a été réalisée en 2005, lorsque l'IETF a normalisé les Cryptographically Generated Addresses [Aur05] (CGA) et Secure Neighbor Discovery [AKZN05] (SEND, le protocole de Découverte de Voisins Sécurisée). Les CGA sont des *Crypto-Based Identifier* (CBID) basés sur le travail réalisé sur les *Statistically Unique and Cryptographically Verifiable* (SUCV) *Identifiers and Addresses* [NDS02]. Une utilisation conjointe d'une adresse CGA qui, par conception, est liée à une clé publique, et du protocole SEND, qui ajoute aux messages NDP une signature numérique calculée à partir de la clé privée associée, permet alors de prouver la possession de l'adresse, l'authenticité et l'intégrité des messages.

Dans ce chapitre, nous présentons le fonctionnement des adresses CGA (Section 3.1). Puis nous décrivons les mécanismes mis en place par le protocole SEND pour compléter la sécurité des adresses CGA (Sections 3.2, 3.3 et 3.4). Ensuite, nous présentons la fonctionnalité permettant la cohabitation entre les nœuds intégrant SEND et les nœuds ne proposant pas de sécurité (Section 3.5). Nous concluons en indiquant les limitations et vulnérabilités du protocole SEND (Section 3.6).

3.1 Adresses CGA

Le terme Cryptographically Generated Addresses désigne un mécanisme de génération d'adresse qui lie une clé publique et un ensemble de paramètres publiques à une adresse IPv6. Ce lien est réalisé par l'utilisation d'une fonction de hachage sur la clé publique et sur un ensemble de paramètres publiques pour obtenir la partie identifiant d'interface de l'adresse.

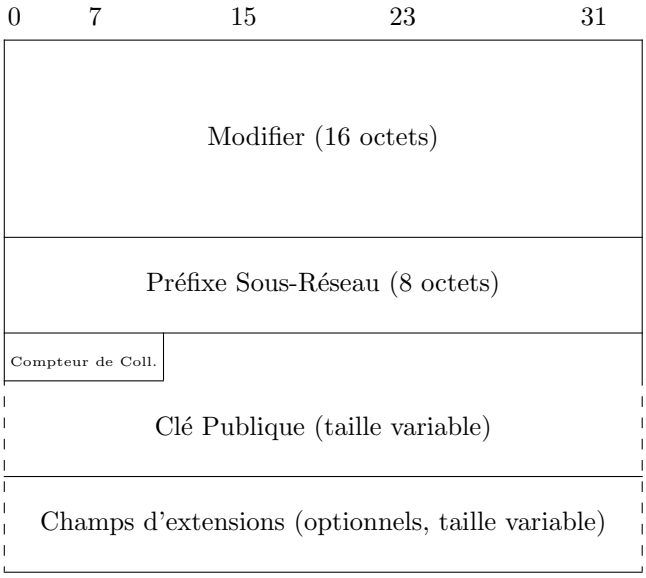


FIGURE 3.1: structure de données *CGA Parameters*

3.1.1 Génération de la CGA

Afin de générer une CGA, le nœud doit construire une structure de données nommée *CGA Parameters*. C'est sur cette structure de données que sera appelée par la suite la fonction de hachage. Cette structure de données, illustrée Figure 3.1 contient :

- un *modifier* : il s'agit d'un nombre aléatoire encodé sur 128 bits. Celui-ci permet d'augmenter la sécurité de l'adresse ainsi que l'aléa de l'adresse. Ainsi, deux adresses CGA générées à partir d'une même clé publique et d'un même préfixe sous-réseau seront différentes ;
- un *préfixe sous-réseau* recopie la partie préfixe sous-réseau de l'adresse (les 64 bits les plus à gauche). Ainsi, le condensât utilisé pour générer l'adresse CGA protégera également le préfixe sous-réseau. Par extension, la CGA protège alors l'adresse complète (le préfixe sous-réseau et l'identifiant d'interface) ;
- un *compteur de collisions*, sur 8 bits, encode le nombre de collisions détectées durant le DAD. Contrairement au mécanisme *Autoconfiguration d'Adresse Sans État* présenté dans le Chapitre 2, où l'identifiant d'interface est généré à partir d'une source statique (par ex. une adresse MAC) et où après une seule collision, le mécanisme de création d'adresse échoue, SEND permet de faire varier la valeur du champ *compteur de collisions* et ainsi de dériver de nouvelles adresses en cas de collision. Néanmoins, après trois collisions, la procédure de Détection d'Adresse Dupliquée échoue et un message est enregistré à destination de l'administrateur système (signalant un problème de configuration ou une attaque) ;

– le champ *clé publique* contient la clé publique du nœud. Cette clé publique, encodée au format DER, représente une structure ASN.1 de type *SubjectPublicKeyInfo* (telle que définie dans le document [HPFS02]). Dans la théorie, ce champ permet aux adresses CGA d'être indépendantes du type de clés car l'encodage DER permet de déterminer le format de clé. En pratique, comme nous le verrons Section 3.2, SEND impose l'utilisation de l'algorithme de chiffrement RSA ;
– les champs d'*extensions* permettent d'étendre les adresses CGA à de nouveaux usages. Le format général de ces extensions est précisé dans le document [BA06]. À notre connaissance, la seule utilisation de ces extensions est effectuée dans les *Hash Based Addresses* (HBA) utilisées par le protocole de *multihoming* Shim6 [Bag09] et [NB09].

La génération de l'adresse CGA consiste à calculer deux condensâts (ou *hash*), nommés *hash1* et *hash2*, sur différentes parties de la structure de données *CGA Parameters*. L'algorithme de génération d'une nouvelle adresse CGA, illustré par la Figure 3.2 et décrit dans le document [Aur05], est le suivant :

0. Bien que cette étape ne fasse pas partie intégrante de l'algorithme, la génération de CGA requiert qu'une paire de clés soit liée à la CGA. Cette paire peut avoir été générée au préalable ou recopiée d'une précédente adresse CGA ;
1. La valeur du *modifier* est générée aléatoirement ;
2. La fonction de hachage SHA-1 [Sta95] est appliquée sur la concaténation du *modifier*, neuf octets de 0, la clé publique encodée (au format DER) et les champs d'extensions (si présents). Les 112 bits les plus à gauche de la sortie de la fonction de hachage forment le condensât nommé *hash2*.
3. Un paramètre influant sur le niveau de sécurité de la CGA, nommé **SEC**, indique le nombre de zéro devant se trouver au début de la valeur *hash2*. Ainsi, si les $16 \times SEC$ premiers bits de *hash2* sont à 0, l'algorithme passe à l'étape 4. Sinon, la valeur du *modifier* est incrémentée de 1 et l'algorithme revient à l'étape 2. Les valeurs de SEC s'étendent de 0 (aucun test effectué) à 7 ;
4. Mettre les huit bits du *compteur de collisions* à 0 ;
5. La fonction de hachage SHA-1 est appliquée sur la concaténation de la valeur définitive du *modifier*, du *préfixe sous-réseau*, du *compteur de collisions*, de la clé publique encodée au format DER et de tous les champs extensions (si présents). Les 64 bits les plus à gauche du résultat forment un condensât nommé *hash1*.
6. L'identifiant d'interface est ensuite dérivé à partir de *hash1* en écrivant la valeur du paramètre SEC sur les trois bits les plus à gauche (bits 0, 1 et 2) et en plaçant les valeurs de "u" et "g" (sur les bits 6 et 7) à 0. Les bits "u" et "g" sont définis dans le document [HD06] et correspondent à la terminologie IEEE EUI-64, où "u" décrit la portée de l'adresse (locale ou universelle) et où "g" indique si l'adresse est une adresse de groupe (toujours à 0) ;
7. Les 64 bits du préfixe sous réseau et de l'identifiant d'interface sont concaténés pour former une adresse IPv6 ;
8. La procédure de Détection d'Adresse Dupliquée est effectuée pour l'adresse nouvellement générée. Si une collision est détectée, le *compteur de collisions* est

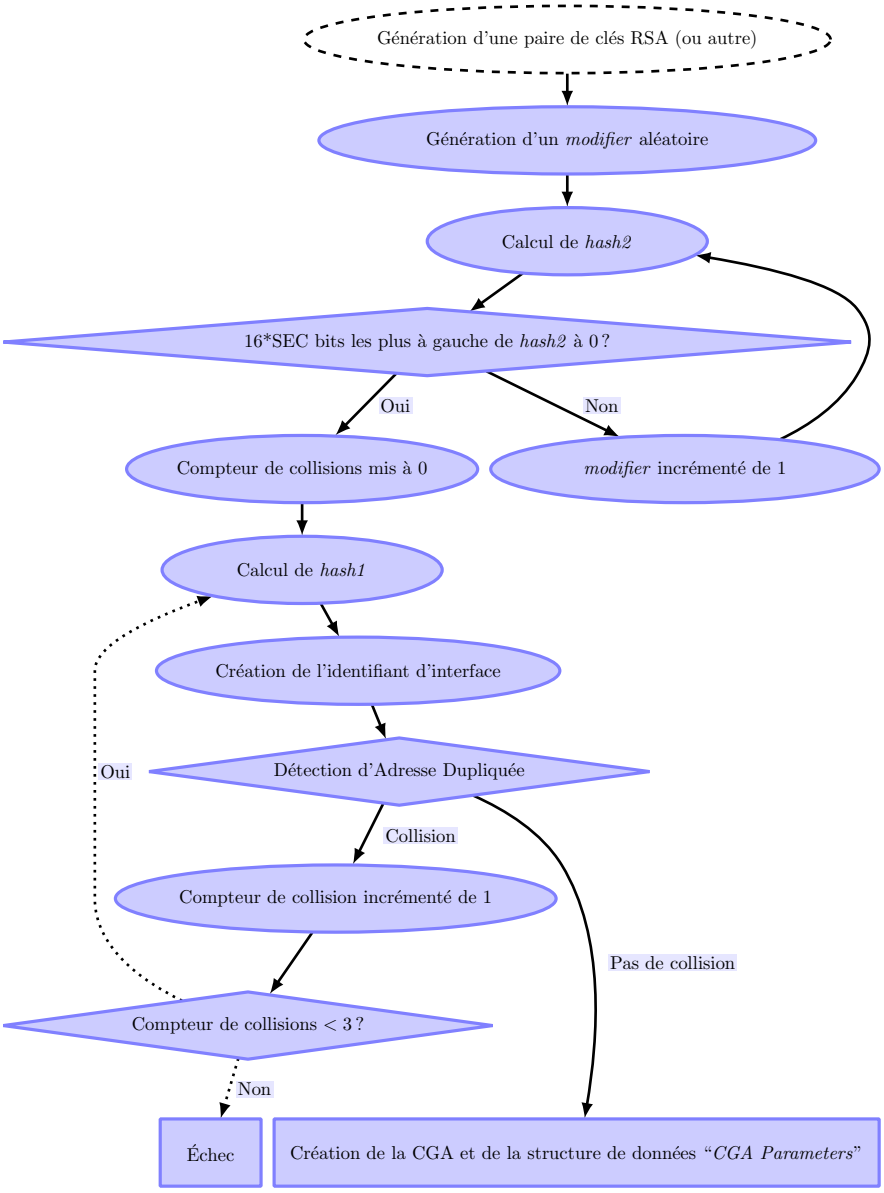


FIGURE 3.2: Algorithme de génération d'une adresse CGA

incrémenté de 1 et l'algorithme retourne à l'étape 5. Quand trois collisions successives se produisent, dues à un problème de configuration ou à une attaque, un message est enregistré à l'attention de l'administrateur et la procédure de génération d'adresse CGA échoue ;

9. La structure de données *CGA Parameters* est formée en concaténant la valeur définitive du *modifier* (appelée aussi *modifier* final), le *préfixe sous-réseau*, la valeur définitive du *compteur de collisions*, la clé publique encodée au format DER et les champs optionnels (si présents).

À la fin de l'algorithme, si tout s'est correctement déroulé, le nœud dispose d'une adresse CGA et de la structure de données *CGA Parameters* associée.

La boucle formée par les étapes 2-3, correspond à une couche de protection supplémentaire des CGA, nommée "extension de hachage". Cette extension est réalisée par le condensât *hash2* et le paramètre SEC, et augmente artificiellement la difficulté de génération d'une CGA. Ainsi, il faut en moyenne $2^{16\times SEC}$ itérations de la fonction SHA-1 avant de remplir la condition sur les premiers bits du *hash2*. Toutefois, il est à noter que cette ajout augmente la difficulté de création d'adresse à la fois pour l'attaquant et le nœud légitime. C'est pourquoi, les valeurs de SEC communément employées sont 0 (aucune protection supplémentaire) et 1. Les valeurs supérieures augmentent de manière trop sensible la durée de génération de la CGA (plusieurs heures sur une machine de bureau).

On notera que les condensâts *hash1* et *hash2* ne sont pas calculés sur les mêmes valeurs en entrée. Ainsi, le calcul de *hash2* n'inclue pas le préfixe sous-réseau. Ceci permet, dans un contexte de mobilité, où le préfixe sous-réseau est modifié, de n'avoir à recalculer que le *hash1* pour adapter l'adresse CGA au nouveau réseau, évitant ainsi le coûteux calcul de *hash2*.

3.1.2 Vérification d'une CGA

Pour pouvoir vérifier une adresse CGA, il est nécessaire de disposer de deux éléments : l'adresse CGA et la structure de données *CGA Parameters* associée. Ensuite, comme décrit dans le document [Aur05], il ne reste qu'à dérouler l'algorithme suivant :

1. Vérifier que le *compteur de collisions* de la structure de données *CGA Parameters* a une valeur comprise entre 0 et 2. La vérification échoue si ce critère n'est pas rempli ;
2. Vérifier que le préfixe sous-réseau contenu dans la structure de données *CGA Parameters* correspond au préfixe sous réseau de l'adresse CGA. La vérification échoue si les préfixes diffèrent ;
3. Exécuter la fonction de hachage SHA-1 sur la structure de données *CGA Parameters*. En prenant les 64 bits les plus à gauche (poids fort) du résultat de la fonction, on obtient *hash1* ;
4. Comparer *hash1* à la partie identifiant d'interface de l'adresse en omettant les trois premiers bits (paramètre SEC) et les bits 6 et 7 ("u" et "g") de la comparaison. Si la comparaison n'est pas satisfaite, la vérification échoue ;

5. Lire le paramètre de sécurité SEC à partir des trois bits les plus à gauche de la partie identifiant d'interface de l'adresse ;
6. Exécuter la fonction de hachage SHA-1 sur le *modifier*, 9 octets à 0, la clé publique encodée au format DER et les champs d'extensions (si présents). Les 112 bits les plus à gauche du résultat de l'appel forment le condensât *hash2*.
7. Vérifier que les $16 \times SEC$ premiers bits de *hash2* sont à 0. Si la condition n'est pas vérifiée, la vérification de la CGA échoue.

Si l'algorithme se déroule sans erreur, alors la CGA est valide.

Comme nous venons de le voir, afin de vérifier une CGA, il est nécessaire de posséder la structure de données *CGA Parameters* associée. À cet effet, SEND complète le mécanisme de génération d'adresse CGA en offrant le transport de la structure de données *CGA Parameters*.

3.2 Options et messages définis dans le protocole SEND

Les adresses CGA seules ne permettent pas de prouver la possession de l'adresse. Ce n'est que lors de l'utilisation de la clé privée associée à la CGA, par la génération d'une signature, que le nœud va prouver la possession de l'adresse. Toutefois, afin de pouvoir vérifier la CGA d'une part et la signature d'autre part, il est nécessaire que la clé publique et la structure de données *CGA Parameters* soient connues du destinataire. Pour se faire, la clé publique et l'ensemble des données publiques contenues dans la structure de données *CGA Parameters* sont stockés dans une nouvelle option NDP nommée *option CGA*.

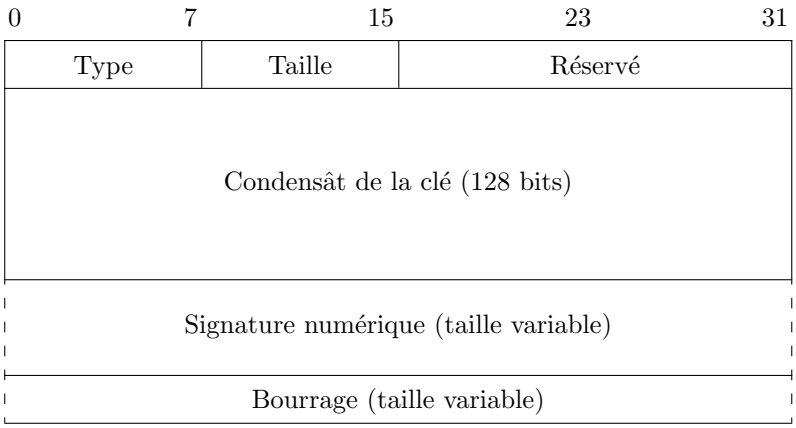


FIGURE 3.3: Format de l'option *Signature RSA*

Afin d'assurer la protection et l'authenticité des messages, SEND définit quatre nouvelles options NDP :

- l'*option CGA*, située en première position parmi les options, transporte la structure de données *CGA Parameters* (Figure 3.1). Cette structure de données est nécessaire pour la vérification d'une adresse CGA. Par ailleurs, la clé publique contenue dans cette structure permet aussi la vérification de la *signature numérique* contenue dans l'*option de Signature RSA*. Cette option est présente dès lors que le message est émis à partir d'une adresse CGA ;
- l'*option Signature RSA* (illustrée Figure 3.3), dernière option du message , contient une *signature numérique* du message et un *condensât de la clé* (publique) associé à cette signature. Ce condensât permet de déterminer quelle clé doit être utilisée pour vérifier la signature. En effet, le protocole SEND autorise, notamment pour les routeurs (comme présenté Section 3.4), la signature de messages en utilisant des clés publiques contenues dans des certificats X.509. Dans ce cas, l'émetteur possède la même clé publique dans son *option CGA* et dans son certificat et le récepteur doit avoir reçu et accepté le certificat pour vérifier le message.
  La signature numérique est réalisée par l'algorithme de chiffrement RSA et la fonction de hachage SHA-1, comme défini dans le document [RSA02]. Elle s'applique aux champs source et destinations de l'entête IPv6, aux champs *type*, *code* et *somme de contrôle* de l'entête ICMPv6 du message NDP, ainsi que sur toutes les options NDP jusqu'à l'*option Signature RSA*.
  Cette option doit être présente dans tous les messages protégés avec SEND ;
- l'*option Nonce* contient une valeur aléatoire codée sur plusieurs octets (au minimum 8 octets). Cette option accompagne toutes les requêtes (messages *Neighbor Solicitation* et *Router Solicitation*) afin de prévenir les rejeux de messages. Les nœuds désirant répondre à ces requêtes doivent alors recopier la valeur de cette option dans leur réponse ;
- l'*option Horodatage* attache au message la valeur de l'horloge du nœud. Elle indique alors l'heure à laquelle le message a été émis. Cette option est jointe à chaque message du NDP afin de prévenir le rejeu. Les différents nœuds doivent alors avoir une horloge synchronisée, bien qu'une différence de 10 minutes par défaut soit autorisée entre les horloges. Par la suite, cet écart est enregistré, de sorte que la vérification de l'horodatage soit plus fine.

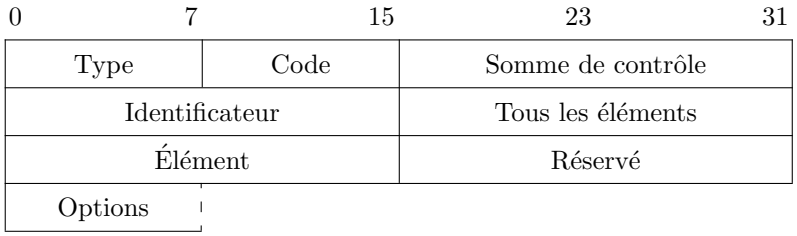


FIGURE 3.4: Format du message *Certificate Path Advertisement*

Une nouvelle fonctionnalité, nommée découverte du chemin de certification, a été introduite dans le protocole SEND. Cette fonctionnalité est réalisée par deux nouveaux

messages ICMPv6 : le message *Certificate Path Solicitation* (CPS) et le message *Certificate Path Advertisement* (CPA). Comme nous le verrons plus en détail Section 3.4, chaque hôte est déployé avec un ensemble d'"ancres de confiance", une liste de certificats en lesquels l'hôte a confiance. D'autre part, chaque routeur est configuré avec un certificat, pour lequel il existe un chemin jusqu'à une ancre de confiance, qui l'autorise à assurer la fonction de routeur.

Le message CPS est envoyé depuis un hôte à destination d'un routeur afin d'obtenir un chemin de certification vers une ancre de confiance connue de l'hôte. Une nouvelle option *ancre de confiance* permet de préciser la ou les ancres auxquelles l'hôte fait confiance, afin de restreindre la réponse du routeur.

Quand le routeur est capable de satisfaire la requête (ancre contenue dans l'option *ancre de confiance* de la requête connue), il envoie un message *Certificate Path Advertisement*. Le format de ce message est illustré Figure 3.4. Le message CPA indique, via une ou plusieurs options *certificat*, les certificats qui composent le chemin de certification. Quand ce chemin est divisé en plusieurs messages (car il comporte trop de certificats), les champs *élément* et *tous les éléments* indiquent respectivement quelle est la position du certificat dans le chemin et combien de certificats composent le chemin complet.

3.3 Utilisation du protocole SEND pour protéger les nœuds

Le protocole SEND utilise deux modèles de sécurité différents : celui décrit dans cette section est basé sur les adresses CGA. La Section 3.4 présente le mécanisme basé sur les certificats.

Les nœuds utilisant des CGA doivent faire usage de la clé privée associée pour prouver la possession de l'adresse. Ainsi, lorsqu'un nœud est "protégé" par SEND, il doit employer l'option de *signature RSA* dans ses messages NDP, afin de prouver l'origine du message. Comme cette signature est liée à une CGA, une *option CGA* est présente dans le même message et permet le transport de la structure de données *CGA Parameters*. Cette structure de données permet d'extraire la clé publique nécessaire à la vérification de la signature accompagnant le message.

Afin de vérifier l'authenticité d'un message, un récepteur devra vérifier la validité de l'adresse CGA (comme présenté Section 3.1.2), puis la "fraîcheur" des options anti-rejeu (*nonce* et *horodatage*) et finalement sera effectuée la coûteuse (comparée aux opérations précédentes) vérification de signature RSA. Si le message passe avec succès tous ces tests, alors l'authenticité de l'émetteur et l'intégrité du message sont prouvées. Par la suite, le message est traité comme un message NDP normal. Si l'un des tests échoue, le message est détruit.

Ainsi, une attaque par usurpation d'adresse, où une option *adresse couche liaison de la source* d'un message est modifiée, sera détectée par le récepteur. De plus, un attaquant ne peut plus forger de message à la place d'un nœud, car celui-ci est le seul à pouvoir prouver la possession de l'adresse en signant avec sa clé privée.

3.4 Utilisation du protocole SEND pour protéger les routeurs

Si l'utilisation conjointe des CGA et de SEND permet de s'assurer de l'authenticité et de l'intégrité des messages, elle ne permet pas de prouver que les informations diffusées par le nœud soient légitimes. En d'autres termes, le nœud lui-même peut émettre de fausses informations et les protéger. Quand le nœud est un hôte, les messages NDP émis ne contiennent que des informations le concernant (option *adresse couche liaison de la source*, etc.), l'influence de ces messages est alors moindre. Toutefois, un routeur dispose d'une autorité bien plus importante. Il avertit des préfixes sous-réseau, influençant ainsi les hôtes durant leur phase d'autoconfiguration, et peut potentiellement être choisi comme routeur par défaut. Ainsi, il est important que le protocole SEND permette au routeur de prouver son rôle.

Pour se faire, chaque routeur dispose d'un certificat qui l'autorise à agir comme tel. Ce certificat peut-être généré de manière directe ou indirecte par une ancre de confiance. Il est recommandé que ce certificat, de type X.509, contienne les extensions "*IP Addresses and AS Identifiers*" définies dans le document [LKS04]. En effet, ces extensions permettent au certificat de préciser pour quel espace d'adresse (et donc quel préfixe sous-réseau) il est autorisé à agir. Le routeur est ainsi contraint à annoncer dans ses messages *Router Advertisement* les préfixes sous-réseau contenus dans l'espace d'adresse imposé par son certificat. Cette protection limite les effets d'une prise de contrôle du routeur par un attaquant.

En complément, lors de son déploiement, chaque hôte est configuré avec une liste d'ancres de confiance. Un chemin de certification peut alors être établi entre le certificat du routeur et une ancre de confiance possédée par l'hôte. L'apprentissage de ce chemin de certification est réalisé par l'hôte à l'aide des messages *Certificate Path Solicitation* et *Certificate Path Advertisement* (présentés Section 3.2).

Afin d'illustrer l'enchaînement des différents mécanismes présentés, la Figure 3.5 illustre l'échange de message réalisé lorsqu'un hôte joint le lien. Ici, juste après son attachement au lien (détecté au niveau 2), l'hôte émet une requête RS destinée à l'adresse multicast "tous les routeurs". Cette requête lui permet de découvrir les routeurs présents sur le lien et de connaître les préfixes sous-réseau qu'ils avertissent. À la réception de cette requête, les routeurs vont émettre, s'ils ne l'ont pas fait récemment, un message RA. Cet échange de message fait partie de l'échange classique réalisé par le NDP. Toutefois, le nœud ne peut faire confiance au routeur avant d'avoir pris connaissance de son certificat. Le protocole SEND insère alors une autre étape afin de pouvoir établir un lien de confiance vers le routeur. L'hôte doit être capable de vérifier deux choses : que le routeur est autorisé à agir comme tel et que la signature du message qu'il vient de recevoir est bien valide. C'est pourquoi l'hôte émet un message CPS afin de découvrir le chemin de certification reliant le certificat du routeur à une ancre de confiance dont il est sûr. Si le ou les CPA émis par le routeur permettent à l'hôte de valider un chemin de certification pour ce routeur et que l'hôte vérifie la signature RSA, alors le message RA est traité normalement.

Une étape souvent omise, mais néanmoins importante, concerne la vérification des

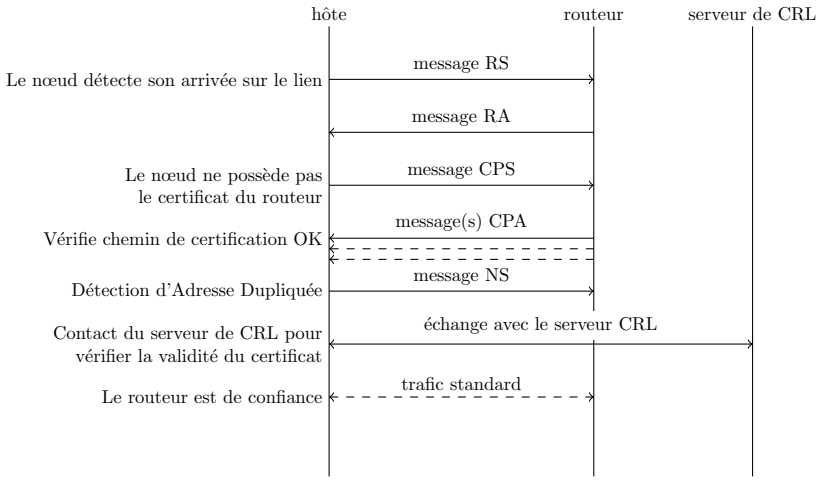


FIGURE 3.5: Échange de messages lors de l'arrivée d'un hôte sur un nouveau lien

listes de certificats révoqués (*Certificate Revocation List*, CRL). En effet, le certificat du routeur ne peut être accepté que de manière provisoire jusqu'à la consultation de ces listes. Si le certificat du routeur a été révoqué, alors les adresses générées à partir de son ou ses préfixes doivent être supprimées. Toutefois, la connexion à une CRL ne peut être réalisée que lorsque l'hôte dispose d'une adresse globale, ce qui oblige le nœud à effectuer ce test avant l'émission de tout autre paquet, juste après assignation de l'adresse sur son interface.

À noter que la spécification [AKZN05] est assez vague concernant l'utilisation de CGA dans les routeurs. Si le texte ne laisse aucun doute sur le fait que tous les nœuds sans exception doivent utiliser des CGA, d'autres parties de la spécification laissent supposer que l'usage du certificat seul sur les routeurs suffirait à prouver leur authenticité.

3.5 Phase de migration vers SEND et utilisation du mode mixte

Durant la phase de migration d'un réseau vers l'utilisation exclusive de SEND, les nœuds utilisant le NDP classique, appelés nœuds non-sécurisés, et ceux implémentant SEND, aussi appelés nœuds sécurisés, doivent cohabiter. À cet effet, SEND possède un élément de configuration lui permettant d'adopter un comportement permissif vis à vis des nœuds non-sécurisés. Ce comportement, nommé mode *mixte*, autorise les nœuds sécurisés à accepter les messages en provenance de nœuds non-sécurisés, tant que ceux-ci n'utilisent pas d'adresses déjà possédées par les nœuds sécurisés. Cette détection est effectuée auprès des entrées du cache de voisinage : les entrées créées par les nœuds

sécurisés ont un drapeau "sécurisé" à 1, les autres à 0. Ainsi, un nœud non-sécurisé ne peut pas écraser une entrée créée par un nœud sécurisé. Par ailleurs, une entrée dont le drapeau "sécurisé" est à 0 peut être mise à jour par un nœud sécurisé, faisant passer le drapeau à la valeur 1 et rendant les mises à jour par le nœud non-sécurisé précédent impossibles.

Il est à noter que l'utilisation du mode mixte ne protège pas les nœuds non-sécurisés et que ceux-ci restent vulnérables aux attaques listées dans [NKN04].

3.6 Analyse des limitations et faiblesses des CGA et du protocole SEND

Le protocole SEND se propose d'améliorer la sécurité de NDP. Cependant les couches de protection ajoutées sont parfois incompatibles avec les extensions du protocole de Découverte de Voisins. De plus, certains mécanismes mis en place sont sujets à des attaques. Cette section se propose de recenser ces limitations et faiblesses.

3.6.1 Incompatibilité de SEND avec la mobilité IP et proxy ND

Le protocole de mobilité, Mobile IPv6 [JPA04], permet à un nœud mobile quittant son réseau mère de continuer à recevoir son trafic existant. Pour se faire, une entité placée sur le réseau mère, nommée Agent Mère, réachemine le trafic à destination du nœud mobile au travers d'un tunnel (évitant ainsi les problèmes d'*ingress filtering* sur le réseau visité). Afin de s'assurer de réacheminer également le trafic qui est envoyé au nœud mobile depuis l'un de ses (anciens) voisins, l'agent mère agit comme s'il possédait l'adresse du nœud mobile. Ainsi, il envoie des messages *Neighbor Solicitation* et *Neighbor Advertisement* en utilisant l'adresse IPv6 du nœud mobile et en plaçant sa propre adresse de couche liaison dans les options. Cela lui permet de s'assurer de recevoir tout le trafic émis par les voisins du nœud mobile. Ce trafic sera par la suite réacheminé au nœud mobile via le tunnel.

Le document [TTP06], nommé *Neighbor Discovery Proxies*, décrit un mécanisme transparent permettant de partager un même préfixe sous-réseau à travers plusieurs liens. Cette fonctionnalité permet, entre autres, de réaliser des ponts (*bridge*) couche 3 quand la création de pont couche 2 est impossible (technologies des différents liens incompatibles, etc.). Dans ce cas, le proxy agit en plaçant son adresse couche 2 dans les options des messages du NDP qu'il reçoit avant de les retransmettre sur les autres liens. Ainsi, deux nœuds placés sur différents liens du même proxy souhaitant communiquer vont apprendre l'adresse couche liaison du proxy et lui envoyer par la suite les paquets. Le proxy se chargera ensuite de transmettre le paquet sur le lien où est connecté le correspondant.

Ces deux mécanismes proposent d'émettre ou de modifier des messages protégés par SEND. La première action implique une connaissance de la clé privée du nœud pour générer la signature RSA. La seconde action altère l'intégrité de la signature RSA du message. Ces deux points entraînent une incompatibilité avec le modèle de sécurité de SEND.

Ce sujet sera développé dans le Chapitre 7 où une solution y sera fournie.

3.6.2 Incompatibilité de SEND avec les adresses anycast

Dans le protocole de Découverte de Voisins, les adresses de type anycast autorisent le partage d'une adresse entre plusieurs nœuds sur un même lien.

Dans la pratique, c'est durant la phase de résolution d'adresse que l'association entre l'adresse et l'un des nœuds partageant l'adresse se fait. Les adresses anycast ne peuvent être utilisées pour initier une connexion. En d'autres termes, c'est le correspondant qui déclenche la procédure de résolution d'adresse. Chaque nœud, à l'issue d'un délai aléatoire après réception d'une requête *Neighbor Solicitation*, pourra émettre sa réponse. Le premier nœud à répondre crée une entrée dans le cache de voisinage de ce dernier. Il est à noter que le délai aléatoire permet d'éviter la congestion et de répartir l'usage de l'adresse anycast à travers les nœuds qui la partagent.

Ici, plusieurs nœuds partagent une même adresse. Pour les adresses CGA, cela implique que la paire de clés et la structure de données *CGA Parameters* doivent être connues de chaque nœud partageant l'adresse anycast. Le partage de la clé privée est délicat car il amoindrit la sécurité de l'adresse CGA et n'est donc pas souhaitable. Ce qui conduit à une incompatibilité entre SEND et le modèle d'adresse anycast.

Ce sujet sera développé dans le Chapitre 7 où une solution y sera fournie.

3.6.3 Vulnérabilité des CGA sur les adresses lien-local

Les CGA ont un identifiant d'interface dont seulement 59 bits sont variables lorsque la valeur du paramètre SEC est fixée.

Cela implique, en se basant sur le paradoxe des anniversaires, qu'un attaquant peut calculer $2^{59/2}$ adresses CGA à l'avance et s'assure ainsi de pouvoir trouver une collision avec un nouveau nœud joignant le réseau avec une probabilité de 50%. Ce calcul est long et présente généralement peu d'intérêt puisque l'identifiant d'interface est lié au préfixe sous-réseau et donc que l'ensemble des adresses calculées ne sont valides que pour un seul préfixe sous-réseau.

Toutefois, sur chacun des liens actifs, un nœud configure automatiquement une adresse lien-local pour communiquer directement avec son voisin. Par construction, ces adresses lien-local sont toutes formées à partir d'un préfixe unique et commun à tous les nœuds (`fe80::/64`).

Un attaquant qui générerait $2^{59/2}$ adresses CGA serait alors capable d'usurper l'adresse de ses voisins. Ce scénario devient malheureusement possible, avec l'augmentation de la capacité des disques.

Une solution pour parer cette attaque consiste à augmenter la valeur du paramètre SEC minimale autorisée dans un réseau. Cela permet d'augmenter la durée de génération des adresses CGA, et ralentit alors le travail de l'attaquant dans la génération des adresses. Néanmoins, cette solution n'est que peu viable sur le long terme, puisque le surcoût de calcul est propagé sur les nœuds légitimes.

À noter qu'il s'agit là d'une attaque qui fonctionne sur les adresses CGA indépendamment de la fonction de hachage utilisée. La Section 3.6.5 présente les vulnérabilités qui

sont liées à la fonction de hachage utilisée pour construire les adresses CGA : SHA-1.

3.6.4 Dénis de services liés à l'utilisation de fonctions cryptographiques

SEND emploie différentes fonctions cryptographiques afin de prouver l'authenticité de la source du message. Ainsi, la vérification d'un message NDP protégé par SEND implique la vérification d'une adresse CGA, c'est à dire deux calculs de condensâts, et la vérification d'une signature RSA. Un attaquant envoyant de nombreux messages NDP à un nœud pourrait alors saturer ses ressources de calcul avec le calcul de condensâts et la vérification de signatures RSA.

Toutefois, la mise en place d'un mécanisme de *rate-limiting*, limitant le nombre de messages NDP générés par nœud et par seconde, peut être mis en place. En effet, lorsque trop de requêtes *Router Solicitation* sont envoyées à un routeur, celui-ci ne répond plus à chaque requête individuellement mais émet de manière régulière des messages de type *Router Advertisement* destinés à l'adresse multicast "tous les nœuds". De la même façon, un nœud recevant au delà d'un certain seuil de requêtes *Neighbor Solicitation* se mettra à diffuser une requête *Neighbor Advertisement* régulièrement.

3.6.5 Vulnérabilité de la fonction de hachage SHA-1

À l'issue de la cryptanalyse de la fonction SHA-1 [Sta95], il a été découvert qu'une attaque de type collision avec une complexité de 2^{63} était réalisable [Coc07].

Cette fonction de hachage est présente dans de nombreux composants du protocole SEND : lors de la génération et vérification de l'adresse CGA, dans les champs *condensât de la clé* et *signature numérique* de l'*option RSA* ou bien encore dans les certificats X.509 déployés sur les nœuds. L'analyse de l'impact de cette attaque sur les CGA et protocole SEND est effectuée dans les documents [BA07] et [KKJ10] respectivement.

La seule menace répertoriée provient d'un nœud qui pourrait générer deux adresses CGA identiques et par la suite pourrait utiliser l'une ou l'autre. Toutefois, SEND ne garantit pas la non-répudiation, et donc il ne s'agit pas là d'une réelle menace.

Une attaque sur la première pré-image [1] (*first preimage attack*) aurait des conséquences plus graves et rendrait possibles les collisions volontaires sur des adresses existantes, rendant ainsi les protections du protocole SEND inopérantes. En prévision de cette attaque, le document [BA07] propose un mécanisme permettant d'utiliser des fonctions de hachage alternatives pour la génération d'adresses CGA.

3.6.6 Attaque de type DoS sur la procédure DAD

Cette attaque fait partie de nos contributions et a été présentée plus en détail à la conférence *Sécurité des Architectures Réseaux et des Systèmes d'Information* (SAR-SSI) 2008 [CC08].

1. Attaque qui consiste à trouver un message qui a le même condensât qu'un message donné.

Il s'agit là d'une attaque déjà existante sur le protocole de Découverte de Voisins qui a été relevée dans le document [NKN04] (présentée Section 2.5.1). Le protocole de Découverte de Voisins Sécurisée doit prévenir les attaques de type rejeu et fournit à ce titre deux options (*nonce* et *horodatage*). Cependant, l'option *nonce* lors de la procédure de Détection d'Adresse Dupliquée ne permet pas de détecter de rejeu. En effet, l'option n'est utilisée que pour s'assurer de la "fraîcheur" d'une réponse et aucune sémantique ne lui est attribuée durant la procédure de DAD. Ainsi, un attaquant n'a qu'à rejouer le message NS émis par un nœud effectuant la procédure DAD pour que celui-ci confonde le message rejoué avec un message légitime. En effet, la victime interprète son propre message (rejoué) comme une message envoyé par un nœud effectuant la procédure DAD pour la même adresse.

Dans la pratique, à la réception du message, la victime va vérifier l'adresse CGA, qui est la sienne, et donc valide. L'option *horodatage* sera valide, car dans le delta de 10 minutes, puisque le paquet a été rejoué en quelques millisecondes par l'attaquant. L'option *nonce* n'est pas analysée pour les messages de type NS. L'option de *signature RSA* est elle aussi valide. Le message rejoué semble donc tout à fait légitime et est traité comme tel.

Deux parades ont été proposées et sont simples à implémenter :

- après la génération de trois adresses et la détection de trois collisions : assigner l'adresse. La probabilité d'obtenir trois collisions d'adresse successives est négligeable et cette solution est donc viable ;
- donner une sémantique à l'option *nonce* durant le DAD et ignorer les messages NS qui portent la même valeur de nonce durant la procédure de DAD.

Ce dernier choix est le moins coûteux et sera donc privilégié.

Depuis l'écriture de l'article [CC08], cette attaque a été testée avec succès sur des routeurs Cisco et a été notifiée au groupe de travail chargé de la maintenance des CGA et du protocole SEND au sein de l'IETF.

3.7 Synthèse du chapitre

Ce chapitre présente le protocole de Découverte de Voisins Sécurisée et les mécanismes de protection des messages du protocole de Découverte de Voisins (CGA, signature RSA, option d'anti-rejeux, ...). Nous avons vu que le protocole SEND permet, d'une part, à un nœud de prouver la possession de son adresse et, d'autre part, à un routeur de prouver qu'il est autorisé à agir comme tel.

Nous avons également présenté les limitations du protocole ainsi que diverses attaques et parades. Parmi les attaques, nous avons décrit l'une de nos contributions.

4 Influence des algorithmes cryptographiques sur les CGA et le protocole SEND : mesures de performance

Le chapitre précédent montre la forte dépendance des adresses CGA et du protocole SEND vis à vis de l'algorithme de chiffrement RSA. Toutefois, l'utilisation de l'algorithme RSA limite le déploiement de ces protections dans les environnements mobiles, où l'énergie et la capacité de stockage sont limitées.

Dans ce chapitre, nous présentons les résultats d'une étude de performances des adresses CGA et du protocole SEND. Dans les Sections 4.1 et 4.2, nous étudions les gains en performances des CGA et du protocole SEND en remplaçant l'algorithme de chiffrement RSA par les courbes elliptiques (ECC, *Elliptic Curve Cryptography*) et ECDSA (*Elliptic Curve DSA*). Puis nous évaluons l'impact de la fonction de hachage sur la génération d'adresses CGA afin d'anticiper l'arrêt de l'utilisation de SHA-1 (Section 4.3). Dans une dernière partie, nous présentons une approche basée sur l'utilisation de processeurs graphiques et des méthodes de calcul GPGPU (*General-Purpose Graphical Processing Units*) pour améliorer la vitesse de génération des adresses CGA (Section 4.4).

Ce chapitre fait partie de nos contributions et a été publié dans le journal *Computers & Security* [CBL10] en 2010. Des relevés de mesures plus complets sont également fournis dans le rapport de recherche associé [BCLM08]. Ces mesures ont été réalisées par Aymen Boudguiga à l'occasion d'un stage de recherche effectué au sein de Télécom SudParis.

4.1 Intégration d'ECC/ECDSA dans SEND

La cryptographie basée sur les courbes elliptiques [HMV04] permet d'introduire la sécurité dans les réseaux où les nœuds mobiles ont de faibles ressources (de calcul, de stockage et de batterie). Cela est particulièrement intéressant pour les réseaux de capteurs et les réseaux Ad hoc. Dans ces réseaux, l'utilisation d'ECC/ECDSA offre les avantages suivants :

- ECDSA fournit un calcul rapide de la signature ;
- la taille du code utilisé est petite, et convient plus facilement aux équipements à faibles capacités ;

– les clés ECC sont plus petites que les clés RSA pour un niveau de sécurité équivalent. Cette particularité est surtout intéressante dans SEND puisque la clé publique est transportée dans chaque message protégé. Ainsi, une clé publique plus petite implique un consommation de bande passante et d'énergie (pour les transmissions) moindres. Le tableau 4.1, extrait du document [Nat07], indique la correspondance entre les tailles de clés RSA et ECC pour un niveau de sécurité comparable.

Nous avons alors effectué de minimes modifications au sein du protocole SEND pour implémenter et utiliser ECC et ECDSA. Voici les modifications que nous proposons :

– remplacer la signature RSA par une signature ECDSA dans SEND [AKZN05]. Le format de l'option *signature RSA* reste inchangé (même si elle contient une signature ECDSA) ;
– changer la clé RSA encodée au format DER dans la structure de données *CGA Parameters* (Figure 3.1) pour y placer une clé ECC encodée au format DER.

Le type de clé publique contenu dans la structure de données *CGA Parameters* indique quel algorithme doit être employé pour vérifier la signature contenue dans l'option *signature RSA*.

Nous proposons, dans nos travaux présentés dans le Chapitre 5, une extension du protocole SEND qui permet d'être indépendant du type d'algorithme utilisé tout en restant interopérable avec les anciens équipements. Les résultats de l'analyse de performance réalisée dans ce chapitre sont applicables à l'extension que nous proposons.

Taille de clé RSA (en bits)	Taille de clé ECC (en bits)
1024	163
2048	224
3072	256
7680	384
15360	512

Tableau 4.1: Équivalence entre les tailles de clés RSA et ECC (pour un niveau de sécurité équivalent) [Nat07]

4.2 Utilisation des courbes elliptiques avec les CGA et le protocole SEND

4.2.1 Méthodologie

Description des valeurs mesurées

Afin d'étudier les avantages et inconvénients à l'utilisation des courbes elliptiques pour remplacer RSA, nous évaluons les durées suivantes :

- la *durée de génération (totale) de la CGA* exprime la durée complète de la génération de l'adresse CGA, depuis la création de la paire de clés jusqu'au calcul de l'identifiant d'interface, en incluant le calcul de *hash1* et *hash2*. C'est le travail qu'effectue un nœud lorsqu'il joint le réseau et qu'il n'a pas sauvegardé de paire de clés par le passé ;
- la *durée de la vérification de la CGA* correspond au temps écoulé pour effectuer les différentes étapes de la vérification de l'adresse CGA (comme indiqué dans la Section 3.1.2), en testant la structure de données *CGA Parameters*. Il est à noter que la vérification de la signature RSA/ECDSA n'est pas inclue dans cette durée ;
- le *calcul du modifier final* correspond au temps écoulé pour trouver la valeur de *hash2* qui remplit la condition sur ses $16 \times SEC$ premiers bits. Par rappel, le calcul de *hash2* n'est effectué que lors de la génération de l'adresse CGA, alors que *hash1* est recalculé pour adapter l'adresse CGA à un nouveau préfixe sous-réseau. Comme nous le verrons, il a une influence très forte sur le *temps de génération de la CGA* ;
- la *durée de génération des clés* exprime le temps écoulé pour générer une paire de clés RSA ou ECC ;
- la *durée de la génération de la signature RSA/ECDSA* exprime le temps nécessaire pour calculer la signature (RSA ou ECDSA). Cette durée correspond au temps de génération du champ *signature numérique* de l'option *signature RSA* dans le protocole SEND ;
- la *durée de la vérification de la signature RSA/ECDSA* indique le temps nécessaire à la vérification de la signature (RSA ou ECDSA). Cette durée correspond au temps requis pour vérifier la signature contenue dans le champ *signature numérique* de l'option *signature RSA*.

Pour nos tests, les clés publiques RSA que nous générons ont un exposant publique égal à 3, ce qui permet une génération de signature plus rapide.

Il est à noter que nous souhaitions utiliser des clés ECC de taille 512 bits, équivalentes à des clés RSA de 15360 bits. Toutefois, la bibliothèque OpenSSL [1], dont notre outil dépend, ne propose pas cette taille de clé. Nous avons alors choisi une taille de clé supérieure disponible, 571 bits, et avons désigné dans nos tableaux la clé RSA d'un niveau de sécurité équivalent par le terme "15360+".

Méthodologie d'échantillonnage

Nous avons d'abord utilisé un estimateur de Monte Carlo pour déterminer le nombre d'échantillons nécessaire. Toutefois, le processus de génération des CGA est un processus aléatoire (principalement à cause du calcul du *modifier* final) et l'estimateur n'a pas été en mesure de retourner de résultats probants.

Nous avons alors fixé le nombre d'échantillons pour chaque mesure à 10 000 et à la suite de nos mesures nous nous sommes assurés que la variance des échantillons était assez faible pour confirmer que le nombre d'échantillons choisis était suffisant.

1. http://openssl.org

Méthode pour mesurer les durées d'exécution

Nous avons effectué les mesures sur un système Linux démarré en mode *single*. Dans ce mode, aucun processus à part le shell et notre programme de mesures n'est exécuté. Ce choix limite au maximum les interactions du système et de l'ordonnanceur sur nos relevés de valeurs. Nous avons ensuite placé l'instruction assembleur RDTSC (*Read Timestamp Counter*) avant et après chaque morceau de code à évaluer. Cette instruction retourne le nombre de cycles d'horloge écoulés sur le processeur au moment de son appel. Ainsi, entre deux exécutions de l'instruction, il est possible de déterminer le temps écoulé. Il suffit de diviser le nombre de cycles d'horloge écoulés par la fréquence du processeur pour obtenir le temps écoulé entre les deux appels.

Le système que nous utilisons, basé sur un Pentium 4, ne dispose pas de mécanisme de réduction de fréquence et est mono-cœur. Il est donc parfaitement adapté à l'appel de l'instruction RDTSC pour effectuer des mesures de temps. Ces mesures seraient plus délicates sur un système plus récent, où la concurrence et les fréquences variables rendent l'utilisation de l'instruction RDTSC plus contraignante.

4.2.2 Comparaison entre l'utilisation de RSA et d'ECC pour la génération d'adresses CGA sur un PC

Nous souhaitons évaluer l'impact de l'utilisation des adresses CGA dans un environnement réel. À cet effet, nous évaluons les durées de génération et de vérification des adresses CGA sur un PC de moyenne gamme. Nous nous intéressons également au lien entre la taille de la clé publique et la durée d'exécution de la fonction de hachage SHA-1 lors de la génération et la vérification d'adresses CGA.

Génération d'une adresse CGA

Cette section compare la durée de génération d'une CGA basée sur une clé RSA et d'une CGA basée sur une clé ECC quand la valeur du paramètre SEC est égale à 0 et à 1. Le Tableau 4.2 et la Figure 4.1 donnent les valeurs moyennes, calculées sur 10 000 échantillons, obtenues sur un Pentium 4 cadencé à 2593 MHz.

Le Tableau 4.2 montre que la durée de génération d'une CGA basée sur une clé ECC de 571 bits avec un paramètre SEC à 1 est plus courte que la durée de génération d'une CGA basée sur une clé RSA de 2048 bits avec un paramètre SEC à 0. De ces valeurs, nous pouvons affirmer qu'ECC permet un meilleur niveau de sécurité tout en diminuant la durée de génération d'une CGA.

Quand la valeur du paramètre SEC est égale à 0, la durée de génération de la CGA représente seulement le temps de calcul de la clé et du condensât *hash1*. Comme prévu, le temps de génération pour une valeur du paramètre SEC égale à 1 est supérieur au temps de génération pour une valeur de paramètre SEC à 0. Cette différence est causée par le temps supplémentaire nécessaire au calcul du *modifier* final, pour que les $16 \times SEC$ premiers bits du condensât *hash2* soient à 0. Ainsi, il faut en moyenne 2^{16} calculs de condensâts avant de trouver la valeur du *modifier* final. Cette valeur est calculée à partir de la Formule 4.1, qui donne le nombre d'essais moyen nécessaires pour

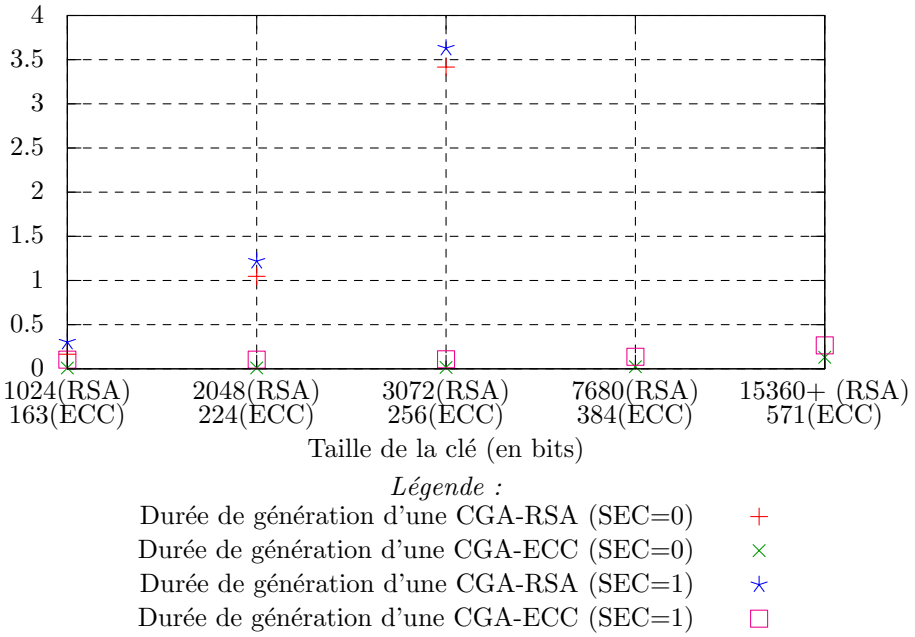


FIGURE 4.1: Comparaison entre les vitesses de génération d'adresses CGA avec RSA et ECC sur un Pentium 4 à 2593 MHz (en secondes)

Valeur de SEC	0				
Taille de clé RSA (bits)	1024	2048	3072	7680	15360+
Taille de clé ECC équiv. (bits)	163	224	256	384	571
Durée de génération CGA-RSA	0.165630	1.047319	3.415941	91.562634	-
Durée de génération CGA-ECC	0.009535	0.010031	0.015732	0.023823	0.129787
Valeur de SEC	1				
Durée de génération CGA-RSA	0.302924	1.218577	3.631235	91.957422	-
Durée de génération CGA-ECC	0.103472	0.103276	0.108027	0.135732	0.265401
Calcul du *modifier* final avec RSA	0.137296	0.171260	0.215295	0.394790	-
Calcul du *modifier* final avec ECC	0.093938	0.093247	0.092297	0.111910	0.135616

TABLEAU 4.2: Durée de génération d'une CGA et du *modifier* final en utilisant des clés RSA et ECC sur un Pentium 4 à 2593 MHz (en secondes)

obtenir la valeur du *modifier* final en fonction du paramètre SEC. Cette formule est adaptée à partir de la formule de complexité d'une attaque sur la première pré-image d'une fonction de hachage [MOVR97].

$$f(SEC) = 2^{16 \times SEC} \tag{4.1}$$

De la Formule 4.1, nous montrons également que la génération d'une CGA dont le paramètre SEC est supérieur ou égal à 2 n'est pas calculable en un temps convenable. Un test non représentatif avec 10 échantillons nous donne un aperçu du temps nécessaire. Ainsi, pour générer une CGA basée sur une clé RSA de 1024 bits avec une valeur de paramètre SEC égale à 2, il faut en moyenne 1.8 heures. Ce résultat est confirmé par la Formule 4.1 qui permet d'extraire un facteur multiplicatif de 2^{16} entre la durée de calcul du *modifier* final avec SEC égal à 1 et avec SEC égal à 2. Si nous prenons 0.137 seconde comme valeur moyenne de la durée de calcul du *modifier* final d'une CGA basée sur une clé RSA de 1024 bits (à partir du Tableau 4.2), alors nous pouvons extrapoler qu'il faut environ 2.5 heures pour générer une adresse (estimation confirmée par les 1.8 heures recueillies par le test préliminaire), ce qui n'est pas acceptable pour un usage réel.

Le Tableau 4.2 indique également que la durée de génération du *modifier* final est plus courte avec les clés ECC qu'avec les clés RSA. Cela provient de la durée de calcul de *hash2* qui est plus courte avec ECC puisque la fonction de hachage SHA-1 est appelée sur une structure de données *CGA Parameters* dont la taille varie (très fortement) en fonction de la taille de la clé publique (comme expliqué dans la section suivante).

Analyse théorique de l'impact de la fonction de hachage SHA-1 sur les performances

Cette section propose l'analyse de l'impact de la fonction de hachage SHA-1 (requise par les spécifications des CGA [Aur05]) sur les temps de génération de CGA fournis dans le Tableau 4.2. Comme nous l'avons dit précédemment, deux condensâts *hash1* et *hash2* doivent être calculés lors de la génération de la CGA. Afin de mieux comprendre l'importance de la fonction SHA-1 dans la durée de la génération de la CGA, quand une paire de clés RSA ou ECC sont utilisées, nous analysons le fonctionnement interne de SHA-1.

L'algorithme de la fonction SHA-1 est divisé en deux phases majeures : le pré-traitement des données et le calcul du condensât [Nat08b]. La phase de pré-traitement commence par une opération d'alignement afin d'ajouter des octets de bourrage si nécessaire sur le message en entrée, puis ce message est divisé en blocs de 512 bits et finalement les valeurs d'initialisation du condensât sont placées dans les différentes variables de la fonction. Supposons que le message a été divisé en N blocs. Le calcul du hash est alors effectué sur chaque bloc et est composé de 4 tours (ou *rounds*), chacun contenant 20 étapes. Ce temps de calcul de condensât est constant pour tous les blocs, ce qui nous permet d'inférer que la complexité de ce calcul est liée au nombre de blocs et peut donc être estimée en $O(N)$.

Taille de clé RSA (bits)	1024	2048	3072	7680	15360
Taille de clé pub. RSA encodée au format DER (octets)	160	292	420	996	1956
Taille de la structure de données *CGA Parameters* (bits)	1480	2536	3560	8168	15848
Nombre de blocs de 512 bits	4	6	8	17	32
Taille de clé ECC équivalente (bits)	163	224	256	384	571
Taille de clé pub. ECC encodée au format octet (octets)	66	80	88	120	170
Taille de la structure de données *CGA Parameters* (bits)	728	840	904	1160	1560
Nombre de blocs de 512 bits	2	2	2	3	4

TABLEAU 4.3: Taille de la structure de données *CGA Parameters* et correspondance en nombre de blocs pour la fonction de hachage SHA-1

Dans le contexte des CGA, SHA-1 s'applique sur la structure de données *CGA Parameters*, et par conséquent la taille de cette structure influence directement le temps de calcul des condensâts. Quand il n'y a aucun champ *extensions*, tous les champs de la structure de données *CGA Parameters* excepté le champ *clé publique* ont une taille fixe. À cet effet, le Tableau 4.3 illustre le lien entre la taille de cette structure de données et le nombre de blocs de 512 bits nécessaire, en fonction du type et de la taille de clé utilisés.

À partir du Tableau 4.3, nous déduisons que les temps de calcul de *hash1* et *hash2* tombent dans le même intervalle de valeurs pour une clé RSA de taille 1024 bits et pour une clé ECC de tailles 384 ou 571 bits. La même remarque s'applique également pour les clés ECC de tailles 163, 224 et 256 bits qui sont toutes trois réparties sur deux blocs. Ces remarques se vérifient dans le Tableau 4.2. Il est important de noter qu'en suivant cette logique, l'on peut augmenter le niveau de sécurité d'une CGA en privilégiant les tailles de clés plus grandes tant que celles-ci ne forcent pas à exécuter la fonction SHA-1 sur un nombre plus grand de blocs.

Il apparaît que les temps de calcul des condensâts *hash1* et *hash2* sont souvent plus grands avec les clés RSA qu'avec les clés ECC puisque les clés RSA génèrent plus de blocs que leurs contreparties ECC (surtout lorsque la taille de clé RSA excède les 2048 bits). Cette même remarque s'applique également lors de la procédure de vérification de CGA, comme nous le montrons dans la section suivante.

Vérification des adresses CGA

La vérification de l'adresse CGA est la première étape effectuée dans le protocole SEND après la réception d'un message ND émis depuis une adresse CGA. Il s'agit là d'une vérification légère et rapidement effectuée comparée à la vérification de la signature RSA. La phase la plus nécessiteuse en ressource dans l'algorithme de vérification, comme nous l'avons vu Section 3.1.2, est composée du calcul des deux condensâts *hash1* et *hash2*. Les autres étapes sont seulement des comparaisons rapides avec ces condensâts. Le Tableau 4.4 illustre ces résultats.

D'une manière analogue à la génération de l'adresse CGA, où ECC fournit des clés

Valeur de SEC	0				
Taille de clé RSA (bits)	1024	2048	3072	7680	15360+
Taille de clé ECC équiv. (bits)	163	224	256	384	571
Durée de vérification CGA-RSA	0.000004	0.000005	0.000006	0.000009	-
Durée de vérification CGA-ECC	0.000004	0.000004	0.000005	0.000005	0.000005
Valeur de SEC	1				
Durée de vérification CGA-RSA	0.000005	0.000006	0.000007	0.000013	-
Durée de vérification CGA-ECC	0.000003	0.000003	0.000003	0.000004	0.000005

Tableau 4.4: Durée de vérification d'une CGA sur un Pentium 4 cadencé à 2593 MHz (en secondes)

plus petites que RSA, ECC se comporte avec de meilleures performances. Celles-ci sont à attribuer au calcul de condensât qui s'effectue sur des structures de données *CGA Parameters* plus petites, comme nous l'avons vu dans la section précédente.

4.2.3 Comparaison entre l'utilisation de RSA et ECDSA dans SEND pour la génération et vérification de signatures sur un PC

Cette section présente les résultats relatifs à la génération et à la vérification de signatures RSA et ECDSA dans le protocole SEND. Pour nos tests, nous avons simulé un scénario SEND où un nœud crée un message *Neighbor Solicitation*, en générant un message aléatoire avec toutes les options SEND (option de *signature RSA* omise). La taille de ce message varie en fonction de la taille de la clé publique afin de s'accorder au cas réel. Ainsi, les message résultants, que nous utilisons pour nos tests sur la signature, ont alors la même taille que ceux qui seraient émis par un nœud implémentant le protocole SEND [AKZN05]. Ceux-ci nous permettent d'effectuer des mesures précises sur les opérations cryptographiques réalisées sur les (vrais) messages générés par le protocole SEND.

Taille de clé RSA (bits)	1024	2048	3072	7680	15360+
Taille de clé ECC équiv. (bits)	163	224	256	384	571
Durée de gén. de signature RSA	0.004568	0.022275	0.053676	0.609052	-
Durée de gén. de signature ECDSA	0.002219	0.002588	0.004439	0.007338	0.042402
Durée de vérif. de la signature RSA	0.000069	0.000165	0.000321	0.001422	-
Durée de vérif. de la sig. ECDSA	0.004398	0.003039	0.005352	0.008775	0.084884

Tableau 4.5: Durée de génération et vérification de signatures RSA et ECDSA sur un Pentium 4 cadencé à 2593 Mhz (en secondes)

Le Tableau 4.5 montre que la durée de génération d'une signature RSA augmente avec la taille de la clé. C'est un résultat prévisible, puisque le calcul de la signa-

ture dépend de la taille du modulo. Ce tableau indique également que le temps de vérification d'une signature ECDSA est toujours plus long que son temps de génération. À cet effet, nos résultats expérimentaux confirment le comportement attendu de la génération de signature ECDSA [GGCS02]. Nous notons également que si la clé est de type RSA et que sa taille dépasse les 7680 bits, la vitesse de génération des signatures passe au dessus des 0.6 secondes. Cette durée trop élevée nuit à un fonctionnement correct du protocole NDP qui impose des délais précis. En effet, si l'une des valeurs de génération et de vérification de signatures excède les 0.5 secondes, la procédure de Détection d'Adresse Dupliquée devient inopérante. Plus précisément, dans le cas d'une collision d'adresses, le nouveau nœud configurant l'adresse en cours d'utilisation va émettre un message NS. Le nœud utilisant déjà l'adresse y répond par un message NA. Ainsi, ils vont chacun être amenés à générer et à vérifier deux messages du NDP en moins d'une seconde (comme indiqué dans [TNJ07]), ce qui implique deux générations et deux vérifications de signatures. C'est pourquoi, quand la durée de génération ou vérification de signatures dépasse les 0.5 secondes, les nœuds ne peuvent procéder à temps à l'échange de messages permettant de réaliser la procédure DAD (dont la durée est limitée par défaut à une seconde).

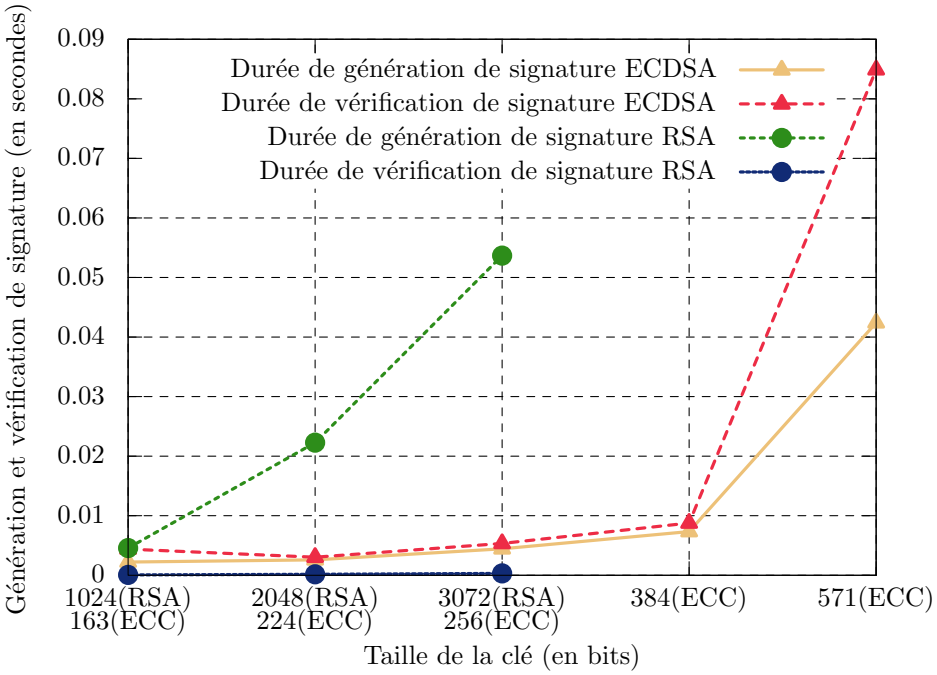


FIGURE 4.2: Comparaison entre les durées de génération et de vérification de signatures RSA et ECDSA sur un Pentium 4 à 2593 MHz

La Figure 4.2 indique de manière graphique l'écart entre la durée de génération et

de vérification de signatures avec RSA et ECDSA. Pour la vérification, RSA est plus rapide qu'ECDSA. À cause de cet écart, il pourrait y avoir des avantages à favoriser l'utilisation de RSA en fonction du modèle de communications que SEND applique :

- Dans un modèle "un à un", où deux nœuds communiquent directement entre eux (par ex. pour échanger leurs adresses de couche liaison), en utilisant de grosses tailles de clé, l'algorithme ECDSA est plus intéressant que RSA sur la moyenne des opérations de génération/vérification (une génération implique une vérification) ;
- Dans un modèle "un à plusieurs", quand un routeur envoie de nombreux messages à un groupe multicast sur le lien (par ex. pour rafraîchir la durée de vie du préfixe), l'algorithme RSA est plus approprié car la vérification de signature est plus rapide et est effectuée par chacun des membres du groupe (potentiellement grand).

Toutefois, nous n'avons pas pu évaluer quel modèle était le plus couramment utilisé dans les déploiements réels du protocole SEND aujourd'hui. Un réseau hétérogène où les routeurs utiliseraient RSA et où les hôtes utiliseraient ECDSA pourrait être un bon compromis.

4.2.4 Gain en performances sur un Tablet PC

Pour obtenir une appréciation de la durée de génération des CGA avec RSA et ECC dans un environnement à ressources limitées, nous avons effectué nos tests sur un Tablet PC. Nous avons utilisé le Nokia N800 avec un processeur compatible ARMv6 cadencé à 400MHz.

Valeur de SEC	0			
Taille de clé RSA (bits)	384	512	1024	2048
Durée de génération (totale) de la CGA	0.651353	1.004133	4.699501	35.484486
Durée de génération de la clé RSA	0.637553	0.990302	4.685756	35.470764
Durée de génération de la signature RSA	0.012324	0.021701	0.114592	0.711035
Durée de la vérification de la RSA signature	0.000522	0.000712	0.001837	0.005783
Durée de calcul du *modifier* final	0	0	0	0
Valeur de SEC	1			
Durée de calcul du *modifier* final	1.761618	1.754493	2.817053	3.873690

Tableau 4.6: Durée de génération d'une adresse CGA/RSA sur un Nokia N800 (en secondes)

À notre connaissance, il n'y a pas d'instruction RDTSC sur les processeurs ARM et notre méthodologie proposée Section 4.2.1 n'est plus adaptée. C'est pourquoi nous avons décidé de mesurer le temps écoulé durant la génération de la CGA avec la fonction *gettimeoftheday()* de la bibliothèque *time*. Nous notons toutefois que cette fonction est imprécise, car elle mesure le temps de génération global (incluant le temps consommé par les autres processus en arrière plan).

Bien que les clés RSA d'une taille supérieure ou égale à 1024 soient recommandées dans le document [Nat08a], nous avons décidé, afin d'être le plus complet possible,

Valeur de SEC	0				
Taille de clé ECC (bits)	163	224	256	384	571
Durée de gén. (totale) de la CGA	0.148385	0.169551	0.308717	0.461143	1.866542
Durée de gén. de la clé ECC	0.079611	0.086993	0.135157	0.186858	0.654368
Durée de calcul du *modifier* final	0	0	0	0	0
Durée de gén. de la sig. ECDSA	0.028778	0.037199	0.085092	0.138247	0.604534
Durée de vérif. de la sig. ECDSA	0.056505	0.045464	0.102846	0.168529	1.207743
Valeur de SEC	1				
Durée de calcul du *modifier* final	1.765540	1.760574	1.760952	2.287539	2.788008

TABLEAU 4.7: Durée de génération d'une adresse CGA/ECC sur un Nokia N800 (en secondes)

d'inclure les clés de tailles inférieures dans nos tests. Ainsi, nous avons utilisé les clés RSA de tailles 384 bits et 512 bits pour évaluer l'influence de la taille de la clé sur le temps de génération.

Le Tableau 4.6 montre que l'algorithme RSA est trop lent pour générer une adresse CGA avec un niveau de sécurité recommandé en un délai satisfaisant. Dans le cas où SEC est égal à 0 (quand *hash2* n'est pas calculé), nous constatons que la durée de génération d'une clé RSA de 1024 bits (qui est la taille minimale recommandée) est supérieure de 3 secondes à la durée requise pour les clés RSA de 384 et 512 bits. Ces résultats nous permettent de dire qu'il ne sera pas possible de générer à la volée des CGA basées sur RSA à partir de nœuds légers. Cela limite également l'utilisation de solutions fournissant une forme d'anonymat, tel que la solution proposée dans le document RFC 4941 [NDK07].

Dans le but d'améliorer la génération des CGA, nous conseillons d'utiliser des clés pré-générées afin de supprimer le délai de la génération de la clé, pour que la génération de la CGA ne dépende plus que du calcul de *hash1* qui est effectué en 10^{-4} seconde. Il faut aussi noter que la pré-génération requiert un stockage de la clé sur le nœud, ce qui implique également une solution de stockage sécurisée.

Dans le cas où le paramètre SEC est égal à 1, nous notons que le *modifier* final est calculé en plus d'une seconde. Ce délai est trop important dans un contexte de mobilité. Nous inférons que pour une valeur du paramètre SEC supérieure ou égale à 1, l'adresse CGA ne peut pas être calculée à la volée pour un nœud léger. Toutefois, comme le *modifier* final est calculé seulement au moment de la génération de la CGA, il ne devrait pas impacter le délai de *handover*, qui ne requiert que la mise à jour du condensât *hash1* pour s'adapter au nouveau préfixe sous-réseau.

Le Tableau 4.7 montre également que pour une valeur de SEC égale à 1, la génération d'une adresse CGA ne peut être réalisée dans un délai satisfaisant. La différence de temps de calcul du *modifier* final entre RSA et ECC s'explique par la différence de taille de clé, comme vu dans la Section 4.2.2.

Le Tableau 4.7 nous permet aussi de noter que l'utilisation d'ECC offre une amélioration conséquente du délai de génération d'adresse. Pour des clés de grandes tailles, le

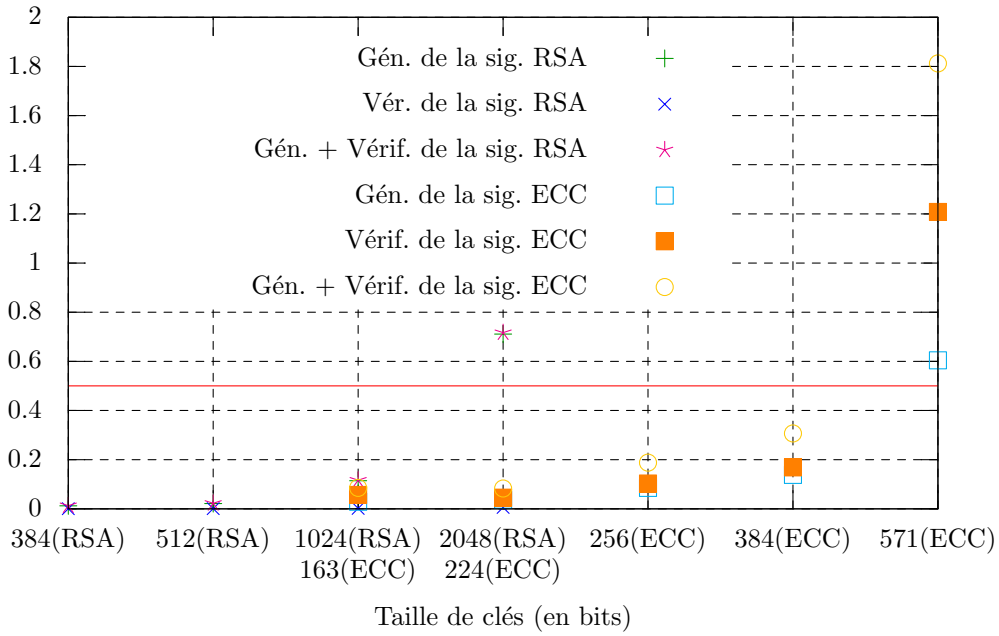


FIGURE 4.3: Vitesse de génération et vérification de signatures avec RSA et ECC sur un Nokia N800 (en secondes)

temps de génération est plus court que leurs contreparties basées sur l'algorithme RSA, mais reste toujours trop important pour une utilisation sur des nœuds légers. Des clés plus petites, de 256 bits, avec une valeur du paramètre SEC à 0, permettent des temps de génération de l'ordre de 0.15 seconde, qui rendent la génération d'adresse CGA transparente à l'utilisateur.

Comme dans la section précédente, selon le scénario de communication ("un à un" ou "un à plusieurs"), RSA peut être préférable à ECC. Toutefois, cette affirmation doit être remise dans le contexte de mobilité où la faible taille de clé offerte par l'utilisation d'ECC permet une économie d'énergie pour le transport durant les communications sans fils. Nous pensons que ce point est à considérer et favorise l'utilisation d'ECC vis à vis de RSA. Nous notons également que les grandes tailles de clés (RSA et ECC confondus) ne sont pas viables sur des nœuds légers, comme souligné dans la Figure 4.3, où les valeurs au dessus de 0.5 seconde (au dessus de la ligne horizontale) ne permettent pas le fonctionnement de la procédure de Détection d'Adresse Dupliquée (voir Section 4.2.3).

Nous concluons cette section en recommandant l'utilisation d'ECC pour la génération de CGA sur les nœuds légers. De plus, grâce à l'utilisation conjointe avec une valeur du paramètre SEC à 0, le calcul d'adresse CGA à la volée devient possible.

4.3 Évaluation de l'impact de la fonction de hachage sur la génération des CGA

À cause de récentes attaques [WYY05], l'utilisation de la fonction de hachage SHA-1 [Nat08b] n'est plus recommandée par le NIST (*National Institute of Standards and Technology*). Ce dernier conseille désormais d'utiliser les fonctions de hachage appartenant à la famille SHA-2 [2]. Un travail en cours, [KKJ10], au sein de groupe de travail "CGA and SEND maIntenance" de l'IETF détaille l'impact des attaques de SHA-1 sur CGA et SEND.

Un autre document, également produit par l'IETF [BA07], décrit comment une réutilisation des valeurs actuellement inutilisées du paramètre SEC (valeurs au dessus de 2) permettrait d'indiquer de nouvelles fonctions de hachage pour les adresses CGA. La sémantique des valeurs 0, 1, et 2 actuelles serait conservée alors que les valeurs de 3 à 7 seraient quant à elles recyclées. Elles indiqueraient toujours un nombre de 0 nécessaires au début du condensât *hash2* (0, 16 ou 32), mais ajouteraient également une indication sur la fonction de hachage à utiliser pour générer/vérifier la CGA. Indiquer la fonction de hachage dans le paramètre SEC, et donc la stocker dans l'identifiant d'interface, permet d'éviter les attaques de type "downgrade", où un attaquant force l'usage d'une fonction de hachage vulnérable sur un nœud victime.

Du fait de ces attaques, et leurs conséquences éventuelles, nous avons choisi d'évaluer les performances des fonctions de hachage qui pourraient remplacer SHA-1 au sein des CGA. Nos tests ont porté sur : SHA-256, SHA-512 [Nat08b] (tous deux actuellement recommandés par le NIST), RIPEMD-160 [DBP96], TIGER2 [AB96] et WHIRLPOOL [whi]. SHA-224 et SHA-384 ne sont pas évaluées car ce sont des versions de SHA-256 et SHA-512 dont la sortie est tronquée. TIGER2 et WHIRLPOOL bénéficient des architectures 64 bits, dont la popularité au sein des serveurs et machines de bureau ne cesse de croître. À l'exception de TIGER2 [BA] et de WHIRLPOOL, où nous avons utilisé une implémentation disponible en ligne, toutes les fonctions de hachages testées font partie de la bibliothèque cryptographique OpenSSL [3].

Pour chaque fonction, nous évaluons la durée de calcul du *modifier* final pour une CGA basée sur des clés RSA de différentes tailles et ayant une valeur de SEC égale à 1. Ces tests nous permettent d'évaluer l'influence de la fonction de hachage sur le coûteux calcul du condensât *hash2*. Ces résultats sont présentés dans la Figure 4.4. Pour rappel, une valeur de SEC égale à 1 implique en moyenne un calcul de 2^{16} condensâts (voir Formule 4.1).

La Figure 4.4 montre que le temps de génération du *modifier* final dépend de la taille de la clé, ce qui est normal, puisque le condensât est calculé sur la structure de données *CGA Parameters* qui contient la *clé publique*. Les résultats sur SHA-256 et SHA-512 ne sont pas surprenants, car ils sont tous deux connus pour être plus lent que SHA-1. SHA-512 profite néanmoins de notre architecture 64 bits et offre des performances supérieures à SHA-256. La fonction WHIRLPOOL [whi], comme décrit dans

2. composée de SHA-256, SHA-384 et SHA-512
3. http://www.openssl.org

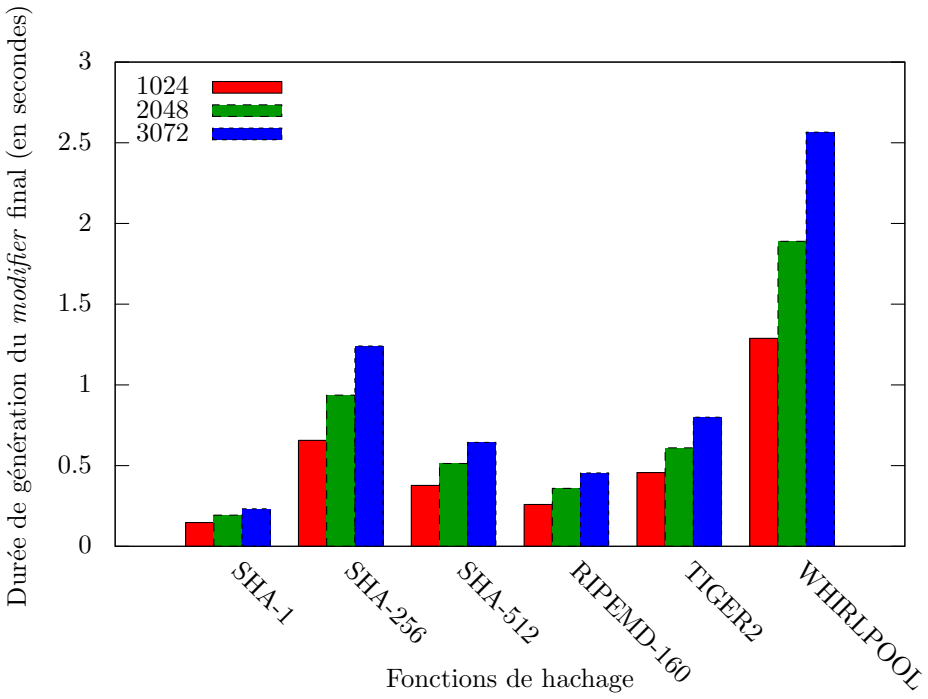


FIGURE 4.4: Comparaison des temps de génération d'une adresse CGA-RSA (SEC=1) en évaluant différentes fonctions de hachage pour différentes longueurs de clés RSA

l'article [NM02], est connue pour être plus lente que SHA-256. Toutefois, cette contre-performance serait plutôt à attribuer à un manque d'optimisation de l'implémentation.

Nous remarquons que l'algorithme le plus efficace pour la génération du *modifier* final est l'algorithme actuellement utilisé : SHA-1. RIPEMD-160 se classe second et constitue également une alternative intéressante. Les performances de TIGER2 sont très proches de celles de RIPEMD-160, puisque TIGER2 profite de notre architecture 64 bits (car l'algorithme travaille sur des mots de 64 bits).

À l'heure actuelle, SHA-256 et la famille complète SHA-2 sont évalués au sein de l'IETF et devraient être proposés comme prochains algorithmes de hachage pour les CGA. Cette décision n'est pas basée sur les performances : dans nos différents tests, nous voyons clairement que SHA-256 et SHA-512 ne sont pas les fonctions les plus rapides. Ce choix est basé sur le niveau de sécurité et sur la robustesse de la famille de fonctions de hachage SHA-2, qui a fait l'objet d'une analyse plus poussée que ses concurrents par le NIST et la communauté de la cryptographie. Toutefois, nous savons maintenant que ce choix ralentira le processus de génération de la CGA.

4.4 Utilisation de techniques GPGPU pour accélérer la génération des CGA

De nos jours, les cartes graphiques intégrées aux nouveaux ordinateurs sont de plus en plus puissantes et, depuis peu, sont également capables d'effectuer du *General-Purpose Computing on GPUs* (GPGPU). C'est à dire qu'elles sont capables d'effectuer des opérations génériques qui sont généralement réalisées sur le CPU. Toutefois, si le coût et l'utilité d'une carte graphique sont parfaitement justifiés auprès des utilisateurs finaux, ce n'est pas le cas pour les accélérateurs cryptographiques, qui sont quant à eux rarement présents. Dans cette section, nous proposons d'utiliser les fonctionnalités offertes par le GPGPU, afin de paralléliser l'algorithme de génération des CGA. Une analyse des gains réalisables avec les futures générations de cartes graphiques est également proposée.

Pour effectuer nos tests, nous avons porté l'implémentation de SHA-1 de la bibliothèque XySSL [XyS] sur la plate-forme fournie par NVIDIA nommée CUDA [CUD]. Cette implémentation a été choisie pour la faible taille de son code qui lui permet de tenir dans la mémoire d'une carte graphique. Le *framework* CUDA offre un langage de programmation proche du C qui facilite la réalisation d'opérations en parallèle sur les cartes graphiques NVIDIA.

Notre idée principale est de paralléliser les exécutions de SHA-1 nécessaires au calcul du *modifier* final. Comme nous l'avons vu, le calcul du *hash2* est souvent l'opération la plus intensive de la génération d'une CGA. De multiples appels à SHA-1 sont effectués sur la structure de données *CGA Parameters*, qui est légèrement modifiée pour chaque appel afin d'effectuer le calcul sur une valeur de *modifier* différente. De ce fait, nous testons les valeurs de *hash2* pour de multiples valeurs de *modifier* en parallèle. Pour se faire, nous modifions légèrement l'algorithme de génération des CGA comme indiqué sur la Figure 4.5.

Nous avons effectué nos tests sur une NVIDIA Geforce 8600 GT et une NVIDIA Geforce 8600 GTS [4]. Ces deux cartes ont le même nombre de Multiprocesseurs (utilisés pour le calcul des données), mais la version GTS a une fréquence de GPU et de mémoire plus élevée (la GTS a un GPU cadencé à 675 MHz et une mémoire à 1000 MHz alors que la version GT est limitée à 540 MHz et 700 MHz respectivement). À cause des limitations de ces cartes de moyenne gamme (faible mémoire partagée, faible nombre de registres, etc.), notre portage du code de SHA-1 ne tient pas complètement dans la mémoire dite "rapide" de la carte et une partie des données est conservée dans la mémoire locale de l'ordinateur (plus lente d'accès), impliquant une perte de performances. Cette limitation est en partie masquée grâce à l'ordonnancement des multiples *threads* qui augmente le nombre d'accès mémoire pour amoindrir les effets de latence.

Nous avons effectué sur chaque carte 500 calculs de *modifier* final avec une CGA basée sur une clé RSA de 1024 bits et ayant une valeur de paramètre SEC égale à 2. Il faut en moyenne 1351 secondes (soit 22 minutes) pour calculer les valeurs du *modifier* final sur la NVIDIA 8600 GT et 852 secondes (soit 14 minutes) sur la NVIDIA

4. Nous avons vérifié que le type et la fréquence du CPU n'influencent pas les mesures.

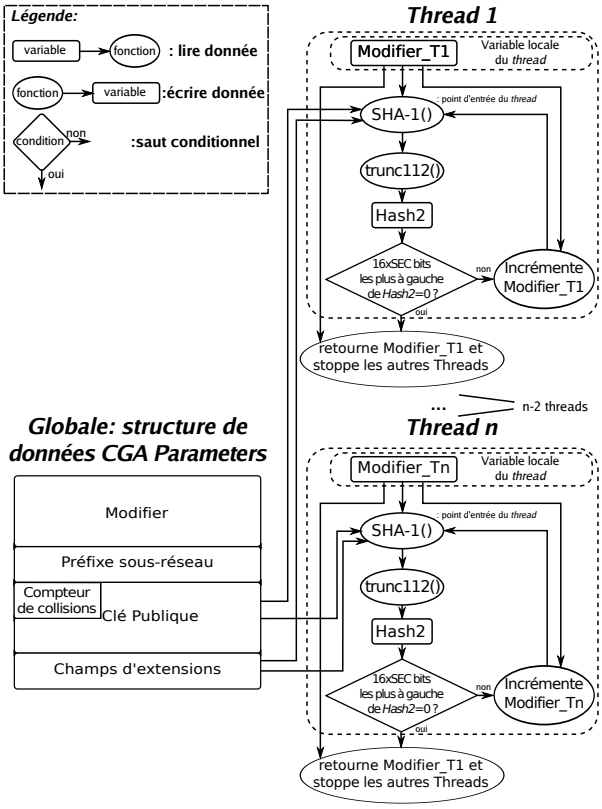


FIGURE 4.5: Calcul parallèle de *hash2* sur un GPGPU

8600 GTS. Cette différence est due à l'écart entre ces deux cartes de la fréquence du GPU et de la mémoire. L'écart type est de 1300 secondes sur la NVIDIA 8600 GT, ce qui indique un fort aléa dans le processus de génération d'adresses. Ces résultats doivent être comparés avec les 2.4 heures nécessaires en moyenne sur un Pentium 4 (voir Section 4.2.2). Avec la NVIDIA 8600 GTS, la vitesse de génération est améliorée par un facteur de 10. Il s'agit là d'une amélioration notable, même si une génération d'adresse qui dure 14 minutes reste trop élevée pour un usage réel.

Nous n'avons pas pu tester les dernières cartes haut de gamme de la série des GT200, mais nous pensons que cette série de cartes permettrait des temps de génération au moins 7 fois plus rapides, offrant ainsi un facteur d'accélération total de 70. Cette conjecture est basée sur le fait que les cartes de la série GT200 contiennent 7 fois plus de Multiprocesseurs, tout en ayant des GPU plus rapides, plus de registres et un nouveau mécanisme de gestion de la mémoire. Avec le GPGPU, les CGA dont la valeur de SEC est égale à 2 pourraient devenir plus répandues dans les prochaines années, quand les temps de générations seront inférieurs à la seconde.

Il existe une proposition [JX09] au sein de l'IETF qui décrit un scénario où les serveurs DHCPv6 généreraient des adresses CGA à la place des nœuds mobiles afin de les décharger. Notre proposition pour l'utilisation du GPGPU serait intégrable dans de tels serveurs DHCPv6. Elle permettrait de réduire le temps de génération d'une CGA et fournirait un remplaçant moins coûteux que les accélérateurs cryptographiques.

Pour conclure cette section, nous notons que l'utilisation du GPGPU pourrait être combinée avec l'arrivée des CPU multi-cœurs afin d'améliorer le calcul de *hash2*. Nous remarquons également que le temps de calcul d'une CGA dont le paramètre SEC vaut 3 est en moyenne 2^{16} fois supérieur au temps de calcul d'une CGA dont le paramètre SEC est égal à 2, ce qui impliquerait des semaines de calculs avec le matériel actuel. À moins qu'un nouveau bond technologique ne survienne, nous pouvons considérer que les CGA dont le paramètre SEC est supérieur ou égal à 3 ne seront pas utilisables dans un futur proche. Ce qui conforte la réutilisation des valeurs du paramètres SEC au dessus de 2, comme dans le document [BA07].

4.5 Synthèse du chapitre

Dans ce chapitre, nous avons montré que l'avènement de nouvelles technologies rendait possible une amélioration significative des performances de CGA et de SEND. Nous avons constaté ces améliorations dans deux environnements différents : une machine du bureau et un nœud mobile disposant d'une puissance de calcul limitée. Nos travaux prouvent qu'il est possible d'utiliser SEND sur ces appareils sans impact majeur sur les performances. De plus, l'utilisation d'ECC rend possible l'utilisation de SEND sur les nœuds mobiles alors que RSA ne le permettrait pas.

Une utilisation pertinente de la généralisation des techniques GPGPU nous a permis d'améliorer le temps de génération des CGA et d'ouvrir de nouvelles perspectives quant à l'usage d'une valeur du paramètre SEC égale à 2, jusque là inutilisable en pratique.

Nous avons également évalué les conséquences de la suppression de la fonction de hachage SHA-1 en étudiant les performances des solutions alternatives.

5 Mécanisme Signature Algorithm Agility pour étendre CGA et SEND à d'autres algorithmes cryptographiques

Le Chapitre 4 ne s'intéresse qu'à démontrer le gain en performances à attendre du remplacement de l'algorithme RSA par ECC au sein des CGA et du protocole SEND. Toutefois, seules les modifications minimales nécessaires à l'utilisation d'ECC dans le protocole SEND y sont décrites.

Le travail présenté dans ce chapitre étend ces recherches et crée un mécanisme rendant le protocole SEND indépendant de l'algorithme de signature. Ce travail, réalisé au niveau protocolaire, autorise également les communications entre les nœuds utilisant différents algorithmes de signature. Ce point nous donne la flexibilité requise pour l'introduction éventuelle de nouveaux algorithmes cryptographiques dans le futur. Tout au long de ce chapitre, nous utilisons le terme *Signature Algorithm Agility* pour se référer à ce mécanisme.

La Section 5.1 présente les limitations actuelles dans les spécifications des adresses CGA [Aur05] et du protocole SEND [AKZN05]. Elle indique également les motivations qui nous ont poussées à travailler sur la problématique de *Signature Algorithm Agility*. La Section 5.2 présente la solution retenue à l'IETF pour rendre les CGA indépendantes de la fonction de hachage. Ensuite, sont abordées nos propres contributions. La Section 5.3 présente notre solution initiale, permettant une compatibilité maximale entre les nœuds SEND conformes au RFC 3971 [AKZN05] (dit aussi "nœuds RFC 3971") et de nouveaux nœuds qui implémenteraient le mécanisme de *Signature Algorithm Agility*. Finalement, la Section 5.4 présente la solution de *Signature Algorithm Agility* allégée, telle que proposée à la standardisation au sein de l'IETF. Cette version permet toujours un échange entre les nœuds RFC 3971 et les nœuds gérant la *Signature Algorithm Agility*. Toutefois, l'effort est placé sur la simplification du mécanisme, ce qui nous a conduits à retirer certaines fonctions de rétro-compatibilité trop coûteuses.

La solution de *Signature Algorithm Agility* pour SEND présentée dans ce chapitre fait partie de nos contributions et a été publiée dans la conférence *Sécurité des Architectures Réseaux et des Systèmes d'Information* [CBLM09] (SAR-SSI) en 2009 (prix du meilleur article étudiant). Les parties plus techniques sont détaillées dans les *drafts* IETF [CLMSV09], [CLSV10b] et [CLSV10a].

5.1 Limitations des CGA et de SEND restreignant l'usage de nouveaux algorithmes cryptographiques

Dans le Chapitre 3, nous avons indiqué que le protocole SEND [AKZN05] ne permet que l'utilisation de l'algorithme RSA et la fonction de hachage SHA-1 pour la signature des messages SEND. En effet, l'option de *signature RSA* n'est pas conçue pour intégrer d'autres algorithmes de signature. Le format de l'option, présenté Figure 3.3, montre que les champs *signature numérique* et *bourrage* ont tous deux une taille variable. Ce qui représente une difficulté pour le décodage de l'option. Toutefois, l'utilisation de l'algorithme RSA permet de déterminer la taille de la *signature numérique* à partir de la taille de la clé publique. D'autres algorithmes de signature, comme ECC, ne permettent pas de déterminer la taille de la signature à partir de la clé publique, d'où l'impossibilité de réutiliser sans modification l'option *signature RSA*.

Le Chapitre 4 démontre que l'utilisation de nouveaux algorithmes cryptographiques, comme ECC, permettent un gain en performance et une amélioration du niveau de sécurité significatifs. De plus, il est toujours préférable d'avoir en secours la possibilité de migrer rapidement d'un algorithme à un autre en cas de vulnérabilités décelées sur l'un des algorithmes utilisés. D'autres motivations non techniques peuvent également justifier l'utilisation de différents algorithmes. C'est notamment le cas de nations privilégiant leurs propres suites cryptographiques (par ex. la suite cryptographique *GOST*).

Comme nous l'avons décrit dans la Section 3.1.1, les adresses CGA ont la capacité de stocker n'importe quel type de clé publique dans leur structure de données *CGA Parameters*. Concernant les fonctions de hachage, les adresses CGA sont aujourd'hui limitées à l'utilisation de la fonction de hachage SHA-1 pour le calcul des condensâts *hash1* et *hash2*. Pourtant, le NIST recommande de ne plus utiliser cette fonction pour de nouveaux protocoles [NIS]. Il est donc utile de prévoir une migration vers d'autres types de fonctions de hachage et un mécanisme de sélection de la fonction de hachage dans les CGA.

5.2 Diversifications des fonctions de hachage dans les CGA

Le document RFC 4982 [BA07] analyse les différentes options pour étendre les CGA à de nouvelles fonctions de hachage. Après avoir discuté les avantages et inconvénients de chaque option, le document propose une solution qui ne compromet pas la sécurité des CGA. Pour ne pas être vulnérable aux attaques de type "*downgrade*", où un attaquant pourrait forcer le nœud vérifiant une CGA à utiliser une fonction de hachage plus faible ou cassée, la fonction de hachage utilisée doit être encodée dans l'adresse.

Reprendre des bits inutilisés dans l'*identifiant d'interface* affaiblirait les CGA en diminuant l'efficacité de *hash1* face aux attaques de type force brute. La solution proposée consiste alors à redéfinir le paramètre de sécurité SEC. La sémantique du paramètre SEC est alors changée afin d'indiquer, en plus du nombre de bits de *hash2*

Valeur du paramètre SEC	Nombre de bits à gauche de *hash2* à 0	Fonction de hachage
0	0	SHA-1
1	16	SHA-1
2	32	SHA-1
3–7	à définir	à définir

TABLEAU 5.1: Nouvelle sémantique du paramètre SEC (ajout de l'encodage de la fonction de hachage) introduite par le document RFC 4982 [BA07]

à 0, la fonction de hachage utilisée pour la génération et la vérification de la CGA.

Puisque les valeurs de SEC au dessus de 2 sont difficilement utilisables pour des raisons de performances, comme nous l'avons montré dans la Section 4.4, celles-ci sont réservées pour de nouvelles fonctions de hachage. Par exemple, la valeur 3 pourrait désigner "0 bit du *hash2* à 0 et fonction de hachage SHA-256".

Toutefois, aucune nouvelle valeur n'a été définie au moment de l'écriture de ce manuscrit. Le Tableau 5.1 récapitule les valeurs actuellement définies. On notera la rétrocompatibilité avec les valeurs définies dans la spécification des CGA [Aur05].

5.3 Version élaborée de la solution de Signature Algorithm Agility

Dans cette section, nous proposons une solution de *Signature Algorithm Agility*. Cette solution s'oriente vers une rétrocompatibilité avec les anciens nœuds et une interopérabilité maximale entre les nœuds. Comme nous le verrons, notre solution privilégie des mécanismes de transitions douces, où plusieurs algorithmes de signature peuvent être utilisés simultanément au sein du même nœud et où une négociation pour le choix de l'algorithme de signature est possible. Même si nous autorisons l'usage de signatures multiples pour un même message afin faciliter la transition, à terme le retour à l'utilisation d'un unique algorithme de signature est préférable.

Lors de la phase de transition vers une utilisation généralisée de la *Signature Algorithm Agility*, les nœuds utilisant SEND sont divisés en deux catégories : les nœuds reposant sur le protocole RSA et le protocole SEND [AKZN05], que nous nommons "nœud RFC 3971", et les nœuds supportant la *Signature Algorithm Agility*, utilisant RSA et/ou ECC et/ou de nouveaux algorithmes de signature.

5.3.1 Extension pour le support des CGA multi-clés

Les CGA telles que définies dans le document RFC 3972 [Aur05] contiennent une possibilité d'extension. En effet, la structure de données *CGA Parameters*, illustrée Figure 3.1 comprend un champ appelé *Extensions* qui suit le format générique décrit par le document RFC 4581 [BA06]. Nous proposons donc de définir une nouvelle

extension *Public Key Extension* qui contient une clé publique, sans contrainte quant au type de clé. Cette clé publique est stockée au format DER correspondant à la définition ASN.1 du type *SubjectPublicKeyInfo* tel que décrit dans le document [BA06].

Ainsi, un nœud peut être associé à plusieurs clés publiques de différents types. Pour cela, il précisera dans le champ *Clé publique* (également au format *SubjectPublicKeyInfo*) de la structure de données *CGA Parameters* une première clé publique, puis dans une ou plusieurs extensions *Public Key Extension* ses autres clés publiques. Pour garantir la rétrocompatibilité, un nœud disposant d'une clé publique de type RSA positionnera en priorité cette clé dans le champ *Clé publique*, ceci dans le but de permettre la lecture de cette clé par les nœuds RFC 3971.

Cette idée de CGA multi-clés est particulièrement intéressante pour les routeurs, où un même routeur peut ainsi communiquer avec des nœuds RFC 3971 et des nœuds supportant la *Signature Algorithm Agility* et un algorithme différent de RSA.

La génération de l'adresse CGA associée à un nœud disposant de plusieurs clés publiques se fait sur la structure de données *CGA Parameters* dans son entier, y compris sur les nouveaux champs *Public Key Extensions*. De cette manière, une adresse CGA se trouve liée à plusieurs clés publiques de types différents.

5.3.2 Adaptation de SEND à de multiples algorithmes de signature

Définition de l'option Signature Universelle

Avec cette notion de CGA multi-clés, un nœud doit pouvoir prouver la possession de chacune de ses clés publiques par le biais de signatures qu'il est possible d'adjoindre au même message ND. Pour cela, nous avons étendu l'option *Signature RSA* de SEND. Nous l'avons renommée *Signature Universelle*. Pour ce faire, nous utilisons le champ *réservé* de 16 bits pour préciser à quelle clé, stockée dans la structure de données *CGA Parameters*, la signature se réfère. Le nouveau format de l'option est présenté Figure 5.1.

De nouveaux champs permettent de désigner la clé publique utilisée et d'indiquer le type de signature associée :

- le champ *taille bourrage* indique la taille du champ *bourrage*. Dans les *drafts* précédant la normalisation du document RFC 3971 [AKZN05], ce champ était présent dans l'option *Signature RSA* pour aider à identifier la taille variable des deux champs *signature numérique* et *bourrage*. Il est à noter que l'algorithme RSA permet de déduire la taille de la signature numérique par la connaissance de la taille de la clé publique (contenu dans la structure de données *CGA Parameters*). Ainsi le champ *taille bourrage* a été amené à disparaître du RFC. Pour d'autres algorithmes, comme ECC, il n'est pas possible de déterminer la taille de la signature de la même façon, d'où la réintroduction de ce champ ;
- le champ *position* indique quelle clé contenue dans la structure de données *CGA Parameters* doit être utilisée pour la vérification du champ *signature numérique*. La valeur 0 pointe vers le champ *clé publique*. La valeur 1 pointe vers la clé contenue dans la première extension de type *Public Key extension*, et ainsi de suite ;

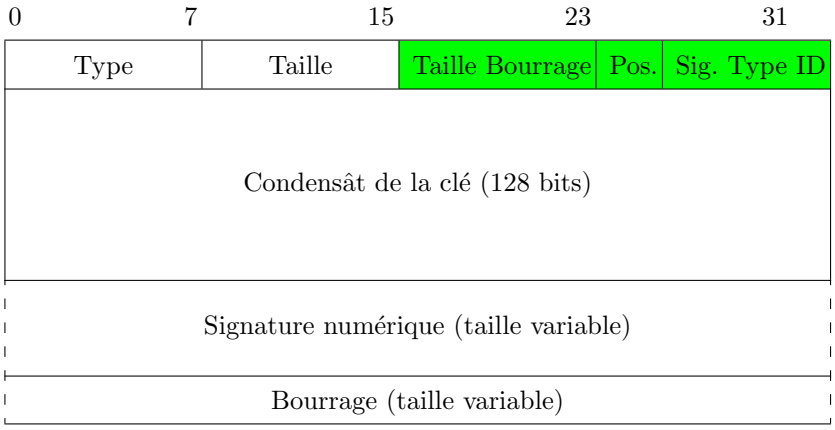


FIGURE 5.1: Format de l'option *Signature Universelle*

Valeur	Algorithme de signature	Fonction de hachage
0	RSA	SHA-1
1	RSA	SHA-256
9	ECDSA (courbe P-256)	SHA-256
10	ECDSA (courbe P-384)	SHA-384
11	ECDSA (courbe P-521)	SHA-512

TABLEAU 5.2: Différentes valeurs du *Signature Type Identifier*

– le champ *Signature Type ID*, sur 5 bits, indique le couple "algorithme de signature / fonction de hachage" utilisé pour la génération du champ *condensât de la clé* et *signature numérique*. Ces valeurs sont indiquées dans le Tableau 5.2. On notera que dans un effort de rétrocompatibilité avec les anciens nœuds, la valeur 0 est utilisée pour désigner le couple RSA/SHA-1, utilisé dans SEND [AKZN05]. Cela facilite la communication entre les nœuds RFC 3971, qui ne vérifient pas ce champ à la réception et le positionnent à 0 à l'émission, et les nouveaux nœuds supportant la *Signature Algorithm Agility*.

Définition de l'option Supported Signature Algorithm

Avec l'introduction de multiples algorithmes de signature, il est clair que certains scénarios, en fonction des algorithmes supportés par les nœuds, risquent d'aboutir à l'impossibilité des nœuds de vérifier la ou les signature(s) précisée(s) dans le message ND. Par exemple, d'anciens nœuds RFC 3971 n'implémentant que des CGA RSA peuvent communiquer avec de nouveaux nœuds utilisant à la fois RSA et d'autres algorithmes de signature. D'autres nœuds pourraient refuser d'utiliser certains algo-

rithmes pour des raisons de consommation énergétique. Il est donc nécessaire de passer par une étape de négociation entre les nœuds.

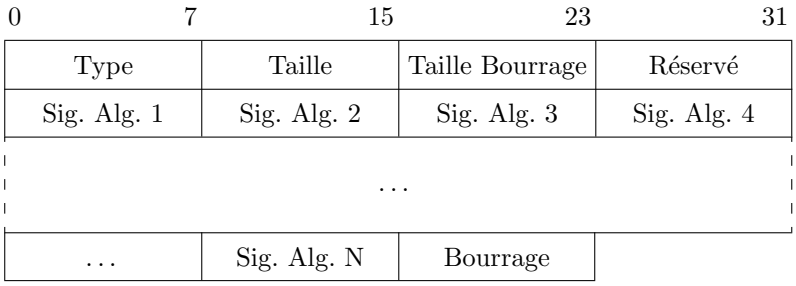


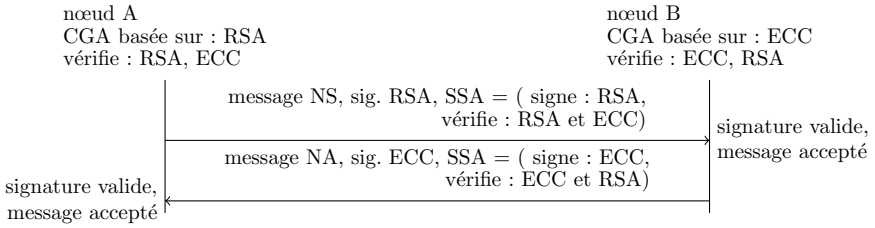
FIGURE 5.2: Format de l'option *Supported Signature Algorithm*

Afin de pouvoir négocier et comprendre le type de clés disponibles en chacun des nœuds, nous avons été amenés à définir une nouvelle option pour les messages ND (sécurisés par SEND). Cette option, dont le format est illustré Figure 5.2, est appelée *Supported Signature Algorithm* (SSA) et contient la liste des algorithmes de signature disponibles sur le nœud. Cette liste est contenue dans différents champs *Signature Algorithm*, encodés sur 1 octet. Pour chacun des algorithmes supportés (classés par ordre de préférence), une distinction est faite si le nœud supporte uniquement la vérification de signature (premier bit le plus à gauche mis à 0) ou bien la signature et la vérification (bit mis à 1). Cela permet aux nœuds d'annoncer qu'ils comprennent certains algorithmes de signature durant la phase de négociation alors qu'ils ne souhaitent ou ne peuvent pas signer avec ces algorithmes. Les 5 derniers bits du champ *Signature Algorithm* contiennent la valeur du *Signature Type Identifier*. La signification de cette valeur est identique au champ *Signature Type Identifier* de l'option *Signature Universelle* : il indique l'algorithme de signature et la fonction de hachage utilisables pour la vérification et la réalisation de la signature. L'ordre des champs *Signature Algorithm* permet de déterminer l'ordre de préférence des différents algorithmes de signature d'un nœud. Ainsi, le premier champ *Signature Algorithm* indique l'algorithme de signature préféré.

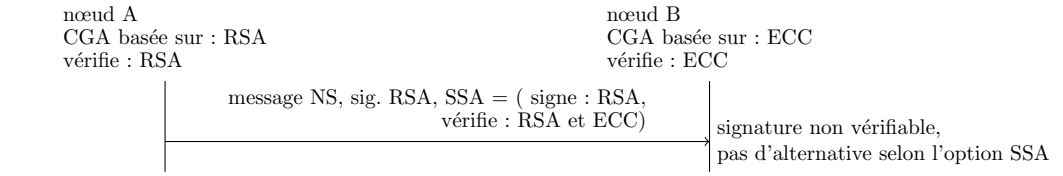
Négociation de l'algorithme cryptographique

La phase de négociation est assez légère. Lors d'un échange de message classique de type *Neighbor Solicitation/Neighbor Advertisement* et *Router Solicitation/Router Advertisement*, l'option SSA est jointe. À la fin d'un échange, l'option SSA permet à chaque nœud de déterminer si la (les) signature(s) a pu être comprise de son correspondant. Dans le cas contraire, le nœud envoie à nouveau un message avec un autre type de signature à son correspondant. L'absence d'option *Supported Signature Algorithm* permet de détecter les nœuds RFC 3971.

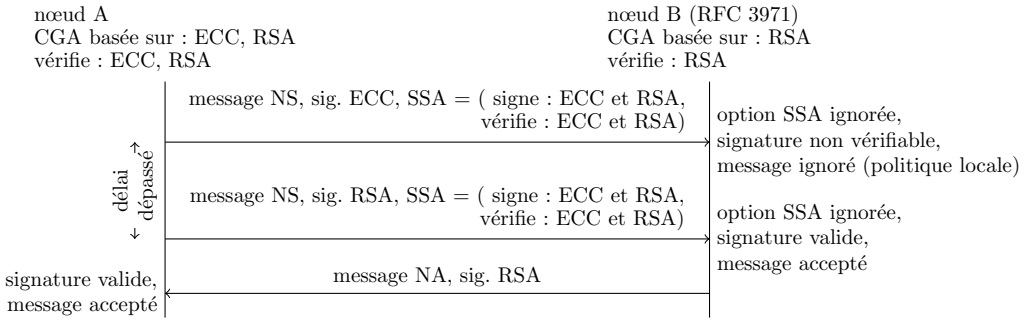
La Figure 5.3 illustre plusieurs cas représentatifs de la négociation durant une procédure de *résolution d'adresse* :



(a) Négociation réussie entre deux nœuds supportant la *Signature Algorithm Agility* et ayant des algorithmes de signature en commun



(b) Échec de la négociation entre deux nœuds supportant la *Signature Algorithm Agility* et n'ayant que des algorithmes de signatures disjoints



(c) Négociation réussie entre un nœud supportant la *Signature Algorithm Agility* et un nœud RFC 3971

FIGURE 5.3: Exemple de négociations possibles

- dans la Figure 5.3a, les deux noeuds sont capables de vérifier l'algorithme de signature de leur correspondant et alors peuvent communiquer entre eux. À la réception du message *Neighbor Solicitation*, le nœud B analyse l'option SSA pour déterminer quel algorithme de signature utiliser pour la réponse ;
- dans la Figure 5.3b, les nœuds ont des ensembles d'algorithmes de signature disjoints et ne peuvent vérifier la signature de leur correspondant. Le nœud B, qui reçoit la requête *Neighbor Solicitation*, ne peut alors vérifier la signature et considère le message comme un message non-sécurisé. Selon la politique de sécurité, le message est ignoré. L'option SSA permet ici de diagnostiquer les raisons de l'échec. Nous verrons toutefois dans la Section 5.3.4 qu'il est possible d'étendre le protocole SEND pour permettre la communication entre ces deux nœuds ;
- Finalement, la Figure 5.3c montre que la communication avec un nœud RFC 3971 est possible. Le nœud A préfère ECC à RSA, et envoie un premier message signé avec ECC. Le nœud B ne fournissant aucune réponse, le nœud A attend un délai minimal avant d'envoyer le même message signé avec RSA. Ici, on constate que l'ordre de préférence des algorithmes de signature peut ralentir l'échange de messages.

Pour les messages spontanés (comme les *Router Advertisement*), "diffusés" sur le réseau et n'ayant donc pas de destinataire spécifique, il est possible que les messages ne puissent pas être vérifiés par le récepteur. Toutefois, les options *Supported Signature Algorithm* contenues dans les messages *Router Solicitation* émis par les nœuds rejoignant le réseau permettent au routeur de déterminer le ou les types de signature à joindre aux messages *Router Advertisement* pour une compréhension par tous les nœuds. Afin d'ajuster ses messages aux capacités des récepteurs, le routeur pourra émettre ses messages *Router Advertisement* en les signant simultanément avec plusieurs signatures (plusieurs options *Signature Universelle*).

5.3.3 Utilisation des certificats SEND dans la Signature Algorithm Agility

Du fait de l'introduction de plusieurs clés associées à une même entité et l'utilisation de l'une de ces clés pour générer et vérifier des signatures, des problèmes sous-jacents se posent.

Quand le nœud est un routeur, il est intéressant de regarder comment est réalisée la publication de ses clés publiques. Dans SEND, les routeurs sont munis de certificats qui contiennent dans leur champ *SubjectPublicKey* la même clé publique que celle précisée dans le champ *Clé publique* de la structure *CGA Parameters*. Pour permettre l'utilisation de CGA à clés publiques multiples (comme le suggère la section 5.3.1), il faudrait soit générer autant de certificats que de clés publiques utilisées, soit encoder plusieurs clés publiques dans un même certificat. Le premier choix étant techniquement irréalisable (déployer un seul certificat par routeur est déjà assez complexe), nous étendons les certificats X.509 afin de leur faire transporter plusieurs clés publiques. À l'image des CGA, nous définissons un champ *Extensions* pour transporter nos clés publiques. Celles-ci sont alors automatiquement signées par l'autorité de certification

du certificat.

5.3.4 Notarisation

Définitions d'un nouveau rôle dans les certificats X.509

Avec la solution actuelle, comme illustré dans la Figure 5.3b, deux nœuds n'ayant aucun algorithme de signature en commun ne peuvent pas entrer en communication. Pour résoudre ce problème, nous donnons la possibilité aux routeurs de jouer le rôle de "notaire" avec la fonction de valider des messages SEND pour le compte de nœuds voisins (à condition qu'il supporte leur(s) algorithme(s) de signature). Cela signifie qu'un routeur peut s'annoncer sur le réseau avec cette fonction de notaire. Pour ce faire, l'autorité de certification du routeur précise dans le certificat X.509 l'autorisation pour le routeur d'agir comme un notaire. Dans la pratique, le rôle de notaire est indiqué par le certificat du routeur via une extension EKU (*Extended Key Usage*), à l'image des nouveaux rôles définis dans le document [GKK10].

Définitions des messages Signature Check Request et Signature Status

Afin de réaliser l'échange avec le notaire, deux nouveaux messages ICMPv6 [CDG06] *Signature Check Request* (SCR) et *Signature Status* (SS) sont définis.

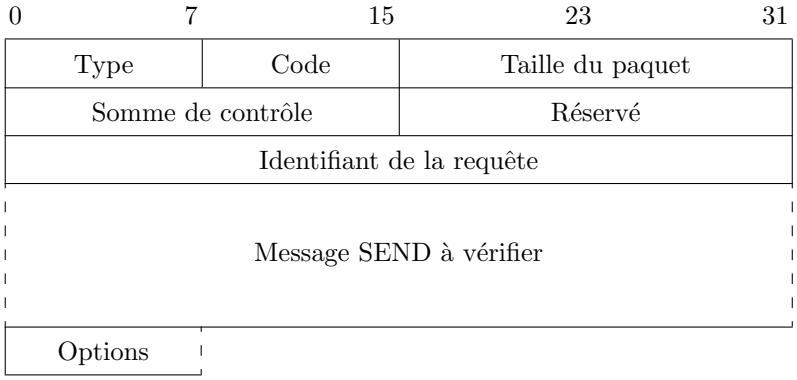


FIGURE 5.4: Format du message *Signature Check Request*

La Figure 5.4 présente le format du message *Signature Check Request*. Les champs du message sont :

- le champ *taille du paquet* indique la taille du message SEND qui est contenu dans le champ *message SEND à vérifier* ;
- le champ *somme de contrôle* correspond à une somme de contrôle de type CRC-16 sur tout le message (*Signature Check Request*). Ce champ est mis à zéro durant le calcul de la somme de contrôle. Le but de ce champ est de détecter rapidement les erreurs de transmissions ;

- le champ *réservé* est réservé à un usage futur ;
- le champ *identifiant de la requête* permet de faire la correspondance entre la requête *Signature Check Request* et la réponse *Signature Status* associée. Cet identifiant est généré par l'émetteur de façon aléatoire ;
- le champ *message SEND à vérifier* contient le message que l'initiateur de la requête n'a pu vérifier par lui-même ;
- le champ *options* contient une ou plusieurs options NDP. Une seule de ces options doit obligatoirement être présente : il s'agit de l'option *Supported Signature Algorithm* qui permet au notaire de déterminer quel type de signature peut-être vérifié par l'initiateur, et donc de choisir la signature à utiliser pour le message *Signature Status*. Le message *Signature Check Request* peut optionnellement être protégé par les options SEND classiques (option *CGA*, *horodatage*, *nonce* et *signature universelle*). Dans ce cas, la protection du message par l'option *signature universelle* permet de s'assurer de l'intégrité du *message SEND à vérifier*. Dans le cas contraire, une modification du *message SEND à vérifier* ne serait détectée que lors de la vérification du champ *condensât du message* du message *Signature Status*.

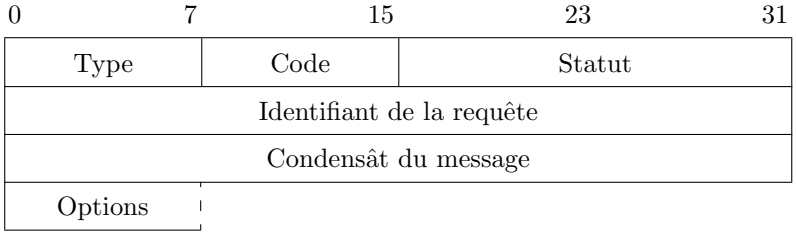


FIGURE 5.5: Format du message *Signature Status*

Le résultat de tests réalisés sur le *message SEND à vérifier* est ensuite transmis au nœud initiateur de la requête en utilisant un message *Signature Status*. La Figure 5.5 présente le format du message *Signature Status*. Les champs qui composent le message sont :

- le champ *statut* indique l'état de la signature qui a été évaluée par le notaire. Parmi les indications possibles, on peut savoir qu'une signature est valide ou erronée (le but de cet échange de message), mais aussi que l'algorithme à vérifier n'est pas supporté par le notaire ou bien que le notaire est trop occupé pour effectuer la vérification ;
- le champ *identifiant de la requête* fait correspondre le message *Signature Status* avec le message *Signature Check Request* initial ;
- le champ *condensât du message* contient un condensât calculé sur *l'identifiant de requête* et le *message SEND à vérifier* de la requête *Signature Check Request*. Ce champ permet de s'assurer que la vérification de signature a bien été effectuée sur un message non modifié ;
- ce message est protégé par SEND comme un message NDP. Il contient les options

CGA, *horodatage* et *signature universelle*. Cela permet de s'assurer de l'authenticité et l'intégrité du message envoyé par le notaire. Le choix du type d'algorithme de signature est effectué en fonction de l'option *Supported Signature Algorithm* contenue dans la requête *Signature Check Request*.

Échange de messages

À la réception d'un message *Signature Check Request*, le notaire procède aux tests suivants :

- Vérification de l'adresse CGA, les options anti-rejeu et l'option *signature universelle* de l'initiateur de la requête (si possible) ;
- Vérification de l'adresse CGA et de l'option *signature universelle* du *message SEND à vérifier*. Une vérification du champ *horodatage* peut être effectuée au préalable afin éviter les attaques par saturation du routeur.

Si la vérification de la CGA de l'initiateur de la requête est valide, le notaire lui répondra par un message *Signature Status* afin de lui indiquer le statut du *message SEND à vérifier*.

Dans le but d'éviter la saturation du notaire, un nœud ne peut effectuer qu'un certain nombre de requêtes de vérification par seconde. Au delà de cette limite, le notaire informe le nœud dans la réponse *Signature Status* qu'il n'a pas traité sa requête et qu'il doit réessayer plus tard.

Il est à noter qu'un nœud ne disposant pas d'adresse CGA mais implémentant les vérifications de SEND est également capable d'interroger un notaire afin de connaître la validité d'une signature.

5.3.5 Vulnérabilités des CGA multi-clés

Pour que la solution de CGA multi-clés soit une alternative viable, elle ne doit pas dégrader le niveau de sécurité de la solution CGA actuelle. Notons que si une CGA générée à partir de différents types de clés voit une seule de ses clés divulguée à un attaquant, elle devient vulnérable. De manière générale, si l'un des algorithmes de signature venait à être cassé ou si l'une des fonctions de hachage devenait moins robuste aux collisions, il serait alors possible d'usurper des CGA par l'utilisation de l'un de ses algorithmes/fonctions plus faibles. Cette attaque est nommée "*bidding down*". Pour éviter ce type d'attaque, nous supposons que lorsqu'un algorithme de signature ou une fonction de hachage devient inutilisable, il est alors impératif d'en interdire son usage, c'est-à-dire, de considérer les messages ND protégés avec l'un d'entre eux comme non sécurisés. À cet effet, nous considérons que chaque nœud dispose d'une politique locale qui autorise ou interdit chacun des algorithmes de signature.

5.4 Version allégée de la Signature Algorithm Agility pour la normalisation

Dans les sections précédentes, nous avons proposé d'importantes modifications du protocole SEND. Ces modifications ont été jugées trop profondes pour être normalisées car elles nécessitent une ré-évaluation de la sécurité des CGA (lors de l'utilisation des CGA multi-clés). De plus, certains compromis peuvent être effectués sur la rétrocompatibilité et l'interopérabilité. Ainsi, certaines conditions sont réévaluées :

- seuls les nœuds qui ont des CGA basées sur le même type de clé publique doivent pouvoir communiquer ensemble ;
- si, lors d'un échange de messages protégés par SEND, l'un des deux nœuds ne peut vérifier la signature de son correspondant, alors le message est traité comme un message ne disposant d'aucune protection (voir Section 3.5) ;
- optionnellement, si deux nœuds utilisent des CGA basées sur des clés publiques de types différents et sont capable de vérifier la signature émise par leur correspondant, ils peuvent communiquer.

Cela implique les modifications suivantes par rapport à la solution précédente :

- les multi-clés CGA ne sont plus utilisées. L'extension de la structure de données *CGA Parameters* nommée *Public Key extension* est supprimée ;
- il s'ensuit que le champ *position* qui indiquait la position de la clé dans l'option *Signature Universelle* (Figure 5.1) est supprimé. Le champ *Signature Type Identifier* est conservé ;
- comme la compatibilité entre tous les types de nœuds n'est plus primordiale, le rôle de notaire est supprimé ;
- l'option *Supported Signature Algorithm* ne permet plus une négociation des algorithmes utilisés mais est utilisée à des fins de diagnostique en cas de non communication entre les nœuds.

5.5 Synthèse du chapitre

Ce chapitre présente une solution permettant la mise en œuvre du mécanisme de *Signature Algorithm Agility* au sein du protocole SEND. Offrir une telle fonctionnalité est souhaitable dans le but d'améliorer la sécurité et les performances et d'offrir une meilleur flexibilité du protocole. Afin de satisfaire des critères d'interopérabilité, nous avons défini de nouvelles options et de nouveaux messages pour que différents nœuds SEND utilisant différents couples d'algorithmes de signature et de fonctions de hachage puissent communiquer entre eux.

Une dernière partie de ce chapitre présente une version amoindrie de la solution. Cette solution a été présentée au groupe de travail *CGA and SEND maIntenance* (CSI) assurant la maintenance du protocole SEND au sein de l'IETF. D'abord accueillie avec enthousiasme, puisque correspondant à l'un des objectifs principaux du groupe, notre solution n'a pas été poussée vers la normalisation, faute de moyens humains. Toutefois, les différents documents que nous avons soumis à l'IETF restent valides et pourront servir de base lorsque l'attention se focalisera de nouveau sur ce groupe de travail.

6 NDprotector : implémentation espace utilisateur des CGA et du protocole SEND

Nous présentons dans ce chapitre notre travail sur NDprotector, une implémentation des CGA et du protocole SEND. Ce travail nous permet de vérifier la viabilité des extensions proposées dans le Chapitre 5 concernant le support de la *Signature Algorithm Agility*. La motivation principale de notre effort est due au manque d'extensibilité des solutions existantes qui ne permettent pas d'intégrer facilement nos différentes propositions (par ex. le code pour la *Signature Algorithm Agility*).

Un des points forts de notre implémentation, en plus du support de la *Signature Algorithm Agility*, est la simplicité de la configuration laissée à la charge de l'utilisateur. Ainsi, pour démarrer la protection SEND sur un hôte, il suffit en général d'une seule opération de configuration : ajouter la ou les ancres de confiance par défaut.

Ce chapitre présente les différents aspects de notre implémentation. La Section 6.1 explique les choix techniques que nous avons effectués. Puis, la Section 6.2 présente l'architecture de notre solution ainsi que le banc d'essai sur lequel nous avons effectué nos tests. La Section 6.3 énonce les ajouts que nous avons faits à notre implémentation pour supporter la *Signature Algorithm Agility* allégée. Ensuite, la Section 6.4 montre des tests d'interopérabilité avec plusieurs implémentations existantes. Pour finir, nous indiquons dans la Section 6.5 les points à améliorer pour une meilleure diffusion de notre implémentation.

Cette contribution s'inscrit dans le cadre du projet ANR MobiSEND [1]. Elle permet de tester la conformité de l'implémentation de référence fournie par NTT DoCoMo aux spécifications de SEND et de valider de nouveaux outils facilitant les attaques sur le protocole SEND.

6.1 Choix techniques

La raison initiale de notre projet NDprotector est de développer rapidement une implémentation de SEND [AKZN05]. Celle-ci doit nous permettre de démontrer la viabilité de notre solution de *Signature Algorithm Agility*.

Afin de faciliter notre travail sur l'analyse et la génération des messages SEND, nous décidons d'utiliser *scapy*. Ce qui nous contraint à l'utilisation du langage de programmation orientée objet Python [2], souvent utilisé pour la réalisation de prototypes,

1. http://mobisend.org
2. http://www.python.org

Scapy est un outil permettant la génération, modification, capture et analyse d'une grande variété de messages réseaux. Cet outil, écrit en langage Python, est fortement paramétrable et extensible et a pour philosophie d'offrir un maximum de contrôles lors de l'interprétation et la création des messages. Il peut également être intégré comme un module dans d'autres programmes. Une extension, nommée *scapy6*, étend cet outil et lui rajoute le support jusque là manquant du protocole IPv6. Par la suite, dans le cadre du projet MobiSEND, Arnaud Ebalard (également co-auteur de *scapy6*) a étendu ce dernier afin de lui ajouter le support des messages et options définies dans le protocole SEND ainsi que les manipulations basiques sur les CGA (génération de la structure de données *CGA Parameters*, calcul de *hash2*, etc.). Cette nouvelle branche de projet *scapy* s'appelle *scapy6-send*.

Nous avons intégré le mécanisme de *Signature Algorithm Agility* dans *scapy6-send*. Durant cette étape, la flexibilité de l'outil *scapy* nous a permis de rapidement modifier et étendre le format des messages et options SEND déjà définies.

Nous avons également construit un moteur permettant le traitement des messages NDP entrants et sortants. Pour cette pièce centrale de notre outil, nous avons retenu un modèle similaire à celui utilisé par l'implémentation de NTT DoCoMo. Les paquets entrants correspondant à des messages NDP sont interceptés à l'entrée de la carte réseau (avant d'être remontés aux différentes couches réseaux du système). Pour chacun de ces messages, la vérification des options SEND est effectuée. Si le paquet est valide, le message est ré-introduit dans le système pour être transféré vers les couches supérieures. Dans le cas contraire, des politiques locales définissent le traitement du paquet (détruit ou accepté). Les messages du NDP émis sont également capturés avant d'être émis par la carte réseau (après avoir été formés par les différentes couches réseaux). Les options SEND y sont alors ajoutées.

Sur la majorité des systèmes d'exploitation, ces opérations peuvent être réalisées grâce à des *hooks* (ou hameçons) qui permettent la recopie et le traitement des paquets en espace utilisateur. C'est ce que permet l'architecture *Netgraph* sous FreeBSD. C'est également le cas de la bibliothèque *libnetfilter-queue*[3], sous GNU/Linux, que nous avons utilisée. Puisque nous avons programmé en Python, nous avons employé les *bindings* Python de la bibliothèque *libnetfilter-queue* fournis par Pierre Chifflier[4]. Dans la pratique, le pare-feu *netfilter* se voit ajouter des règles qui redirigent une certaine catégorie de paquets (configurée selon le filtre) vers l'espace utilisateur. Un système de queues permet de séparer le traitement des messages capturés en fonction du filtre déclenché et de leur affecter des traitements différents.

Nous devions également choisir comment réaliser les opérations systèmes : ajout et suppression d'adresses IPv6, ajout de filtres *netfilter*, etc. Puisque le langage Python est à la base un outil pour la création de scripts systèmes, nous avons naturellement choisi de faire appel aux outils présents sur le système comme la commande *ip* fournie par le paquet *iproute2*[5] pour l'ajout d'adresses et la commande *ip6tables* pour la mise en place des filtres d'interception.

3. http://www.netfilter.org/projects/libnetfilter_queue/index.html
4. http://software.inl.fr/trac/wiki/nfqueue-bindings
5. http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2

Pour implémenter la partie routeur dans notre nœud, l'annonce périodique de messages *Router Advertisement* et la réponse aux messages *Router Solicitation* sont laissées à la charge du démon *radvd*[6].

6.2 Implémentation de NDprotector

6.2.1 Architecture des différents composants

La Figure 6.1 illustre l'architecture de nos composants. Notre architecture est composée de plusieurs modules :

- le pare-feu *netfilter* redirige vers les différents modules les messages NDP et les messages CPS/CPA. Cette redirection est configurée par le *module de filtrage* qui met en place les règles sur les différentes interfaces ;
- un *cache de voisinage* local est mis en place. C'est dans celui-ci que sont stockées les informations supplémentaires relatives au voisin comme la valeur du dernier horodatage reçu. Comme nous avons décidé d'autoriser la cohabitation de nœuds sécurisés et de nœuds non-sécurisés (aussi appelée mode "mixte"), la duplication du cache de voisinage, déjà présent à l'intérieur du noyau du système d'exploitation, est nécessaire. Ainsi, nous pouvons garantir qu'un attaquant ne pourra mettre à jour une entrée du cache de voisinage (du noyau) pour un nœud dont l'adresse est protégée ;
- les messages NDP entrants sont envoyés sur le *module de traitement des messages NDP entrants*. Les différentes options de SEND y sont traitées et le message est vérifié (vérification de la CGA, de la signature, etc.). Si le message est valide, alors il est transféré à la *pile réseau*. Pour chaque message accepté, l'entrée correspondante dans le *cache de voisinage* est mise à jour. Si le mode "mixte" est activé, les messages non protégés ou dont la vérification a échoué seront également transmis à la *pile réseau* et modifieront le *cache de voisinage* local, à l'unique condition qu'ils n'écraseront pas une entrée existante dans la table de voisinage. Si le message reçu est un *Router Advertisement*, il peut donner lieu à la création d'une adresse CGA (procédure d'*Autoconfiguration d'Adresse Sans État*) qui sera réalisée par le *module de gestion des adresses CGA* ;
- la génération, l'affectation, la suppression d'adresse CGA sur le nœud s'effectuent au sein du *module de gestion d'adresses CGA*. C'est également ce module qui lie l'adresse CGA à la paire de clés associée et qui inclut la génération de la *signature numérique* ;
- les paquets NDP sortant sont redirigés vers le *module de traitement des messages NDP sortants*. Le module ajoute les protections SEND aux messages (option *CGA*, option *nonce*, option *horodatage*, option *signature RSA*). Une fois cette opération effectuée, le message est réacheminé vers la carte réseau ;
- le *module de traitement du chemin de certification* gère la partie concernant le chemin de certification. Si le nœud est un routeur, le module reçoit les requêtes

6. http://www.litech.org/radvd/

CPS et émet les réponses CPA associées. Si le nœud est un hôte, il traite les réponses CPA reçues, les valide et les stocke dans le *cache de certificats*.

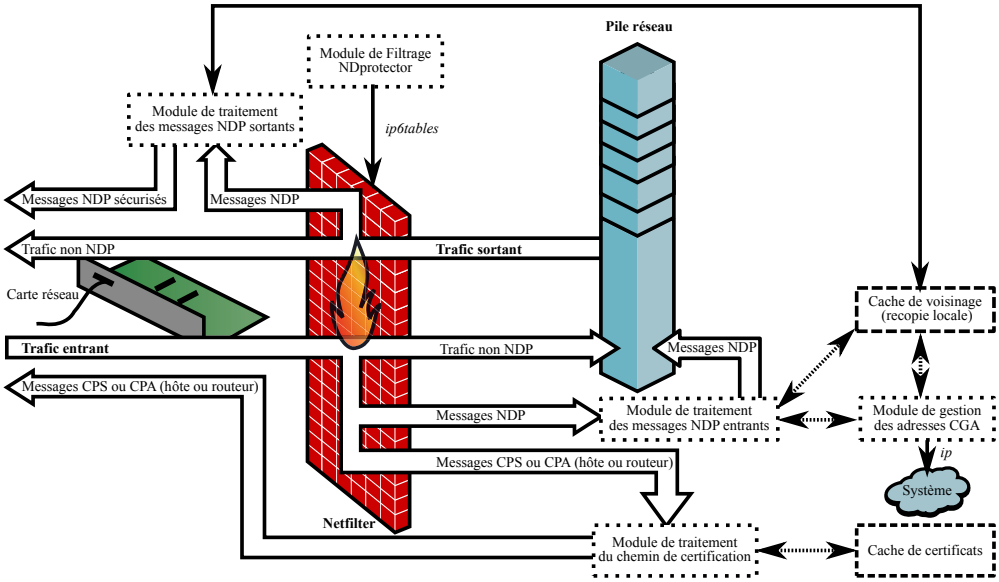


FIGURE 6.1: Architecture de l'implémentation NDprotector

Le langage Python autorise la programmation orientée objet et la séparation du code en modules. L'architecture présentée Figure 6.1 se retrouve alors dans notre code où chaque module est stocké dans un fichier différent.

D'un point de vue méthodologie, chaque module dispose dès sa conception de tests unitaires. Ceux-ci sont effectués grâce à l'outil *nose*[7]. Ils nous permettent de détecter très rapidement des régressions dans le code et le logiciel de gestion de version git[8] nous permet de déterminer quels changements récents sont à leurs origines.

6.2.2 Embryon de système de greffons

Comme nous le verrons dans le Chapitre 8, certains de nos tests nécessitaient de modifier fortement le comportement de l'implémentation. Nous souhaitions être capables de fournir ces modifications à l'utilisateur et lui laisser la possibilité d'activer ou non ces fonctionnalités.

L'ajout d'un système de greffons s'est tout naturellement imposé. Dans la pratique, nous avons rajouté des *hooks* dans chaque module afin de pouvoir étendre certaines fonctions clés.

7. http://somethingaboutorange.com/mrl/projects/nose/
8. http://git-scm.com/

Notre système de plugins repose sur un jeu de capacités (ou *capabilities*) : chaque capacité correspond à l'ajout d'une ou plusieurs fonctions au sein d'un module spécifique. Par exemple, la capacité *filtering* indique que le plugin modifie le module de filtrage. Un plugin peut disposer de plusieurs capacités, lui permettant d'ajouter des fonctionnalités dans autant de modules que nécessaire.

Le système de greffons ne prévoit des *hooks* et des capacités que dans les modules où nous avons eu besoin de modifier précédemment le comportement. Le système de greffons s'étendra alors au fur et à mesure de la création de nouveaux greffons.

6.2.3 Distribution et packaging

Le code source de NDprotector est disponible sur un site web publique [NDp]. Nous avons appliqué la licence BSD à notre code. Ce choix permet à n'importe quel individu de copier, modifier et redistribuer notre implémentation. Nous espérons que ce choix permettra à terme l'inclusion de contributions extérieures.

Les archives incluent l'extension *scapy6-send* qui n'a, au moment de l'écriture de ces lignes, pas encore été intégrée dans la branche principale du projet *scapy*.

Pour une installation facilitée, nous avons également généré des paquetages pour les distributions Linux Gentoo, Debian et Ubuntu.

Paradoxalement, même si après l'installation des paquets la configuration de NDprotector est très simple, l'implémentation n'est pas encore prête à être déployée et manque actuellement de scripts d'initialisation.

6.2.4 Description du banc d'essai

Pour tester notre implémentation, nous avons mis en place un banc d'essai nous permettant de tester les différentes combinaisons de types de nœuds. La Figure 6.2 illustre la disposition du banc d'essai et indique les connexions entre les différents nœuds. Ce banc d'essai est composé de :

- deux nœuds fonctionnant sous GNU/Linux configurés en mode hôte (hôte 1 et hôte 2), pour confirmer le bon fonctionnement des interactions d'hôte à hôte ;
- un nœud fonctionnant sous GNU/Linux configuré en mode routeur (routeur 1), pour tester les fonctionnalités avancées de SEND, comme la découverte de chemin (échange de messages *Certificate Path Solicitation*/*Certificate Path Advertisement*). Celui-ci repose sur le démon *radvd* pour le traitement des messages RS et l'émission périodique de messages RA ;
- un routeur Cisco placé en coupure nous permet de rester connecté au réseau de Télécom SudParis tout en bloquant les paquets IPv6 dans les deux sens (routeur Cisco). Cela permet d'éviter que le réseau de Télécom SudParis (disposant de l'IPv6) n'interfère avec nos tests mais également que nos tests ne perturbent pas nos voisins sur le réseau de Télécom SudParis.

Le nœud assurant la fonction de routeur est connecté à l'Internet IPv6 via l'utilisation d'un tunnel IPv6 (encapsulation UDP). Ce tunnel est fourni par le broker *SixXS* [9].

9. http://www.sixxs.net/

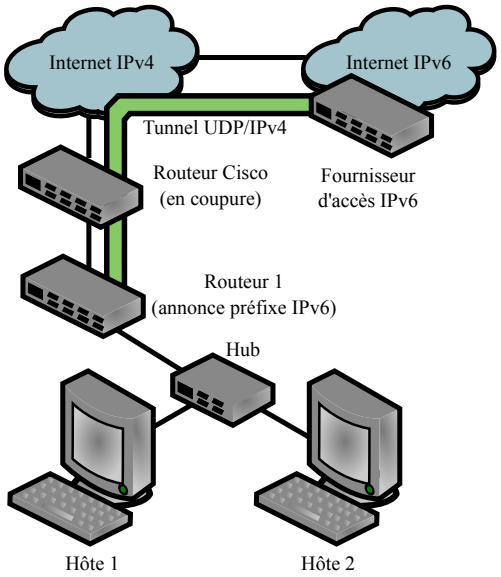


FIGURE 6.2: Configuration du banc d'essai

	hôte "mixte"	hôte non "mixte"	routeur "mixte"	routeur non "mixte"
hôte "mixte"	oui	oui	oui	oui
hôte non "mixte"	oui	oui	oui	oui
hôte non SEND	oui	non	oui	non
routeur non SEND	oui	non	-	-

TABLEAU 6.1: Échanges de messages testés sur le banc d'essai ("oui" indique un succès de l'échange ; "non" indique un échec)

Il nous permet d'annoncer via le démon *radvd* des *préfixes sous-réseau* routables et de vérifier la connectivité des hôtes à plus d'un saut. De plus, cela nous permet de nous assurer que les hôtes acquièrent correctement la route vers le routeur par défaut après la procédure d'*Autoconfiguration d'Adresse Sans État*.

Lors de l'utilisation du banc d'essai, nous avons pu vérifier que notre implémentation se comportait correctement. Le Tableau 6.1 montre que nous avons effectué des tests entre chaque type de nœuds. Les échanges se comportent comme prévu : les nœuds "mixtes" communiquent avec les nœuds non protégés par SEND alors que les nœuds non "mixtes" ne communiquent pas avec les nœuds non protégés par SEND.

6.3 Extensions de l'implémentation NDprotector pour gérer la Signature Algorithm Agility

Notre implémentation contient la version allégée de notre solution de *Signature Algorithm Agility*, telle que définie Section 5.4. Elle ne contient donc pas le support des CGA multi-clés, même si celle-ci a été développée par la suite, comme nous le verrons dans le Chapitre 7.

Afin d'intégrer ce support, nous avons ajouté des champs supplémentaires à l'option *signature RSA*. Le *module de traitement des messages NDP sortants* est modifié pour que l'option *Supported Signature Algorithm* soit jointe à chaque message NDP émis. Nous avons étendu notre copie locale du cache de voisinage afin de stocker les informations fournies par l'option *Supported Signature Algorithm*.

Nous avons vérifié la compatibilité des nœuds entre eux et avons obtenu des résultats similaires au Tableau 6.1. La compatibilité entre, d'une part, les nœuds ayant des CGA basées sur ECC et la capacité de vérifier une signature RSA et, d'autre part, les nœuds ayant des CGA basées sur RSA et la capacité de vérifier une signature ECC a également été vérifiée. Nous avons également pu confirmer la faible taille des messages émis par les CGA basées sur ECC.

À noter que pour profiter du support de l'algorithme de signature ECDSA, la bibliothèque OpenSSL présente sur le système doit être une version 1.0 ou supérieure. Lorsque nous avons testé le support ECC des versions précédentes, des bugs critiques empêchant la génération et vérification correcte des certificats X.509 ont été détectés.

6.4 Tests de compatibilités de notre implémentation de la Signature Algorithm Agility

Même s'il est difficile d'évaluer le déploiement des CGA et du protocole SEND, le support est développé et incorporé dans de nombreux produits. Nous avons compilé une liste des implémentations existantes :

- NTT DoCoMo [10] fournissait initialement une version de référence de SEND. Depuis 2008, DoCoMo Labs USA a stoppé la maintenance de l'implémentation. Toutefois, la licence (permissive) BSD qui protégeait le code a permis à certains groupes de continuer à maintenir l'implémentation. C'est notamment le cas du projet MobiSEND qui maintient une version plus à jour [11] de l'implémentation. Cette implémentation fonctionne sous GNU/Linux et FreeBSD ;
- Cisco [12] propose dans certains de ses produits le support de SEND ;
- Juniper [13] inclut également le support de SEND dans certains de ses produits ;

10. http://www.docomolabs-usa.com/lab_opensource.html
11. http://mobisend.org/software.html
12. http://www.cisco.com/en/US/docs/ios/ipv6/configuration/guide/ip6-first_hop_security_ps10591_TSD_Products_Configuration_Guide_Chapter.html
13. http://www.juniper.net/techpubs/software/junos/junos93/swconfig-routing/secure-neighbor-discovery-configuration-guidelines.html

- Alcatel Lucent [14]a également développé une implémentation partielle (CGA et seulement options de *signature RSA*) ;
- Microsoft inclue le support des CGA dans ses systèmes d'exploitations depuis Windows Vista ;
- L'université centrale du Vénézuela propose une implémentation nommée Easy-SEND [CG09] développée en Java, centrée sur la facilité d'installation. Cette implémentation fonctionne sous GNU/Linux ;
- *Huawei Technologies Corp* et l'Université des Postes et Télécommunications de Beijing (BUPT, *Beijing University of Post and Telecommunications*) ont développé une implémentation fonctionnant sur GNU/Linux de CGA et de SEND [15]. La particularité de cette implémentation est d'inclure des modifications dans le noyau Linux pour le support du protocole SEND et des CGA.

Pour vérifier l'interopérabilité de notre implémentation incorporant un support de la *Signature Algorithm Agility*, conçu pour être rétro-compatible avec les spécifications de SEND [AKZN05], et donc les implémentations existantes, nous avons utilisé l'implémentation fournie par NTT DoCoMo (implémentation de référence). Par la suite, nous avons également évalué notre compatibilité vis à vis de l'implémentation de Cisco (dont l'image nous a été gracieusement fournie), ce qui nous a permis d'évaluer la qualité des implémentations industrielles.

6.4.1 Implémentation de NTT DoCoMo USA labs

Le code source de l'implémentation a été légèrement modifié pour gérer l'option SSA. En effet, quand celle-ci est présente, l'implémentation ignore totalement les messages NDP. Ce comportement n'est pas conforme aux spécifications du protocole de Découverte de Voisins [NNSS07], où seules les options non reconnues ne doivent pas être traitées.

Par ailleurs, même si l'implémentation de NTT DoCoMo est l'implémentation de référence elle souffre de quelques limitations et n'est donc pas totalement conforme aux spécifications de SEND [AKZN05]. En effet, l'implémentation ne supporte que partiellement le DAD : un seul message *Neighbor Solicitation* est envoyé, puis l'adresse est assignée, aucune écoute de réponse n'est réalisée et donc aucune collision ne peut être détectée.

	Hôte NDprotector	Routeur NDprotector
Hôte NTT DoCoMo	compatible	compatible
Routeur NTT DoCoMo	compatible	-

Tableau 6.2: Résultats des tests d'interopérabilité avec l'implémentation de NTT DoCoMo

Le Tableau 6.2 présente les résultats des tests d'interopérabilité avec l'implémentation de NTT DoCoMo. Notre implémentation traite parfaitement les messages émis par

14. http://www.ietf.org/mail-archive/web/ipv6/current/msg06059.html
15. http://code.google.com/p/ipv6-send-cga/

l'implémentation de NTT DoCoMo. Plus intéressant, un hôte utilisant l'implémentation de NTT DoCoMo est capable de se configurer grâce aux messages émis par un routeur utilisant NDprotector. Ce test d'interopérabilité est un succès.

6.4.2 Implémentation de Cisco

Le matériel Cisco dont nous disposons n'est pas compatible avec les images firmware qui nous ont été fournies. Pour effectuer nos tests, nous avons alors eu recours à l'usage de Dynamips [16], un simulateur de routeurs Cisco. Celui-ci nous a permis d'utiliser les images incluant le support de SEND. Le routeur simulé est plus lent que le routeur matériel, mais ceci n'affecte pas nos tests d'interopérabilité puisqu'ils n'évaluent pas les performances.

Tout comme l'implémentation de NTT DoCoMo, l'implémentation de Cisco refuse de traiter les messages ayant une option SSA. Toutefois, le code source n'étant pas disponible, nous n'avons pas pu modifier ce comportement et avons alors retiré l'option SSA de nos messages protégés par SEND.

Le Tableau 6.3 présente le résultat des tests d'interopérabilité. Seul le test entre nœuds configurés en mode hôte est positif. En effet, afin de simplifier notre implémentation de la procédure de validation du chemin de certification, nos messages CPS ne contiennent pas l'option *ancre de confiance*, ce qui est autorisé par la spécification de SEND [AKZN05]. Il semblerait que cette option soit requise pour qu'un routeur Cisco réponde à une requête CPS. Puisque nous ne souhaitions pas modifier notre implémentation, l'aspect routeur de l'implémentation de Cisco n'a pu être évalué.

Il est à noter que cette implémentation est vulnérable à l'attaque que nous avons présentée Section 3.6.6 et détaillée dans l'article [CC08]. Lors de la tentative d'assignation de la CGA sur l'interface, un rejeu du message NS de la procédure de DAD suffit à faire échouer l'assignation. Alors que la procédure d'assignation d'adresses proposée par SEND [AKZN05] prévoit 3 essais, ici, aucun autre essai n'est réalisé.

Les différents problèmes identifiés dans cette section ont été reportés à Cisco.

	Hôte NDprotector	Routeur NDprotector
Hôte Cisco	compatible	manque de support de l'option *ancre de confiance*
Routeur Cisco	manque de support de l'option *ancre de confiance*	-

Tableau 6.3: Résultats des tests d'interopérabilité avec l'implémentation de Cisco

6.5 Perspectives d'évolutions

L'objectif initial du projet était de fournir une implémentation "preuve de concept" de notre support de la *Signature Algorithm Agility*. Par la suite, nous avons pris

16. http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator

conscience de l'intérêt que pourrait avoir un tiers à utiliser notre implémentation (configuration simple, facilité d'extension du code, etc.). Cette orientation initiale est la raison pour laquelle notre implémentation manque d'outils pour simplifier l'installation ou de scripts de démarrage. Un manque représentatif est l'absence de script d'initialisation pour lancer NDprotector durant le démarrage d'un nœud.

Nous avons identifié les points à améliorer pour faciliter l'accès à notre implémentation et améliorer sa diffusion :

- nettoyage du code pour améliorer sa qualité et sa robustesse. Une forte majorité du code est déjà factorisée et le travail consiste à vérifier la couverture du code et à identifier les chemins rarement pris pour y rechercher d'éventuels bugs ;
- ajout de scripts d'initialisation, pour permettre un lancement automatique de NDprotector lors du démarrage d'un nœud ;
- amélioration de la qualité de la documentation. La majorité des informations sont déjà disponibles à l'utilisateur (fichier de configuration contenant des commentaires détaillés, etc), mais un travail sur la forme pour faciliter l'accès à la documentation doit être effectué ;
- exploration actuelle de la possibilité d'utiliser les sockets de type *netlink* pour configurer les adresses IP. Les sockets de type *netlink* sont des sockets internes qui permettent de communiquer avec différents éléments de bas niveau du noyau (routage, firewall). Ceci dans le but de diminuer les privilèges nécessaires à l'exécution de NDprotector et ainsi d'améliorer la sécurité ;
- le test d'interopérabilité avec l'implémentation de Cisco en Section 6.4.2 nous a indiqué qu'un travail de notre part doit être effectué pour améliorer la compatibilité avec les implémentations existantes.

6.6 Synthèse du chapitre

Ce chapitre présente l'implémentation NDprotector que nous avons développée durant la thèse. Cette implémentation nous permet de prouver la viabilité de notre proposition allégée concernant la *Signature Algorithm Agility*. Nous avons pu tester la compatibilité de notre implémentation avec les implémentations existantes et ainsi prouver sa conformité.

Comme nous le verrons également dans les Chapitres 7, 8 et 9, cette implémentation nous permet d'appuyer nos travaux connexes à la *Signature Algorithm Agility*.

7 Utilisation étendue de la Signature Algorithm Agility pour améliorer la compatibilité du protocole SEND

Dans le Chapitre 3, nous avons énoncé les incompatibilités qui empêchent l'utilisation simultanée du protocole SEND et d'autres protocoles existants reposant sur des fonctionnalités du protocole de Découverte de Voisins, tels que Proxy Mobile IPv6 (PMIPv6), Mobile IPv6 (MIPv6), Proxy ND, etc. Pour les mêmes raisons, le protocole SEND ne permet pas l'utilisation des adresses anycast, où plusieurs nœuds partagent une même adresse. En effet, la protection de l'adresse fournie par le protocole SEND empêche l'utilisation de l'adresse par un nœud autre que son propriétaire et donc le partage d'adresse par des nœuds tiers. Ce partage est pourtant nécessaire à ces protocoles. Toutefois, assurer la sécurité de l'adresse et permettre le partage de l'adresse sont deux buts qu'il peut être souhaitable de réaliser simultanément. Ce chapitre se propose de résoudre ces incompatibilités en modifiant le protocole SEND et présente les alternatives existantes.

Dans une première partie, nous présentons notre travail pour améliorer la compatibilité du protocole SEND avec la fonctionnalité de *Proxy Neighbor Discovery*. Dans les Sections 7.1.1 et 7.1.2, nous présentons deux modes de fonctionnement du *Proxy Neighbor Discovery*. Ensuite la Section 7.1.3 pointe les incompatibilités entre le protocole SEND et le *Proxy Neighbor Discovery*. Nous étudions dans la Section 7.1.4 les solutions existantes qui améliorent cette compatibilité. Toutefois, nous ne sommes pas entièrement satisfais des solutions actuelles et nous proposons, dans la Section 7.1.5, une nouvelle solution basée sur nos travaux sur la *Signature Algorithm Agility*. Pour finir, nous comparons les avantages et inconvénients de notre solution par rapport aux solutions existantes dans la Section 7.1.7.

Dans une seconde partie, nous expliquons comment améliorer le protocole SEND pour qu'il fonctionne avec l'adressage anycast. Les Sections 7.2.1 et 7.2.2 présentent deux utilisations différentes des adresses anycast. La Section 7.2.3 illustre les incompatibilités entre le fonctionnement du protocole SEND et l'utilisation des adresses anycast. Puis la Section 7.2.4 présente les solutions existantes qui améliorent la compatibilité du protocole. Dans la Section 7.2.5, nous proposons notre solution, basée sur la *Signature Algorithm Agility*, qui a pour avantage de très peu modifier les nœuds tiers. Finalement, la Section 7.3 présente une évaluation des performances que nous avons réalisée.

Ce chapitre contient notre proposition pour améliorer la compatibilité entre le *Proxy ND* et SEND et entre l'adressage anycast et SEND. Cette proposition fait partie de

nos contributions et a été présentée à la conférence *Sécurité des Architectures Réseaux et des Systèmes d'Information* [CL10] (SAR-SSI) en 2010.

7.1 Utilisation de la Signature Algorithm Agility pour assurer la fonctionnalité de Proxy Neighbor Discovery sécurisé

Dans ce chapitre, nous utilisons le terme "Proxy Neighbor Discovery" (*Proxy ND*) pour désigner la fonction offerte par une tierce partie, le proxy, quand il avertit ou modifie les messages ND d'un de ses voisins. Un tel voisin est nommé "nœud protégé" tandis que son adresse est nommée "adresse protégée".

Le terme *Proxy ND* se réfère dans ce chapitre à deux scénarios distincts décrits dans les Sections 7.1.1 et 7.1.2.

7.1.1 Définition du Proxy ND dans les documents du NDP et de Mobile IPv6

La spécification du protocole de Découverte de Voisins [NNSS07] définit le *Proxy ND* comme un mécanisme où un routeur assure la protection de l'adresse d'un nœud lorsque celui-ci se trouve sur un lien différent. Cette spécification décrit ce mécanisme de manière assez large, afin de laisser par la suite de nouveaux protocoles l'étendre à des usages plus spécifiques. Ainsi, il est seulement décrit comment le proxy protège l'adresse du nœud absent (nœud protégé) en répondant aux requêtes du protocole de Découverte de Voisins adressées à celui-ci. Dans la pratique, le proxy rejoint le groupe multicast "du nœud sollicité" pour s'assurer de recevoir ses requêtes. Le proxy pourra alors répondre aux messages *Neighbor Solicitation* par des messages *Neighbor Advertisement* où il placera son adresse de couche liaison dans l'option *adresse couche liaison de la cible*, ce qui lui permettra de se substituer au nœud. Afin de ne pas perturber les annonces émises par le nœud si celui-ci revient sur le lien, les annonces effectuées par le proxy ont le drapeau "*override*" désactivé. Ce qui a pour effet de ne pas écraser une entrée du cache de voisinage déjà existante.

Le *Proxy ND* a plus tard été étendu dans la spécification du protocole de mobilité Mobile IPv6 [JPA04], où un Nœud Mobile demande l'assistance du proxy pour protéger son *adresse mère* (*Home Address*, HoA) quand il quitte le *réseau mère*. La Figure 7.1 illustre un Nœud Mobile présent au sein de son réseau mère lorsqu'aucune gestion de la mobilité n'est encore nécessaire. Le nœud se charge lui-même de la protection de son adresse mère. C'est son routeur par défaut, l'Agent Mère (*Home Agent*, HA) qui lui transfère le trafic reçu depuis l'extérieur du réseau mère (notamment celui du nœud correspondant 2).

Après son départ, illustré Figure 7.2, le Nœud Mobile établit un tunnel avec l'Agent Mère. Ce tunnel permet au Nœud Mobile de continuer à utiliser son Adresse Mère et de poursuivre les connexions déjà établies. Puisque le préfixe sous-réseau du Nœud Mobile est toujours averti par l'Agent Mère, celui-ci reçoit tous les messages destinés à

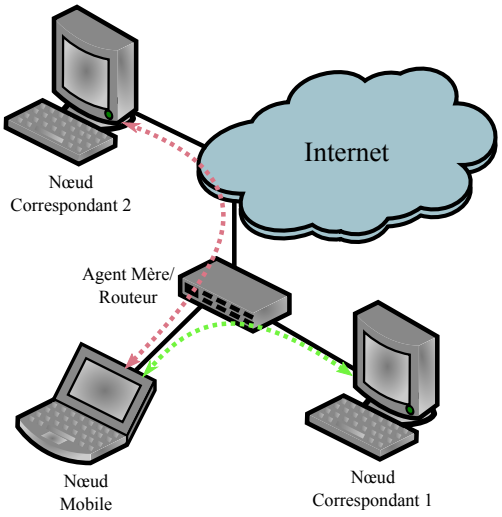


FIGURE 7.1: Exemple d'utilisation de Mobile IPv6 : le nœud mobile est encore présent sur le réseau mère

ce préfixe et donc les messages adressés au Nœud Mobile. L'Agent Mère reroute alors le trafic à destination du Nœud Mobile à l'intérieur du tunnel. Toutefois, les anciens voisins du Nœud Mobile (ici le Nœud Correspondant 1) n'ont pas conscience de la mobilité de leur correspondant et effectuent des procédures de résolution d'adresse pour communiquer avec le Nœud Mobile qu'ils pensent être sur le lien. Pour résoudre ce problème, l'Agent Mère joue le rôle de *Proxy ND* et agit comme s'il possédait l'adresse du Nœud Mobile (qui est ici l'"adresse protégée"). À cet effet, il va émettre des messages *Neighbor Advertisement* pour prévenir les anciens voisins du Nœud Mobile et répondre aux requêtes *Neighbor Solicitation* destinées au Nœud Mobile, en plaçant dans les options *adresse couche liaison de la source* et *adresse couche liaison de la cible* sa propre adresse de couche liaison. Ainsi, l'Agent Mère s'assure de recevoir le trafic local destiné au Nœud Mobile. L'Agent Mère redirige par la suite ce trafic au Nœud Mobile via son tunnel.

7.1.2 Définition du Proxy ND dans le document RFC 4389 (ND Proxies)

À l'origine, le concept de "ND Proxies", défini dans le document RFC 4389 [TTP06], correspond à un besoin simple où un seul préfixe sous-réseau IPv6 est étendu sur plusieurs liens et où les nœuds de ces différents liens doivent pouvoir communiquer, comme illustré Figure 7.3. Dans ce contexte, le proxy relaye les messages du NDP entre les liens et modifie "en vol" le contenu des options *adresse couche liaison de la source* et *adresse couche liaison de la cible* pour y placer sa propre adresse couche

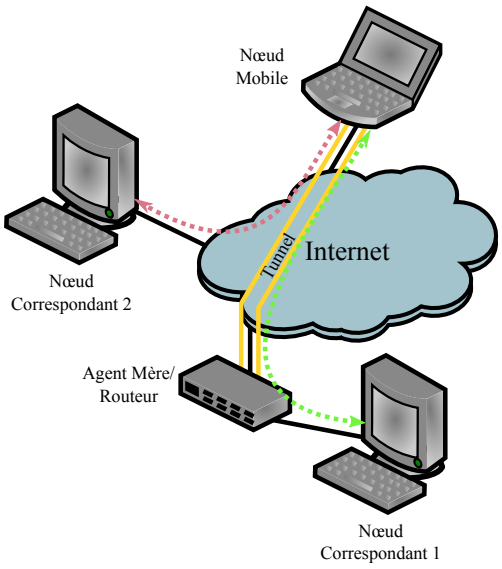


FIGURE 7.2: Exemple d'utilisation de Mobile IPv6 : le nœud mobile a quitté le réseau mère et communique avec ses correspondants grâce à un tunnel

liaison, comme illustré dans la Figure 7.4. Ainsi, les nœuds situés sur les différents liens n'apprennent que l'adresse couche liaison du proxy (par ex. durant la procédure de *résolution d'adresse*). Lorsqu'ils veulent communiquer avec les correspondants situés sur un autre lien, le trafic est alors envoyé au proxy, qui fait suivre les paquets sur le lien où est attaché le correspondant.

Ce comportement est particulièrement utile quand une fonctionnalité de *bridge* (pont) de niveau 3 est requise entre plusieurs liens de technologies différentes (où la mise en place de *bridge* de niveau 2 ne serait pas possible) et quand l'on souhaite éviter d'avoir recours à la traduction d'adresse pour partager un préfixe sous-réseau entre plusieurs liens.

Il est à noter que le proxy agit différemment de la Section 7.1.1 car il peut être amené à modifier des messages *Router Advertisement* si un routeur est situé sur un des liens où le proxy agit. De plus, comme illustré dans la Figure 7.3, plusieurs proxys peuvent partager le même préfixe et être mis en chaîne afin d'augmenter le nombre de liens où le préfixe sous-réseau est partagé.

7.1.3 Incompatibilité entre le Proxy ND et SEND

La Section 7.1.1 présente une variante du *Proxy ND* qui émet des messages à la place du nœud protégé pour avertir son adresse. Si le nœud protégé utilise SEND, alors le proxy doit également utiliser SEND pour protéger ses messages. Dans le cas où le proxy

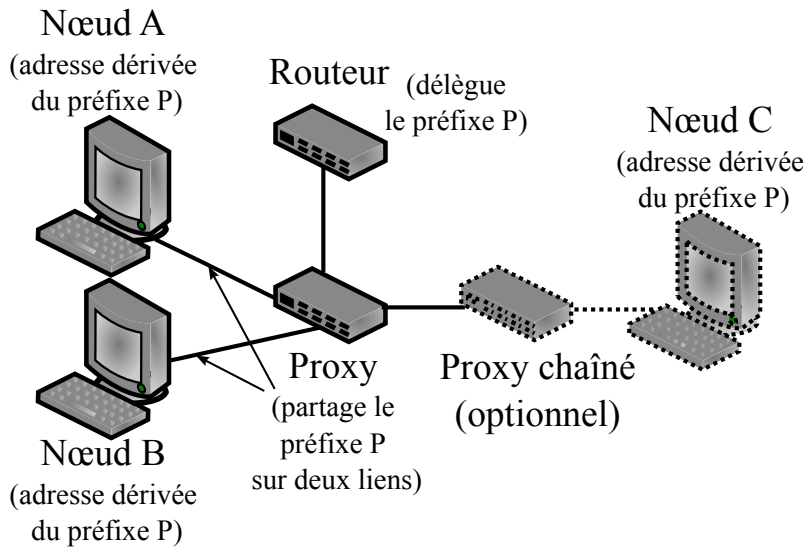


FIGURE 7.3: Architecture du proxy implémentant le document RFC 4389 [TTP06]

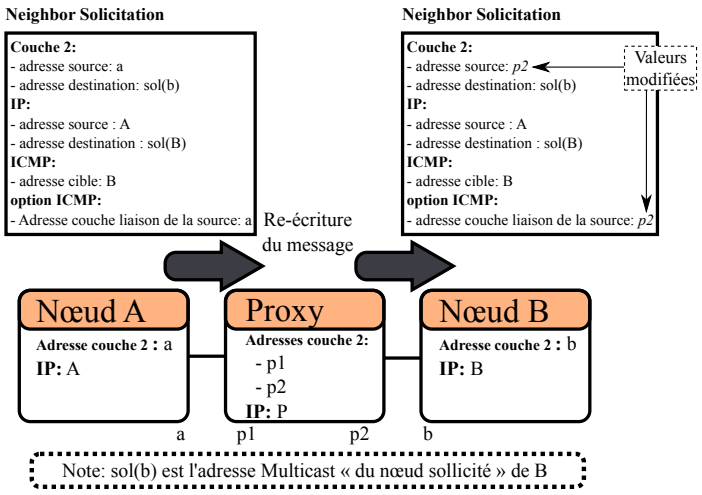


FIGURE 7.4: Intervention d'un proxy RFC 4389 durant une procédure de *Résolution d'Adresse*

ne pourrait utiliser SEND pour protéger ses messages, aucune entrée dans le cache de voisinage des correspondants du nœud protégé ne serait mise à jour. Toutefois, le proxy doit avoir connaissance de la clé privée du nœud afin de pouvoir générer une signature valide et prouver la possession de l'adresse. Pour des raisons de sécurité, partager une clé privée entre plusieurs nœuds paraît peu concevable.

La seconde approche du *Proxy ND* que nous avons présentée dans la Section 7.1.2 modifie les messages en vol pour y placer son adresse de couche liaison. Si les messages sont protégés avec le protocole SEND, cette modification fausse la signature et fait par la suite échouer la vérification du message sur le nœud correspondant. Cette signature ne peut pas être recalculée puisque le proxy ne dispose pas de la clé privée du nœud. De plus, le nœud protégé n'a pas obligatoirement connaissance de la présence du proxy. Ainsi, il ne peut pas prévoir que celui-ci va avertir son adresse, contrairement au cas où le nœud utilise Mobile IPv6 et sait quand il n'est plus attaché au réseau mère. Il ne peut, dans ce cas, effectuer aucune action pour faciliter la tâche du proxy.

Nous venons de voir que les deux variantes du mécanisme de *Proxy ND* présentent des incompatibilités avec le modèle de sécurité de SEND. D'où le besoin, comme nous le verrons dans la section suivante, d'adapter le protocole SEND à la fonctionnalité de *Proxy ND*.

À noter que le document [CKD10] fournit plus de détails techniques sur les incompatibilités entre la fonction de *Proxy ND* et le protocole SEND.

7.1.4 Les solutions actuelles et leurs limitations

Comme nous l'avons vu précédemment, l'incompatibilité entre SEND et le *Proxy ND* est handicapante pour les protocoles qui se basent sur le *Proxy ND* et qui souhaitent protéger leurs adresses. C'est pourquoi quelques propositions de modification du protocole SEND permettant de le faire cohabiter avec le *Proxy ND* sont apparues. Cette section recense ces solutions et présente leurs avantages et leurs limitations.

Certains aspects, comme l'anonymat, sont également abordés. S'ils paraissent revêtir un intérêt mineur de premier abord, ils ont en réalité de grosses conséquences sur la sécurité d'une solution. Ainsi, dans Mobile IPv6 [JPA04], un attaquant peut saturer le lien qui relie le nœud mobile au réseau mère quand l'attaquant apprend que l'adresse de ce dernier est protégée par l'agent mère (proxy). D'où l'intérêt d'avoir une solution qui rend les messages ND du nœud et du proxy indistinguables.

7.1.4.1 Support du Proxy ND sécurisé pour SEND

Le document [KLBGM10] propose d'étendre le protocole SEND [AKZN05] en ajoutant un nouveau type d'option de signature nommée *Signature du Proxy* et en utilisant les attributs *Extented Key Usage* (EKU) des certificats X.509 [CSF+08] de chaque routeur SEND pour expliciter le rôle de proxy (tel que défini dans [GKK10]). Cette option n'est utilisée que par le proxy lorsqu'il protège le nœud. Ainsi, la solution ne peut pas fournir l'anonymat. Le format de l'option est similaire à l'option de *Signature RSA*. Seul le sens du champ *Signature Numérique* est modifié. Il est maintenant généré à partir de la clé privée du proxy. Pour la vérification, le proxy agissant comme routeur

sur le réseau envoie des messages *Router Advertisement* (spontanés dans le cas d'un attachement à un lien), qui déclenchent la procédure de *Certificate Path Solicitation* sur les nœuds qui n'ont pas déjà enregistré la clé Publique du proxy). Par la suite, ces nœuds vérifient le certificat du proxy et y apprennent la clé publique de ce dernier. Ils peuvent dès lors commencer à vérifier les messages protégés. Quand chacun de ces nœuds connaît la clé Publique d'un proxy qui est dans son voisinage, le chaînage des proxys est possible.

L'option *Signature du Proxy* permet de s'affranchir de la vérification de l'adresse CGA (les messages protégés ne transportent pas d'option CGA). La proposition n'apporte, par ailleurs, aucune modification sur la spécification des CGA [Aur05]. En revanche, elle impose une modification de tous les nœuds afin qu'ils puissent vérifier la signature du proxy. Le déploiement des certificats, comme pour SEND [AKZN05], n'est pas spécifié mais ne change que par la présence d'une extension *EKU* indiquant le rôle de *proxy*.

Le désavantage de la solution en terme de sécurité est l'augmentation du rôle du routeur assurant la fonction de proxy qui devient de plus en plus central : il devient une cible de choix pour un attaquant. On notera ainsi qu'un proxy compromis peut établir une nouvelle attaque de type "homme du milieu". Le document [NKN04] décrit une attaque qui reste valide en utilisant SEND. Un routeur compromis peut siphonner le trafic du réseau en utilisant des messages RA dont les options *Prefix Information* (PIO) indiquent de ne pas communiquer sur le lien (drapeau *L* à 0). Cela a pour effet de rediriger le trafic des nœuds utilisant les préfixes annoncés vers le routeur. Seules les adresses dérivées du préfixe Lien-Local (fe80::/64) ne sont pas concernées par cette attaque (car les PIO ne peuvent annoncer le préfixe Lien-Local). La solution *Secure Proxy ND* autorise une nouvelle attaque complétant la précédente. Un proxy peut être autorisé à protéger des adresses de Lien-Local via son certificat. Dès lors, un proxy corrompu peut s'insérer durant la phase de résolution d'adresse de n'importe quel nœud (qu'il requiert ou non une protection) afin de rediriger le trafic vers lui même. Il peut alors monter efficacement une attaque "homme du milieu" sur le préfixe Lien-Local. Des certificats, contraints à des préfixes de sous-réseau de taille 128 bits (une adresse), peuvent néanmoins être utilisés plus finement pour contrer cette faiblesse.

Ce désavantage est contrebalancé par un grand avantage intrinsèque des solutions basées sur les certificats : la possibilité d'inclure une date de péremption dans le certificat et de le révoquer. Cela permet d'annuler les droits d'un proxy compromis.

Un autre aspect de l'utilisation des certificats dans cette solution est de rajouter des proxys dans le réseau après que les nœuds aient été déployés et configurés pour peu que ceux-ci aient été fournis avec la même ancre de confiance que celle des certificats des proxy.

Cette solution est en cours de normalisation par le groupe de travail "CGA and SEND maIntenance" (CSI) au sein de l'IETF.

7.1.4.2 Délégation de droits pour l'annonce ND

Le document [CKD10] catalogue les différentes approches pour permettre la fonction de proxy ND conjointement à l'utilisation de SEND [AKZN05]. Parmi celles-ci,

l'approche par "délégation d'autorisation" (*Authorization Delegation*) abordée dans le document [NA02] propose à un nœud de désigner un proxy et de lui déléguer le droit de signer les messages de signalisation ND.

Cette approche peut être une simple forme de signature d'éléments publics du proxy de la part du nœud protégé ou peut être plus complète en proposant un certificat généré par le nœud et signé avec sa clé privée. L'approche par certificat permettant de limiter la durée de vie du certificat et/ou de le révoquer (il reste encore à déterminer ici comment les nœuds pourront contacter la liste de révocation pour les certificats auto-signés par le nœud). Afin de faire confiance à un proxy, on peut imaginer que celui-ci dispose d'un certificat indiquant son rôle comme pour la solution précédente [GKK10].

Cette solution, moins centralisée que la précédente, diminue l'intérêt que peut avoir un attaquant à corrompre un proxy, celui-ci ne pouvant nuire qu'aux nœuds l'ayant autorisé au préalable. Elle semble n'entraîner que des modifications sur SEND [AKZN05] et permet d'ajouter de nouveaux proxys une fois l'adresse configurée.

Le chaînage des proxys est possible lorsque l'on autorise les différents proxys à se re-déléguer les droits qu'ils ont obtenus des nœuds qu'ils protègent. On peut ainsi obtenir un certificat émanant d'un nœud qui autorise un proxy qui lui même signe un certificat pour un autre proxy, lui accordant aussi les droits d'agir comme proxy pour le nœud initial.

7.1.4.3 Solution à base de signature en anneau

Les solutions présentées Sections 7.1.4.1 et 7.1.4.2 n'offrent pas la confidentialité de la localisation. Pour pallier ce problème, les auteurs des documents [KWRG06] et [KG05] proposent une solution basée sur l'algorithme de "signature en anneau" Rivest-Shamir-Tauman (RST) [RST01]. La "signature en anneau" diffère de la signature de groupe par l'absence de manager de groupe (dont le rôle est central car il génère puis distribue la clé) et offre l'anonymat à celui qui signe le message. C'est à notre connaissance la seule solution qui propose l'anonymat [1] de l'entité qui signe les messages ND.

Le document [KG05] explicite le côté technique de la solution : celle-ci modifie sur tous les nœuds (même ceux non protégés) le comportement par défaut de SEND [Aur05] mais aussi, bien que de façon mineure, l'utilisation des CGA [AKZN05]. Une adresse, maintenant nommée "multi-key CGA" (ou M-CGA), est créée à partir de la clé Publique RSA du nœud à protéger et de celle(s) du ou des proxys. Cela est possible suite à un échange de CPS/CPA entre le nœud et le(s) proxy(s) où le nœud protégé apprend le certificat du/des proxys et où la clé de chaque proxy est extraite à partir du certificat. C'est en choisissant de faire confiance à un certificat pour former une M-CGA que le nœud autorise le proxy à agir comme tel. De ce point de vue, il est donc du ressort du certificat, à l'image de la solution présentée en section 7.1.4.1, de prouver que le routeur qui assure la fonction de proxy a été autorisé par une entité de confiance à agir comme tel. On notera par ailleurs que la taille de la clé Publique RST ainsi que la taille de la signature sont linéaires en fonction du nombre de membres/clés

1. anonymat toutefois relatif puisque l'entité qui signe prouve qu'elle fait partie du groupe.

(et donc du nombre d'entités autorisées à protéger le nœud).

L'article [KWRG06] offre une comparaison des performances entre l'utilisation de la signature RSA et l'utilisation de RST. Lorsque deux membres composent le groupe (le nœud protégé et le proxy), la génération de signature RSA et RST est de rapidité équivalente. La vérification de signature, elle, est deux fois plus lente lors de l'utilisation de l'algorithme RST. Dans le cas général, le ralentissement est peu pénalisant. Par la suite, lorsque le nombre de membres dans le groupe augmente, comme prévu, la durée nécessaire pour la signature/vérification augmente linéairement.

D'un point de vue configuration, aucune configuration n'est nécessaire à condition que le nœud soit au préalable averti qu'il doit construire une M-CGA. C'est le cas lors de l'utilisation de MIPv6 [JPA04]. Dans le cas de l'utilisation du Proxy ND type RFC 4389 [TTP06], une configuration manuelle du nœud sera plus probable, car celui-ci n'aura aucun moyen a priori de savoir qu'il se trouve derrière un proxy et qu'il doit générer une M-CGA. La possibilité de mise en chaîne sera problématique, puisque le nœud de manière standard n'apprend pas la clé Publique des routeurs qui sont situés à plus d'un saut.

À noter toutefois que même si l'algorithme RST fournit l'anonymat du signataire, elle ne protège pas la fuite d'information via l'émission de messages de type ND. Ainsi, il est possible en observant un changement dans l'option *Source Link-Layer Address* et *Target Link-Layer Address*, de déterminer que le proxy commence à protéger une adresse (redirige le trafic du nœud vers lui même) et donc que le nœud mobile vient de quitter le réseau.

Nous soulignons également qu'il s'agit de la seule solution à avoir jusque là fait l'objet d'une implémentation.

7.1.4.4 Utilisation de relation "symbiotique" pour protéger le Proxy ND

Le document [HN09] décrit la création d'une relation "*forte mais distante*" que les auteurs appellent relation symbiotique (*Symbiotic Relationship*). Cette relation est décrite comme une relation unidirectionnelle entre deux nœuds A et B. En créant cette relation, A permet à B de prouver "sa relation" avec A. Cette preuve permet ainsi à B de prouver qu'il a une autorisation de A à une tierce partie (ici, l'autorisation d'agir comme proxy).

D'un point de vue technique, la procédure de génération de CGA décrite dans [Aur05] est légèrement modifiée, restant compatible avec les nœuds implémentant le protocole CGA "standard" [Aur05]. Le champ *Modifier* qui est normalement un nombre de 128 bits généré de manière aléatoire, est ici généré à partir des 128 premiers bits du hash d'une ou plusieurs clés publiques et d'un nombre aléatoire. Ces clés publiques appartiennent aux routeurs d'accès que le nœud autorise à agir comme proxy. Elles sont apprises lorsque le nœud rejoint le lien, dans les messages de type RA sécurisés.

Par la suite, le protocole SEND [AKZN05] est modifié pour qu'un nœud souhaitant prouver sa relation signe les messages avec sa clé Privée et joint au message l'ensemble des clés Publiques et le nombre aléatoire utilisés pour la création de l'adresse. Ces derniers paramètres permettent au nœud recevant le message de re-dériver l'adresse CGA du nœud protégé et de vérifier la relation entre le proxy et le nœud protégé.

Bien que le document soit assez vague à ce sujet, il semble que la modification apportée au mécanisme de CGA n'est pas totalement transparente. En effet, comme décrit dans la Section 3.1.1, le *Modifier* a initialement une valeur aléatoire par la suite incrémentée afin de construire une valeur de *hash2* respectant la valeur du paramètre SEC. Lorsque SEC vaut 0, l'approche actuelle ne pose pas de problème, mais lorsque SEC vaut 1, il n'est plus possible de construire l'adresse comme indiqué. L'incrémentation du *Modifier* casserait le lien avec la ou les clés Privées des proxys. Une solution que nous proposerions serait de stocker ce nouveau *Modifier* contenant la/les clés Publiques dans un champ d'extension de la structure de données *CGA Parameters*. Cette solution rendrait la procédure transparente et n'occuperait que 20 octets supplémentaires.

La solution propose également une forme d'anonymat assez faible en proposant au nœud de cacher sa présence en laissant le proxy protéger son adresse, même lorsqu'il se trouve sur le lien. Cette forme d'anonymat, comme celle de la solution précédente, est vulnérable à un type de surveillance des messages ND visant à capturer des variations d'adresses de couche 2.

7.1.5 Solution basée sur la Signature Algorithm Agility

Nous proposons ici une nouvelle approche, basée sur la *Signature Algorithm Agility* telle que décrite dans le Chapitre 5. Comme nous l'avons présenté, la principale motivation pour la *Signature Algorithm Agility* est le besoin de pouvoir migrer vers de nouveaux algorithmes lorsque RSA et SHA-1, actuellement intégrés en dur dans CGA [Aur05] et SEND [AKZN05], ne sont pas adaptés (problèmes de sécurité récents de SHA-1, lourdeur de RSA,...).

Nous pensons ici qu'il est possible de réutiliser plusieurs aspects de nos travaux sur la *Signature Algorithm Agility* :

- les adresses CGA ont été étendues afin qu'il soit possible de stocker plusieurs clés publiques supplémentaires dans la structure de données *CGA Parameters* ;
- nous avons étendu l'option *Signature RSA* du protocole SEND que nous avons renommée option *Signature Universelle*. Celle-ci peut désormais, grâce à un pointeur qui désigne une clé publique stockée dans la structure de données *CGA Parameters*, indiquer la clé utilisée pour réaliser la *signature numérique*.

Ici, nous réutilisons la possibilité de stocker plusieurs clés publiques dans la CGA et nous proposons que l'une de ses clés appartienne au nœud et qu'une ou plusieurs autres appartiennent à un ou plusieurs proxys.

L'apprentissage de la clé publique des proxys se fait durant l'*Autoconfiguration d'Adresse Sans État*. Le nœud envoie un message RS à destination de tous les routeurs. Ceux-ci répondent avec un message RA. À la réception de ces messages, le nœud va déclencher la procédure de *Certificate Path Validation*. Après un bref échange CPS/CPA, le nœud peut décoder le certificat de chaque routeur où se trouve indiquée, pour les routeurs assurant également le rôle de proxy ND, l'extension EKU correspondante [GKK10]. La clé publique de chaque proxy valide est extraite du certificat, puis est stockée dans un champ d'extension de la structure de données *CGA Parameters* [CLMSV09]. L'adresse CGA est ensuite construite selon le mécanisme standard

proposé dans [Aur05]. Par la suite, le nœud peut signer normalement son adresse en utilisant la clé publique qu'il possède et le proxy peut faire de même en utilisant la clé privée associée à la clé publique de son certificat.

L'intérêt principal de notre solution réside dans la réutilisation de la *Signature Algorithm Agility*. En effet, les modifications au niveau des nœuds à protéger et des proxys sont mineures et concernent principalement la création de l'adresse. Et pour les nœuds tiers ne nécessitant pas de protection, il n'y a aucune modification à effectuer pour qu'ils puissent vérifier les messages émis par les proxys et par les nœuds protégés.

De plus, les performances peuvent bénéficier de l'utilisation d'ECC, comme présenté en [CBLM09] et [CBL10]. Notre approche ne souffre pas de problème de performance quand le nombre d'entités autorisées à agir comme proxy augmente, contrairement à la solution proposée en 7.1.4.3.

7.1.6 Limitations de la solution

Notre solution se base sur les CGA multi-clés pour stocker plusieurs clés publiques. Cela implique que chaque message NDP sécurisé par SEND inclut une *option CGA* qui contient plusieurs clés publiques. D'où l'augmentation de la taille des messages avec le nombre de clés publiques. Néanmoins, contrairement à la solution [KWRG06], la taille de la *signature numérique* reste fixe. Toutefois, il existe une limite supérieure au nombre de clés publiques qui peuvent être stockées dans la structure de données *CGA Parameters*. Nous avons estimé cette limite à environ onze clés quand nous utilisons des clés ECC de 256-bits. Ajouter de nouvelles clés publiques pourrait forcer à fragmenter le message ND, avec les effets de bord qui en résultent, puisque la valeur minimale de la MTU en IPv6 serait atteinte (1280 octets). Néanmoins, nous sommes confiants que dans des scénarios réels, les nœuds n'auront pas besoin d'autoriser dix proxys simultanément.

Un autre élément à considérer est le niveau de sécurité d'une CGA multi-clés. En effet, celui-ci dépend du niveau de sécurité de la clé publique du nœud et du niveau de sécurité de la ou des clés publiques des proxys. Comme nous l'avions déjà noté Section 5.3.5, la plus faible des clés publiques détermine le niveau de sécurité global de la CGA multi-clés. Nous conseillons, quand cela est possible, que tous les nœuds d'un même domaine administratif soient déployés avec des clés publiques de même type et de même taille. D'un point de vue plus global, un nœud doit rejeter la clé publique d'un proxy qui aurait un niveau de sécurité inférieur à la sienne.

Finalement, contrairement aux solutions présentées dans les documents [KLBGM10] (Section 7.1.4.1) et [NA02] (Section 7.1.4.2), nous ne pouvons pas ajouter ou supprimer de proxys après que l'adresse CGA ait été générée. La liste des proxys autorisés étant figée en dur dans la structure de données *CGA Parameters*. Néanmoins, il est possible de stocker plusieurs clés publiques inutilisées dans l'adresse CGA lors de sa création. Ensuite, lorsque de nouveaux proxys apparaissent, il suffit de transmettre à chacun de ceux-ci une des paires de clés disponible (liée à une des clés publiques stockée dans l'adresse). Cela entraîne une complexification du protocole que nous ne souhaitons cependant pas. Cette limitation est particulièrement handicapante pour les connexions de longue durée.

Solution	le nœud autorise le proxy	modifié pour la solution	modification de nœuds tiers	anonymat	mise en chaîne des proxys	autorisation après auto-configuration
Proxy ND sécurisé	non	SEND	pour les messages émis du proxy	non	possible	possible
Approche par délégation d'autorité	oui	SEND	pour les messages émis du proxy	non	possible	possible
Utilisation de la signature en anneau	oui	CGA et SEND	pour tous les messages	oui	difficile	impossible
Lien symbiotique	oui	CGA et SEND	pour les messages émis du proxy	oui	difficile	impossible
Basé sur la *Signature Algorithm Agility*	oui	rien si support de la *Signature Algorithm Agility*	aucune quand ils gèrent la *Signature Algorithm Agility*	non	difficile	impossible

TABLEAU 7.1: Comparaison des différentes solutions

7.1.7 Positionnement de la solution basée sur la Signature Agility par rapport aux autres solutions

Le Tableau 7.1 présente une synthèse des propriétés et des fonctionnalités des différentes solutions et nous permet de situer notre solution.

Nous proposons, comme les solutions des Sections 7.1.4.2, 7.1.4.3 et 7.1.4.4, une solution décentralisée, où l'autorisation d'agir comme proxy est dépendante du nœud (c.-à-d. le nœud autorise le proxy). Pour nous, il s'agit d'un critère important qui diminue sensiblement l'intérêt que peut avoir un attaquant à corrompre le proxy et supprime l'attaque "homme du milieu" présentée dans la section 7.1.4.1. En supposant que la *Signature Algorithm Agility* soit déployée sur les nœuds, il n'y a pas besoin de modifications lourdes sur les mécanismes CGA et SEND contrairement aux solutions vues en Sections 7.1.4.3 et 7.1.4.4. De plus, si les nœuds tiers (nœuds voisins autre que proxys ou nœuds protégés) supportent la *Signature Algorithm Agility*, ils ne nécessitent aucune modification. Comme la mise en place de l'autorisation du proxy est effectuée durant l'*Autoconfiguration d'Adresse Sans État*, notre proposition ne requiert pas la création de messages de signalisation supplémentaires, au contraire de l'approche basée sur la délégation d'autorité (section 7.1.4.2).

Notre solution n'est pourtant pas exempte de défauts. Parmi ceux-ci, relevons qu'elle ne supporte pas l'anonymat (Section 7.1.4.3), ne permet pas l'ajout de nouveau proxy (Sections 7.1.4.1 et 7.1.4.2) et rend difficile la mise en chaîne des proxys (Sections 7.1.4.1 et 7.1.4.2).

7.2 Utilisation de la Signature Algorithm Agility pour protéger les adresses Anycast

Cette deuxième partie du chapitre s'intéresse aux adresses anycast configurées sur le lien-local et au partage d'adresses utilisé dans le protocole Proxy MIPv6. Nous verrons que ce partage est incompatible avec les protections offertes par le protocole SEND. Afin de résoudre cette incompatibilité, nous proposons une solution basée sur la *Signature Algorithm Agility* pour permettre la protection des adresses anycast par le protocole SEND.

7.2.1 Adressage anycast sur le lien-local

Cette section porte sur l'adressage anycast lien-local, où plusieurs nœuds sur un même lien partagent la même adresse IPv6, comme illustré Figure 7.5. Ce type d'adresse permet la répartition de charges et le basculement en cas de panne sur un de nœuds. Il s'agit d'une approche différente du routage anycast (illustré Figure 7.6) qui est utilisé par exemple dans le protocole de routage BGP (*Border Gateway Protocol*) où les différents nœuds ne sont pas présents sur le même lien. Du point de vue du correspondant, une adresse IPv6 anycast n'est pas distinguable d'une adresse IPv6 unicast classique. Néanmoins, d'un point de vue pratique, le comportement d'une adresse IPv6 anycast varie légèrement du comportement d'une adresse IPv6 unicast. Plus précisément, à la réception d'un message *Neighbor Solicitation* destiné à une adresse IPv6 anycast (par ex. durant la procédure de *résolution d'adresse*), les nœuds partageant l'adresse vont attendre durant un délai aléatoire avant d'envoyer le message *Neighbor Advertisement*. Ce délai aléatoire diminue le risque de congestion et offre un moyen simple de répartir la charge entre les nœuds qui partagent l'adresse. Par ailleurs, le message *Neighbor Advertisement* contient un drapeau *override* qui indique si l'information contenue dans le message doit écraser des entrées existantes dans le cache de voisinage du récepteur. Dans les messages *Neighbor Advertisement* émis par les adresses anycast, ce drapeau doit être désactivé : un nouveau message ne peut pas mettre à jour une entrée du cache de voisinage créée par un autre nœud partageant l'adresse. Ce mécanisme évite les situations de compétitions (en anglais, *race condition*) et s'assure qu'une fois la connexion établie, un correspondant ne se verra pas rediriger vers un autre nœud partageant l'adresse.

Il est aussi à noter qu'une adresse anycast ne peut être utilisée pour initier une nouvelle connexion, seule une adresse unicast peut être utilisée à cette fin.

7.2.2 Proxy Mobile IPv6 (RFC 5213)

Le protocole *Proxy Mobile IPv6*, défini dans le document [GLD+08], propose une solution de mobilité locale basée sur le réseau. La Figure 7.7 illustre les différents types de mobilité et situe la mobilité locale par rapport à d'autres types de mobilité, comme la mobilité globale offerte par MIPv6 [JPA04]. L'intérêt principal de ce protocole est qu'aucune modification n'est réalisée sur le nœud "mobile" (*Mobile Node*, MN). Ainsi, seules les fonctions de la pile IPv6 standard sont suffisantes pour le MN. Il en résulte

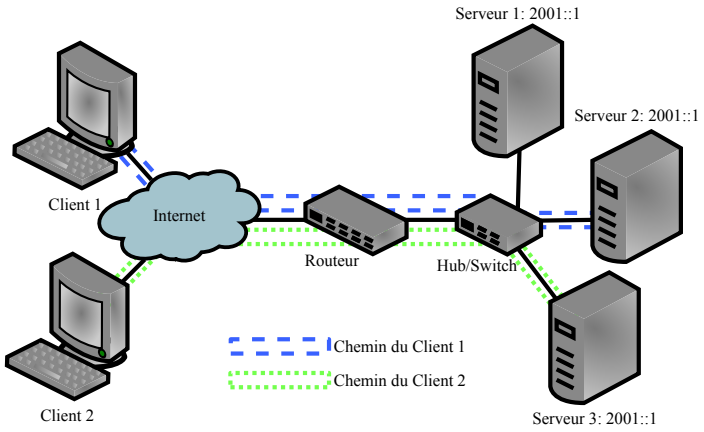


FIGURE 7.5: Exemple d'utilisation de l'adressage anycast sur le lien-local

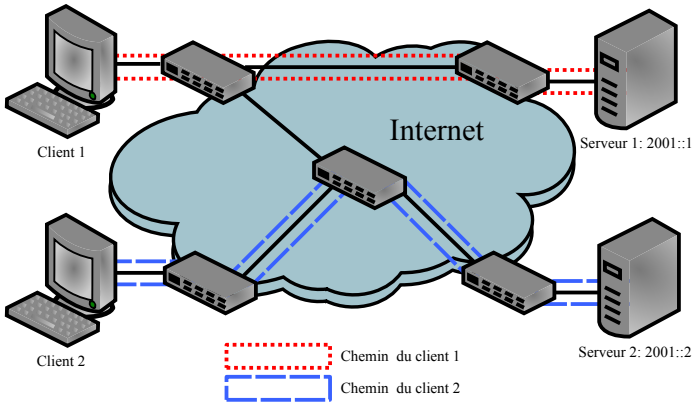


FIGURE 7.6: Exemple d'utilisation du routage anycast

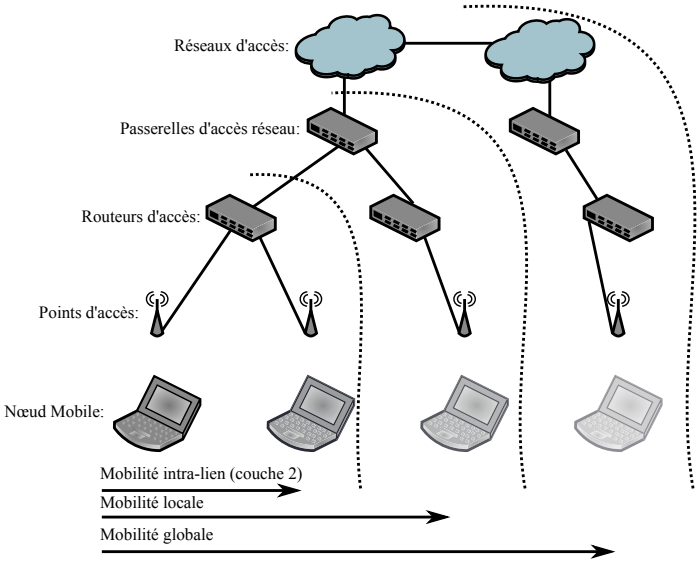


FIGURE 7.7: Illustration des différents types de mobilité et des équipements impliqués

que toute la complexité du contrôle de la mobilité doit être gérée du côté du réseau. À cet effet, deux nouvelles entités, la *Mobile Access Gateway* (MAG) et la *Local Mobility Anchor* (LMA), sont définies. La MAG assure le rôle de routeur d'accès, tandis que la LMA centralise les informations de mobilité du MN.

Lors de sa première connexion sur le réseau, le MN se voit attribuer un préfixe sous-réseau. Ce préfixe est entièrement réservé au MN. Quand un nœud se déplace sur le réseau, il se détache d'une MAG et se ré-attache à une nouvelle MAG. En joignant son nouveau lien, le MN envoie un message *Router Solicitation* à destination de tous les routeurs, comme indiqué dans la spécification [NNSS07]. La MAG qui se trouve sur ce lien interroge la LMA afin de connaître le préfixe sous-réseau qui a été attribué au nœud mobile. Elle avertira ensuite dans ses messages *Router Advertisement* ce même préfixe sous-réseau. Ainsi, du point de vue du nœud mobile, le nouveau lien est semblable à l'ancien lien. Par la suite, la LMA ré-achemine sur la nouvelle MAG les paquets à destination du nœud mobile.

Afin d'éviter de possibles collisions d'adresses lien-local entre le MN et les différentes MAG, quand le MN s'attache au réseau pour la première fois, la LMA choisit une adresse lien-local qui sera partagée entre les différentes MAGs. Par la suite, lorsque le MN se déplace, les MAGs interrogent la LMA afin de présenter la même adresse lien-local au MN.

7.2.3 Incompatibilités entre les adresses anycast et SEND

Les Sections 7.2.1 et 7.2.2 présentent deux mécanismes qui s'appuient sur le partage d'adresse IPv6. Toutefois, seul le possesseur de l'adresse, c.-à-d. le possesseur de la paire de clés publique/privée, peut ajouter la signalisation SEND aux messages du NDP. Une approche naïve consisterait à partager la paire de clés entre les nœuds qui désirent utiliser la même adresse. Toutefois, une telle approche affaiblirait le niveau de sécurité de ces solutions. Une autre solution doit donc être définie.

Ce problème, fortement similaire à celui énoncé dans la Section 7.1.3, indique une incompatibilité entre la protection offerte par le protocole SEND et le fonctionnement des adresses anycast.

7.2.4 Solutions existantes pour partager les adresses anycast en utilisant SEND

À notre connaissance, seules les solutions [KLBGM10] et [KWRG06], présentées dans les Sections 7.1.4.1 et 7.1.4.3, offrent la possibilité d'utiliser des adresses anycast dans un environnement SEND. Le document [KLBGM10] restreint le partage d'adresses aux seuls nœuds assurant le rôle de routeur. Ceux-ci sont autorisés via leur certificat à avertir n'importe quelle adresse d'un préfixe sous-réseau défini : il est donc possible d'avertir la même adresse. Cette limitation n'est pas présente dans le document [KWRG06] qui permet à plusieurs nœuds de partager une adresse en formant un anneau et en construisant une adresse CGA à partir de leurs clés publiques.

7.2.5 Utiliser les CGA multi-clés pour stocker la clé publique d'un voisin

De la même manière que la solution présentée Section 7.1.5, nous proposons de stocker les clés publiques des différents nœuds partageant l'adresse anycast dans des champs *extension clé publique* de la structure de données *CGA Parameters*. Ainsi, lorsqu'une adresse anycast est formée à partir de ces multiples clés, chaque nœud possédant la clé privée associée à une des clés publiques pourra revendiquer la possession de l'adresse. Par la suite, l'option de *signature universelle* permet de désigner la clé publique liée à la *signature numérique*.

La principale difficulté consiste à partager les clés publiques et la structure de données *CGA Parameters* entre les nœuds. Pour les routeurs agissant comme MAG dans le scénario PMIPv6 (voir Section 7.2.2), cette distribution peut être réalisée par le LMA. Toutefois, nous n'avons pas encore trouvé de solutions satisfaisantes pour le scénario où les adresses sont des adresses anycast sur le lien-local (voir Section 7.2.1). La distribution pourrait être réalisée par un administrateur (c.-à-d. un tiers de confiance) qui installerait manuellement les clés publiques et les clés privées nécessaires sur chaque nœud partageant l'adresse. Toutefois l'opération serait très complexe à réaliser. Une autre piste serait l'utilisation d'une phase d'apprentissage avant la création de l'adresse. Durant cette phase, tous les nœuds seraient considérés comme de confiance et diffuseraient leur clé publique. Ainsi, à l'issue de la phase d'apprentissage,

Nombre de clés	Taille du message	Gén. de la CGA	Gén. de la Sig.	Vérif. de la Sig.
1	312	1.142	0.075	0.035
2	408	1.173	0.077	0.036
4	592	1.204	0.080	0.037
11	1232	1.425	0.097	0.045

TABLEAU 7.2: Analyse de performance des principales opérations cryptographiques réalisées sur un message *Neighbor Solicitation* (les durées en secondes, les tailles en octets)

la CGA multi-clés pourrait être construite. Nous notons toutefois qu'une solution plus pérenne reste encore à définir.

Puisque la solution présentée dans cette section est très similaire à la solution présentée dans la Section 7.1.5, les limitations qui ont été décrites dans la Section 7.1.6 s'appliquent également ici.

7.3 Implémentation et évaluation des performances

Dans une phase préliminaire, nous avons étendu l'implémentation NDprotector [NDp] pour le support des CGA multi-clés. De par son usage de langage Python, NDprotector est facilement extensible, même s'il peut être plus lent que les autres implémentations existantes (comme celle développée par NTT DoCoMo). Ce dernier point pourrait être handicapant si nous effectuons des mesures de temps absolues (comme nous l'avons fait dans le Chapitre 4). Toutefois, nous mesurons des temps relatifs et notre implémentation basée sur NDprotector est suffisante pour évaluer la surcharge de calculs liée à l'augmentation du nombre de clés publiques dans la CGA multi-clés.

Pour prouver la faisabilité de notre solution, nous avons implémenté et testé notre proposition pour sécuriser l'usage des adresses anycast (présentée Section 7.2.5). Nous avons préféré implémenter cette solution plutôt que la solution pour protéger le *Proxy ND* (présentée Section 7.1.5) car celle-ci était plus facile à tester et déployer sur notre banc d'essai. Néanmoins, les deux propositions étant très similaires, l'évaluation de performances réalisée pour l'adressage anycast est aussi valable pour le *Proxy ND*.

La Table 7.2 et la Figure 7.8 illustrent les performances de notre proposition durant la génération de l'adresse CGA, et la génération et la vérification d'un message *Neighbor Solicitation*. Nous avons utilisé 10000 échantillons pour chacune des configurations de stockage de clé publique, comme nous l'avons fait pour nos travaux précédents sur l'évaluation des performances des CGA [CBL10] (voir également Section 4.2.1). Nous avons préféré les clés ECC aux clés RSA puisqu'elles surpassent ces dernières dans de nombreux domaines : taille de signature, taille de clé et vitesse de génération de signature. Dans cette évaluation précise, nous avons utilisé une clé ECC de 256 bits, ce qui est l'équivalent d'une clé RSA de 2048 bits en terme de sécurité.

Le Tableau 7.2 montre que le nombre de clés publiques contenues dans la CGA multi-clés a une influence sur la vitesse de génération et de vérification des signatures.

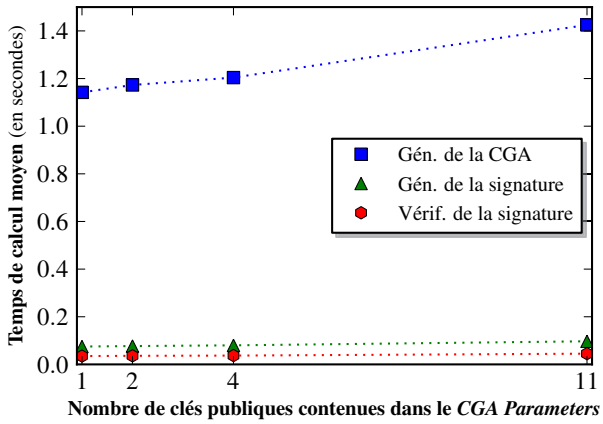


FIGURE 7.8: Représentation graphique de l'analyse de performances des principales opérations cryptographiques réalisées sur un message *Neighbor Solicitation*

Quand le nombre de clés publiques stockées dans la CGA multi-clés évolue, le temps de calcul de la CGA augmente (c.-à-d. calcul de *hash1* et *hash2*) et la taille du message augmente de manière linéaire. Ce comportement est prévisible : toutes ces opérations sont effectuées sur la structure de données *CGA Parameters* dont la taille s'accroît en fonction du nombre et de la taille des clés supplémentaires. Les sauts présents dans la Figure 7.8 lors de l'augmentation du nombre de clés sont dus à la fonction de hachage qui traite les données par blocs de 512 bits, comme nous l'avons déjà remarqué dans la Section 4.2.2.

Dans le pire des cas, quand l'adresse est partagée entre onze nœuds, le temps de génération de la CGA augmente de seulement 25% comparé au cas où l'adresse CGA ne contient qu'une seule clé publique. Ce pourcentage peut sembler excessif au premier abord, mais il doit être contrebalancé par le fait que le calcul de l'adresse CGA n'est effectué qu'une seule fois : durant l'initialisation du nœud. L'impact de notre solution est tout aussi minimal pour les opérations cryptographiques plus courantes, comme la génération et la vérification de la signature d'un message, où le nombre de clés influence (au maximum) les durées de 10 millisecondes (sur une implémentation non optimisée).

7.4 Synthèse du chapitre

Ce chapitre présente les incompatibilités entre le protocole SEND et les protocoles se basant sur le partage d'adresses et entre le protocole SEND et l'adressage anycast. Pour ces deux problèmes, nous présentons une liste de solutions qui permettent d'améliorer la compatibilité du protocole SEND avec ces protocoles.

Ce chapitre détaille également deux solutions basées sur la *Signature Algorithm Agility* qui permettent d'autoriser le partage d'adresses dans le protocole SEND. Ainsi, il devient possible d'utiliser la technique de *Proxy ND* et l'adressage anycast conjointement avec SEND. Cela permet d'améliorer la sécurité des protocoles Mobile IPv6, Proxy Mobile IPv6 et de la RFC 4389 (*Neighbor Discovery Proxies*). Ces solutions présentent un avantage majeur sur la solution basée sur l'utilisation de certificats et actuellement en cours de normalisation à l'IETF : l'autorisation donnée au proxy pour agir comme tel pour une adresse protégée est fournie par le nœud qui va être protégé (et non par une entité centrale). Ce choix permet de limiter le pouvoir que détient le proxy. À condition que la *Signature Algorithm Agility* soit déployée sur tous les nœuds d'un réseau, notre solution propose une alternative simple, viable et facile à déployer en comparaison des autres solutions déjà proposées. Nous avons étendu notre implémentation NDprotector afin de tester et prouver la faisabilité de notre solution. Grâce à cette implémentation, nous avons vérifié que notre proposition n'influençait que très peu les performances des opérations cryptographiques liées au protocole SEND.

Nous avons par ailleurs prouvé la faisabilité et la pertinence de nos solutions en développant une implémentation et en réalisant sur cette dernière une analyse des performances.

8 Proposition de mécanismes de génération d'adresses IPv6 pour améliorer l'anonymat des nœuds

Ce chapitre présente deux mécanismes de génération d'adresses visant à garantir l'anonymat dans les réseaux IPv6. Ceux-ci se basent sur les CGA afin de protéger des attaques locales sur les adresses IPv6. Ces CGA sont créées pour chaque nouvelle connexion et supprimées quand la connexion est terminée. La première proposition, nommée *CGA Éphémères Pseudo-Anonymes*, offre l'anonymat à la couche 3 (IP) tandis que la seconde, nommée *CGA Éphémères Anonymes*, permet l'anonymat à la couche 2 (par ex. Ethernet). Ces solutions ne s'intéressent pas à garantir l'anonymat pour les couches aux dessus de la couche 3.

Pour commencer ce chapitre, nous donnons les définitions du vocabulaire lié à la notion d'anonymat (Sec. 8.1). Ensuite, nous faisons un état de l'art des différents mécanismes de génération d'adresses IPv6 existants et étudions leur rapport avec la notion d'anonymat et de vie privée (Sec. 8.2). Dans les Sections 8.3.1 et 8.3.2, nous détaillons le positionnement des *CGA Éphémères* ainsi que leur modèle de sécurité. Nous verrons qu'il s'agit de la seule solution à assurer l'anonymat tout en protégeant les adresses (grâce à l'utilisation des CGA). Le concept général des *CGA Éphémères* est introduit dans la Section 8.3.3. Un premier mécanisme de génération d'adresses de la famille des *CGA Éphémères*, utilisant les *CGA Éphémères Pseudo-Anonymes*, est ensuite présenté (Sec. 8.3.4). De légers ajouts au protocole de Découverte de Voisins permettent d'améliorer les performances des *CGA Éphémères Pseudo-Anonymes* de manière significative (Sec. 8.3.5). Pour prouver la pertinence de notre solution, nous présentons une implémentation des *CGA Éphémères Pseudo-Anonymes* au sein de l'implémentation NDprotector (Section 8.3.6) et évaluons les performances de celle-ci (Section 8.3.7). Dans la Section 8.3.8, nous présentons un second mécanisme de génération d'adresses sur les *CGA Éphémères Anonymes*. Comme ce mécanisme repose sur la création de nouveaux identifiants MAC, nous construisons un mécanisme de détection d'adresses MAC dupliquées (Sec. 8.3.9). Nous terminons la présentation de ces deux mécanismes par une évaluation de la qualité de l'anonymat fourni (Sec. 8.3.10) et une analyse de leur sécurité (Sec. 8.3.11). Pour conclure, nous comparons nos solutions aux autres mécanismes de génération d'adresses (Sec. 8.3.12).

8.1 Définitions

Au cours de ce chapitre, nous utilisons plusieurs termes liés à l'anonymat. Comme il n'est pas rare de trouver dans la littérature plusieurs définitions pour chacun de ces termes, nous préférons préciser les définitions que nous utilisons tout au long de ce chapitre :

anonymat : (*anonymity* en anglais) "l'anonymat d'un sujet signifie que le sujet n'est pas identifiable à l'intérieur d'un ensemble de sujets" (traduit de la définition fournie par le document [PH10]) ;

pseudo-anonymat : désigne une notion d'anonymat allégé où le sujet est identifiable à l'intérieur d'un ensemble de sujets par un seulement certains autres sujets. C'est notamment le cas lorsqu'un tiers de confiance établit l'identité du sujet : celui-ci a connaissance de l'identité réelle du sujet ;

inassociabilité : (*unlinkability* en anglais) "l'inassociabilité de deux ou plus éléments d'intérêt (sujets, messages, actions, ...) du point de vue de l'attaquant signifie que dans le système (comprenant ces éléments et éventuellement d'autres éléments), l'attaquant ne peut pas distinguer suffisamment si ces éléments d'intérêt sont liés ou non" (traduit de la définition fournie par le document [PH10]) ;

associabilité : (*linkability* en anglais) est l'inverse de l'inassociabilité ;

8.2 Mécanismes de génération d'adresses existants

Cette section présente différents mécanismes de génération d'adresses IPv6. Ceux-ci sont classés en deux catégories : les mécanismes dits "à état" et les mécanismes "sans état" (comme l'*Autoconfiguration d'Adresse Sans État* que nous avons déjà présentée). Les mécanismes "à état", pour leur part, nécessitent la présence d'une entité persistante sur le réseau, un serveur.

8.2.1 DHCPv6 (RFC 3315)

Le *Dynamic Host Configuration Protocol for IPv6* (DHCPv6), défini dans le document [DBV+03], est un mécanisme de génération d'adresses à état. Il repose sur l'utilisation d'un serveur pour diffuser des paramètres de configuration aux nœuds IPv6. Ces paramètres de configuration sont plus poussés que ceux proposés par le mécanisme d'*Autoconfiguration d'Adresse Sans État*. Ainsi, il est possible de configurer, entre autres, des options de démarrage PXE, d'apprentissage du fuseau horaire ou encore de mise à jour dynamique de l'entrée DNS du nœud. Bien que le serveur DHCPv6 puisse lui même attribuer les adresses, il peut également laisser les nœuds utiliser l'*Autoconfiguration d'Adresse Sans État* [TNJ07]. Cependant, dans le premier cas, le serveur peut mémoriser les adresses attribuées afin de pouvoir les ré-assigner aux nœuds au fil de leur multiples reconnexions.

L'intérêt majeur de DHCPv6 est la centralisation de l'information dans le serveur qui offre au protocole un contrôle précis sur chaque nœud. De plus, et c'est là une impor-

tante différence avec le mécanisme d'*Autoconfiguration d'Adresse Sans État*, chaque nœud peut être paramétré avec des éléments de configuration différents.

8.2.2 L'Autoconfiguration d'Adresse Sans État (RFC 4862)

L'*Autoconfiguration d'Adresse Sans État* [TNJ07], présentée dans la Section 2.2, est le mécanisme de génération d'adresses par défaut en IPv6.

Récemment, de nouvelles options, comme l'option de *configuration DNS* dans les messages *Router Advertisement* [JPBM07], ont remis en cause le déploiement du protocole DHCPv6.

L'*Autoconfiguration d'Adresse Sans État*, en tant qu'extension du protocole de Découverte de Voisins, fournit un mécanisme qui permet à un nœud de créer une adresse topologiquement valide à partir d'un préfixe sous-réseau et de son adresse couche liaison. La partie préfixe sous-réseau est apprise dans les messages *Router Advertisement* tandis que l'identifiant d'interface est dérivée à partir de l'adresse couche liaison. Toutefois, ce dernier point pose des problèmes de traçabilité (et d'associabilité). En effet, puisque l'adresse couche liaison (c.-à-d. une adresse MAC) reste généralement stable à travers le temps, la partie identifiant d'interface de l'adresse demeure identique, et cela quelque soit le préfixe sous-réseau annoncé. Cela pose un problème pour un nœud qui se déplace sur l'Internet (par ex. utilisation nomade) car son identifiant d'interface permet de le suivre dans ses déplacements.

8.2.3 Extensions pour le respect de la vie privée dans l'Autoconfiguration d'Adresse Sans État (RFC 4941)

Pour résoudre le problème d'associabilité de l'*Autoconfiguration d'Adresse Sans État*, le document *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, décrit dans la RFC 4941 [NDK07], propose une approche intéressante. Celle-ci consiste à générer des adresses IPv6 dites *temporaires* où la partie identifiant d'interface est aléatoire et où les adresses sont marquées comme *privées*. La génération aléatoire d'adresse peut être basée sur le processus de génération d'adresses des CGA [Aur05]. Ces adresses sont choisies par l'application pour les connexions sortantes quand la protection de la vie privée (*privacy*) est nécessaire, alors que les adresses non-*temporaires* sont utilisées pour les connexions entrantes et sont connues de tous (par ex. enregistrées dans une entrée DNS). Une adresse *temporaire*, comme son nom l'indique, a une durée de vie limitée, et passe à l'état "dépréciée" (illustré Figure 2.1) après une certaine durée. Une nouvelle adresse *temporaire* est alors créée pour remplacer l'ancienne. Le délai entre le passage de l'adresse de l'état "dépréciée" à l'état "invalide" permet aux connexions établies à partir de cette adresse de finir correctement, afin de ne pas interrompre une session TCP [Pos81] ou UDP [Pos80].

Il est à noter que de multiples connexions peuvent être établies avec une même adresse *temporaire*, ce qui permet d'établir une association entre les connexions et le nœud et pose donc toujours le problème d'associabilité à un degré moindre.

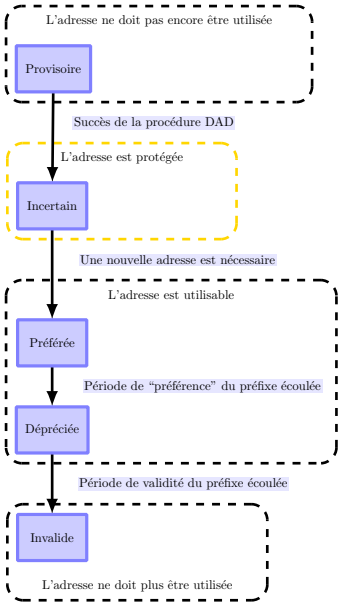


FIGURE 8.1: Cycle de vie d'une adresse IPv6 (avec l'ajout du nouvel état "incertain")

8.2.4 Adresse IPv6 Éphémères

Les *adresses IPv6 éphémères*[KAM09b] offrent une approche alternative (bien que non finalisée) aux adresses *temporaires* [NDK07] et proposent de générer une nouvelle adresse par connexion sortante. Tout comme dans la Section 8.2.3, cette adresse est générée de manière aléatoire. Néanmoins, la durée de vie de cette adresse étant liée à la durée de vie d'une connexion sortante, il devient plus difficile de lier les différentes connexions ensembles. Par ailleurs, une chaîne de hachage peut être utilisée à la place d'une génération aléatoire de l'adresse quand un pseudo-anonymat est requis. Ce qui différencie cette proposition du document RFC 4941 [NDK07] est l'encodage du port source TCP/UDP de la connexion à l'intérieur de la partie identifiant d'interface de l'adresse. Pour la partie préfixe sous-réseau, le document recommande la création d'un nouvel espace d'adresse réservé à l'utilisation des *adresses IPv6 éphémères*. Ce dernier point rend les *adresses IPv6 éphémères* différentiables des autres adresses.

D'un point de vue performance, la création de nouvelles connexions sortantes crée une nouvelle adresse, ce qui implique le déclenchement de la procédure de Détection d'Adresse Dupliquée (DAD) qui va empêcher l'utilisation de l'adresse, et donc ralentir l'établissement de la connexion, jusqu'à ce que l'unicité de l'adresse ait pu être vérifiée.

Afin d'annuler le ralentissement causé par la procédure de DAD, le document compagnon [KAM09a] propose d'ajouter aux adresses un nouvel état nommé "incertain", illustré Figure 8.1, situé entre les états "provisoire" (quand la procédure de DAD n'est pas achevée) et "préférée" (quand la procédure de DAD est terminée et que l'adresse est

utilisable). Pendant cet état, l'adresse n'est pas encore utilisable, mais la procédure de DAD est effectuée et le nœud défend l'adresse (pour éviter la duplication d'adresses). Pour réserver l'adresse, le nœud écoute l'adresse multicast du "nœud sollicité" correspondante et répond aux messages *Neighbor Solicitation* dont l'adresse source est l'adresse non-spécifiée. Ces messages correspondent aux messages émis par les nœuds effectuant une procédure de DAD sur l'adresse protégée.

8.2.5 FLASCHE

L'article [Zug05] présente FLASCHE, une solution développée en dehors de l'IETF. Celle-ci propose la génération d'un identifiant aléatoire pour chaque nouvelle connexion (par ex. connexions TCP). Un soin important est porté à la génération d'adresses afin que deux adresses générées par un même nœud ne soient pas associables. Cela inclut la création d'un identifiant aléatoire à la couche 2 et à la couche 3. L'article présente en détail l'implémentation de la solution mais ne fournit aucune étude des performances, et ne permet donc pas de déterminer la pertinence de la solution. Toutefois, l'auteur décide de supprimer la Détection d'Adresse Dupliquée pour des raisons de performances. Ce choix est critiquable : la solution proposée nécessite la création de nombreuses adresses et augmente d'autant le risque de collision (non volontaire) d'adresses.

8.2.6 Les mécanismes de génération d'adresses et la protection de l'adresse

Seules les Sections 8.2.4 et 8.2.5 proposent la mise en place d'identifiants permettant l'anonymat. Toutefois aucune de ces solutions ne permet la protection de l'adresse, que ce soit via l'utilisation des CGA ou d'un mécanisme indépendant. Par le passé, il a été prouvé que les mécanismes de génération d'adresses pouvaient intégrer le support des CGA, puisque celui-ci a été intégré aux solutions présentées Sections 8.2.2 et 8.2.3.

Toutefois, aucun de ces mécanismes ne propose à la fois une garantie de l'anonymat et la protection de l'adresse contre les attaques locales.

8.3 CGA Éphèmères

Dans cette section, nous proposons deux solutions qui utilisent un identifiant unique, basé sur les CGA, pour chaque nouvelle connexion.

8.3.1 Positionnement de la solution

Le but de notre approche est de garantir l'anonymat du nœud dans les réseaux TCP/IP. À cet effet, nous constatons que pour ralentir le travail d'un attaquant souhaitant tracer le nœud, il est souhaitable qu'aucune des différentes couches ne laisse échapper d'informations permettant d'identifier celui-ci.

Nous avons alors décidé de focaliser notre travail sur les couches 2 (liaison) et 3 (réseau). Le travail pour fournir l'anonymat aux autres couches est également important puisque ce sont ces couches qui détermineront le succès de notre solution. Des travaux similaires ont été entrepris pour les autres couches :

couche application si aucune précaution n'est appliquée, la couche application laisse passer des informations importantes qui annulent les mécanismes de protection de l'anonymat implémentés dans les couches basses, comme cela a été montré dans le document [Eck10] pour les applications Web. Cette menace peut être contrée par des mécanismes de sécurité assurant la protection de la couche application, comme le protocole HTTPS [Res00] qui sécurise le protocole HTTP [FGM+99] ;

couche transport le protocole TCP [Pos81] permet un traçage du nœud grâce aux numéros de séquence. Les documents [Bel96] et [ANN07] proposent de rendre ce champ aléatoire afin d'éviter le traçage. L'utilisation de la protection IPSEC/ESP [Ken05] peut également empêcher la fuite d'informations de cette couche et des couches supérieures ;

couche physique l'identification par empreinte des ondes radio de l'émetteur [BHK05] permet de reconnaître, et donc de tracer, le nœud émetteur. L'emploi de cette technique reste limitée à du matériel spécialisé et onéreux.

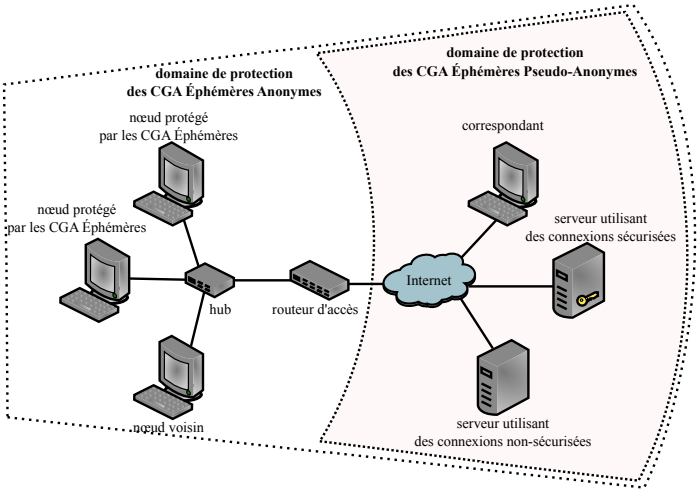


FIGURE 8.2: Domaine de protection des solutions des CGA Éphémères

La Figure 8.2 illustre le positionnement de nos deux solutions, *CGA Éphémères Pseudo-Anonymes* et *CGA Éphémères Anonymes*, dans le réseau. On voit que les *CGA Éphémères Pseudo-Anonymes* offrent l'anonymat après le premier saut et sont donc à privilégier dans les réseaux privées où le lien peut être considéré comme de confiance. Pour les réseaux publiques, les *CGA Éphémères Anonymes* offrent une protection dès le premier saut, assurant l'anonymat du nœud vis à vis de ses voisins.

Nous avons construit ces solutions à partir des contraintes suivantes :
- la protection doit fonctionner sur un réseau en IPv6. En effet, nous pensons que la migration vers l'IPv6 va s'accélérer lorsque la pénurie d'adresses IPv4 sera atteinte : ainsi le réseau IPv6 prédominera à terme ;
- aucune infrastructure de sécurité ou de confiance n'est disponible pour assurer la protection du nœud. Cela implique qu'il n'y a aucun proxy anonymisant disponible sur le réseau ;
- au niveau du lien, le nœud doit être capable de protéger ses adresses (c.-à-d. contre les tentatives d'usurpations) ;
- finalement, afin de faciliter l'adoption de notre proposition, nous imposons que les nœuds correspondants n'ont pas à être modifiés afin de communiquer avec les nœuds supportant notre proposition.

8.3.2 Modèle de sécurité

Nous souhaitons placer le nœud dans un environnement hostile où aucun mécanisme de sécurité externe ne peut l'aider à obtenir l'anonymat. Pour se faire, nous considérons les postulats suivants sur le nœud et son voisinage :
- aucune infrastructure de sécurité ne permet au nœud d'assurer l'anonymat de ses communications ;
- les nœuds, au delà d'un saut, ne sont pas considérés comme de confiance et peuvent s'allier pour tracer le nœud ;
- dans le cas des *CGA Éphémères Anonymes*, les nœuds voisins (c.-à-d. situés à un saut) ne sont pas considérés comme de confiance. Ils peuvent également s'allier pour tracer le nœud ;
- un attaquant peut (s'il est situé sur le même lien) essayer d'usurper une ou plusieurs adresses appartenant à un même nœud ;
- les couches supérieures (TCP, HTTP, ...) n'offrent pas d'informations compromettant l'anonymat du nœud. Ce dernier point nécessite l'utilisation de protections spécifiques dans les protocoles de couches supérieurs.

Puisque nous souhaitons protéger l'adresse des attaques par usurpation, l'utilisation des CGA et du protocole SEND est naturelle. La création d'un identifiant unique pour chaque connexion à la couche 3 (pour les *CGA Éphémères Pseudo-Anonymes* et les *CGA Éphémères Anonymes*) et à la couche 2 (pour les *CGA Éphémères Anonymes*) permet de s'assurer l'inassociabilité des connexions.

8.3.3 Vue d'ensemble des CGA Éphémères

Le principe des adresses *Ephemeral Cryptographically Generated Addresses* (CGA Éphémères) repose sur une génération d'adresses très dynamique semblable à celles proposées Sections 8.2.4 et 8.2.5. Plus précisément, chaque connexion TCP/UDP sortante [1] est liée à une adresse CGA à usage unique. À la fin de la connexion, l'adresse est supprimée du nœud.

1. nous considérons qu'une session UDP est identifiée par un quadruplet - adresse source, adresse destination, port source, port destination - qui reste fixe au cours du temps.

Dans les sections suivantes, nous détaillons deux variantes de ce travail : les *CGA Éphémères Pseudo-Anonymes* (fournissant l'anonymat à la couche 3) et les *CGA Éphémères Anonymes* (fournissant l'anonymat aux couches 2 et 3).

Il est à noter que, dans ce chapitre, la solution *CGA Éphémères* n'inclut pas les adresses CGA qui pourraient être configurées sur un nœud afin de recevoir des connexions entrantes. De telles adresses statiques peuvent cohabiter sur un nœud qui utilise des *CGA Éphèmères* et sont plus appropriées pour les serveurs d'applications publiques qui possèdent une entrée DNS.

8.3.4 Les CGA Éphémères Pseudo-Anonymes

Dans cette première approche, nommée *CGA Éphémères Pseudo-Anonymes*, les différentes CGA utilisées par le nœud sont générées à partir de la même clé publique. Puisque cette clé publique est connue des voisins à un saut du nœud après les échanges de messages SEND, les voisins peuvent relier les différentes *CGA Éphémères Pseudo-Anonymes* au nœud d'origine. Pour les nœuds situés à plus d'un saut, il n'est pas possible d'effectuer cette association et les adresses CGA sont seulement vues comme des adresses générées de manière aléatoire, en tous points indistinguables des adresses décrites par le document RFC 4941 [NDK07].

Il est à noter que la clé publique de ce nœud peut être fournie par une entité de confiance, qui autorise le nœud à joindre le réseau. Dans ce cas, puisque le processus de génération des CGA fait appel à un deuxième élément aléatoire (le *modifier*), l'entité de confiance, si elle n'est pas présente sur le lien (c.-à-d. voisinage à un saut), n'est pas capable de lier les différentes adresses CGA du nœud. Dans le cas où l'entité de confiance est présente sur le même lien que le nœud, il lui est possible de lier les différentes adresses du nœud et de pouvoir reconstruire l'activité du nœud.

Pour conclure, l'approche *CGA Éphémères Pseudo-Anonymes* n'offre pas l'anonymat au premier saut, mais offre l'anonymat (l'inassociabilité) après le premier saut. Cela implique une confiance en ses voisins directs. Toutefois, si un attaquant situé à distance peut s'assurer que le nœud est seul utilisateur du préfixe sous-réseau (par ex. l'utilisateur possède un tunnel qui fournit l'IPv6 à son nœud), les *CGA Éphémères Pseudo-Anonymes* ne peuvent pas assurer l'anonymat du nœud.

8.3.5 Amélioration des délais via la construction d'un pool d'adresses CGA

Nous pensons ici, comme dans le Chapitre 4, que les mauvaises performances d'une solution de sécurité peuvent détourner les utilisateurs de celle-ci. C'est la raison pour laquelle nous attachons beaucoup d'importance à réduire l'impact que peut avoir notre solution sur l'utilisation d'un nœud. Comme nous l'avons vu dans [CBL10], la durée de la génération d'une adresse CGA est majoritairement influencée par les durées de la génération de la paire de clés et du calcul du *modifier* final. Toutefois, ces deux durées sont relativement faibles comparées au délai induit par la procédure de Détection d'Adresse Dupliquée (DAD) que le nœud doit effectuer avant de pouvoir utiliser l'adresse. En effet, sur une machine de moyenne gamme, la génération d'une

paire de clés RSA de 1024 bits prend environ 160 millisecondes, temps qui peut-être réduit à 10 millisecondes en utilisant les courbes elliptiques, alors que la procédure de DAD impose par défaut un délai d'une seconde avant l'utilisation éventuelle de l'adresse (cas où aucune collision n'est détectée, cf Section 2.4).

Dans le but d'amoindrir les délais induits par la procédure DAD, nous proposons d'utiliser la proposition du document [KAM09a], et de créer un état intermédiaire "incertain", illustré dans la Figure 8.1. Cet état permet à l'adresse IPv6 d'être réservée pour une utilisation future. Durant le temps où l'adresse est dans l'état "incertain", le nœud va défendre l'adresse. Contrairement à la proposition de "DAD Optimiste" [Moo06], cette approche ne réduit pas le délai avant que l'adresse ne soit disponible pour émettre un paquet. Toutefois, quand l'adresse est dans l'état "incertain", elle peut-être utilisée instantanément puisque la procédure de DAD a déjà été effectuée et que l'unicité de l'adresse a été vérifiée.

Nous étendons la proposition [KAM09a] en ajoutant un *pool* d'adresses à chaque nœud. La création de ce *pool* force le nœud à effectuer une procédure de DAD sur chaque adresse du *pool*. Tant que la régénération du *pool* est plus rapide que la vitesse d'épuisement des adresses, de nouvelles connexions peuvent être établies sans délai supplémentaire.

La taille du *pool* d'adresses est variable et doit être ajustée en fonction du type d'applications lancées par le nœud. Par exemple, les applications P2P vont générer l'ouverture rapide de nombreuses connexions simultanées. Par ailleurs, un sur-dimensionnement de la taille du *pool* n'a pas de conséquences autres que l'augmentation de la durée de remplissage du *pool* et la génération de trafic NDP supplémentaire.

Avec cette amélioration, la procédure de DAD ralentit l'utilisation d'une adresse seulement durant le démarrage du nœud quand le *pool* n'est pas complètement constitué. Cependant, dans certains scénarios où le nœud se déplace et change de préfixes sous-réseau, le *pool* ne contient pas encore d'adresses dérivées de ce préfixe et l'attente (minimale) d'une seconde pour utiliser la première adresse est trop longue. Dans ce cas, la procédure de *DAD Optimiste* [Moo06] peut offrir une amélioration en permettant au nœud d'utiliser une adresse avant la fin de la procédure de DAD et alors que son unicité n'a pas encore été prouvée.

Il est à noter que lorsqu'une nouvelle connexion s'ouvre, le manque de lien entre l'adresse couche liaison et l'adresse *CGA Éphémères Pseudo-Anonyme* déclenche une procédure de *Résolution d'Adresse* sur le nœud récepteur. Cet échange de messages *Neighbor Solicitation - Neighbor Advertisement* ralentit l'établissement de la connexion. Pour éviter ce ralentissement, nous proposons que le nœud envoie des messages *Neighbor Advertisement* "gratuits" à ses correspondants avant d'établir une connexion.

8.3.6 Implémentation des CGA Éphémères Pseudo-Anonymes dans NDprotector

Nous avons implémenté les *CGA Éphémères Pseudo-Anonymes* sous forme de greffon dans l'implémentation *NDprotector* [NDp]. L'implémentation *NDprotector* a été précédemment présentée dans le Chapitre 6 et son système de greffons dans la Section 6.2.2.

Afin de faciliter notre travail d'analyse des paquets et de découverte de nouvelles connexions, nous avons implémenté les *CGA Éphémères Pseudo-Anonymes* seulement pour les connexions TCP [Pos81]. Ces dernières, contrairement aux connexions UDP, sont facilement reconnaissables car le premier paquet émis est un paquet TCP dont le drapeau SYN est actif. Une règle de filtrage permet alors d'intercepter ces paquets et de les traiter dans *NDprotector*.

L'implémentation commence à peupler son *pool* d'adresses dès son démarrage. Quand celui-ci est à la moitié des adresses qui le composent initialement, il entame une phase de régénération, où il calcule autant d'adresses que nécessaire pour re-atteindre la limite de son *pool*.

Pour chacune de ces adresses, la procédure de DAD [TNJ07] est effectuée. Nous n'incluons pas actuellement le support du *DAD Optimiste* [Moo06], et ne pouvons donc pas bénéficier de son optimisation. De plus, l'état "incertain" n'étant pas implémenté dans le noyau Linux, nous devons émuler son fonctionnement. Pour ce faire, nous plaçons les adresses nouvellement générées dans l'état "dépréciée", où elles ne seront pas choisies pour établir les nouvelles connexions. Cet état simule l'état "incertain" puisque le DAD est effectué auparavant sur chaque adresse et que le nœud continue à réserver l'adresse.

Par la suite, pour forcer le mécanisme de sélection de l'adresse source de Linux à choisir une adresse, nous plaçons successivement l'adresse dans l'état "préférée" puis de nouveau dans l'état "dépréciée". En effet, quand une seule des multiples adresses d'une interface est dans l'état "préférée", le mécanisme de sélection d'adresse la choisit pour émettre le paquet. Pour influencer le mécanisme de sélection d'adresses et contrôler la prochaine adresse utilisée, nous nous assurons que lors du fonctionnement de notre implémentation, à tout moment, une seule adresse du nœud est dans l'état "préférée" (la prochaine *CGA Éphémère Pseudo-Anonyme* à être utilisée) et que toutes les autres adresses sont dans l'état "dépréciée" (les adresses restantes du *pool* et les adresses en cours d'utilisation). Cette gestion de l'état de l'adresse est illustrée dans la Figure 8.3.

Quand une connexion TCP est initialisée, un paquet contenant un drapeau SYN est intercepté par notre implémentation (car une règle est mise en place par la *libnetfilter-queue*). L'adresse source (qui est donc en état "préférée" puisqu'elle a été choisie pour émettre le paquet) est changée en l'état "dépréciée". Une nouvelle adresse est ensuite préparée pour la prochaine connexion : cette adresse est choisie dans le *pool* d'adresses et est marquée comme "préférée". Quand la connexion est terminée (c.-à-d. un paquet avec un drapeau FIN est émis par un de deux cotés de la connexion), la suppression de l'adresse est programmée.

8.3.7 Analyse de performances des CGA Éphémères Pseudo-Anonymes

Pour se rapprocher le plus possible de l'attente des utilisateurs, nous avons mesuré l'incidence de notre solution sur le temps d'ouverture d'une connexion TCP. Nous avons souhaité savoir si l'augmentation de ce délai restait raisonnable. Afin mesurer cette durée, nous avons utilisé l'outil *Apache HTTP server benchmarking tool* (qui fait parti

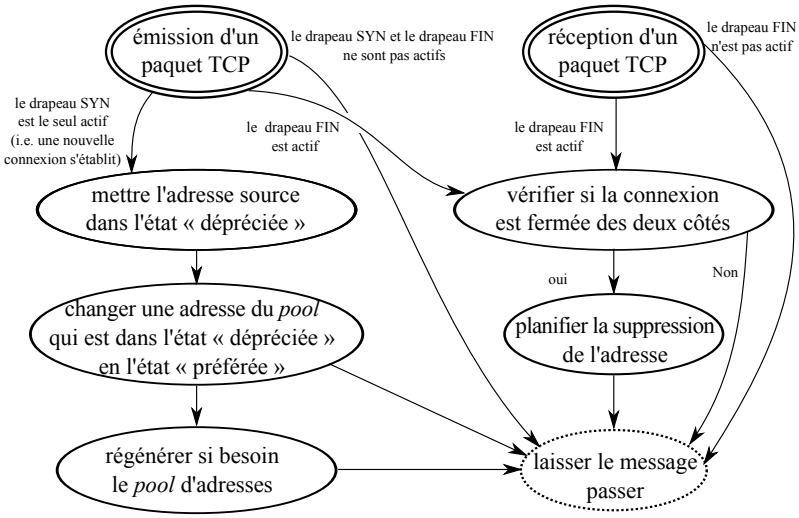


FIGURE 8.3: Contrôle par NDprotector du mécanisme de sélection de l'adresse sous Linux

de serveur Web Apache [2]) pour ouvrir séquentiellement de nombreuses connexions.

Nous avons mesuré le temps d'ouverture de connexions à destination de trois serveurs différents. Deux sont situés à distance et utilisent des routes différentes, `http://www.kame.net` et `http://ipv6.google.com`. Ils sont accessibles via un tunnel IPv6 configuré sur le nœud. Le troisième serveur Web est situé sur un nœud voisin. Cela nous permet d'évaluer l'impact de la procédure de *résolution d'adresse* sur notre proposition.

La Figure 8.4 présente les résultats de notre évaluation de performance réalisée sur un processeur Intel Core 2 Duo cadencé à 2Ghz. Les barres indiquent le "temps de connexion", qui se réfère ici au temps écoulé entre la demande d'initialisation de la connexion vers le serveur (qui a déclenché le message TCP contenant le drapeau SYN) et la première réponse du serveur (c.-à-d. le premier paquet avec un drapeau ACK actif reçu). Pour chaque serveur, nous avons mesuré ce délai pour 150 connexions. Les valeurs recueillies à l'issue de ces premières connexions restant stables (avec un écart type inférieur à la dizaine de millisecondes), nous n'avons pas jugé nécessaire d'évaluer plus d'échantillons.

Les quatre premières barres de la Figure 8.4, qui représentent les deux sites distants, indiquent que notre solution présente un surcoût de 10 à 15 millisecondes, ce que nous considérons comme raisonnable pour une implémentation en espace utilisateur. Les cinquième et sixième barres permettent de visualiser le délai ajouté par notre solution quand la procédure de *résolution d'adresse* doit être réalisée. Bien que ce délai soit assez long (un peu plus de 100 millisecondes), il s'explique par le ralentissement que

2. `http://apache.org`

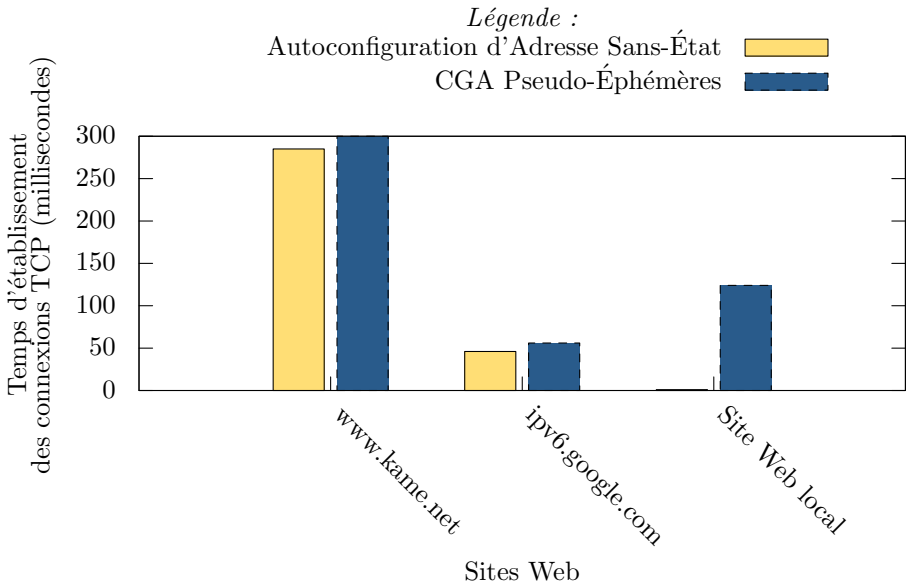


FIGURE 8.4: Évaluation du délai moyen d'établissement d'une connexion TCP avec et sans les *CGA Éphémères Pseudo-Anonymes*

fait subir SEND à l'échange de messages *Neighbor Solicitation - Neighbor Advertisement*. Comme nous l'avons précisé dans la Section 8.3.5, ce délai peut-être supprimé en précédant l'établissement d'une connexion par l'envoi d'un message *Neighbor Advertisement* "gratuit".

Sur nos divers tests, nous avons utilisé des adresses CGA basées sur une clé RSA de 1024 bits et une valeur de SEC égale à 1. Ces paramètres influencent principalement la durée de génération de l'adresse et donc de la régénération du *pool*. Nous avons mesuré que nous avions une vitesse de génération assez lente (0.33 adresses par seconde). Toutefois, ce mauvais résultat est due à l'architecture de *NDprotector*. En réalité, l'évaluation des performances réalisée dans [CBL10] est plus optimiste et montre que du matériel plus lent (un Pentium 4 cadencé à 2.5Ghz) peut réaliser un taux de régénération de 7 adresses par seconde. Cela signifie qu'il serait possible de conserver un rythme de 7 ouvertures de connexions par seconde sans réussir à vider le *pool* d'adresses.

Nous avons également vérifié qu'une fois la connexion TCP établie, notre solution n'avait pas d'influence sur le débit.

8.3.8 Les CGA Éphémères Anonymes

L'anonymat apporté par l'approche *CGA Éphémères Pseudo-Anonymes* n'est pas fonctionnel au premier saut puisque la même adresse MAC et la même clé publique sont utilisées pour chaque message. Si utiliser une clé publique différente pour chaque

adresse est assez aisé (il suffit d'en générer une nouvelle), fournir l'anonymat au niveau de la couche liaison (Ethernet dans le cas qui nous intéresse) est une chose difficile à réaliser. Certaines approches sur les réseaux 802.11b, comme [GG05], proposent un changement périodique de l'adresse MAC pour diminuer l'associabilité des connexions. Cependant il ne s'agit pas là d'anonymat, puisqu'à un instant donné plusieurs adresses IPv6 peuvent partager la même adresse MAC (d'une durée de vie possiblement courte), ce qui permet de lier les différentes adresses et activités d'un même nœud.

L'idée de notre nouvelle approche, nommée *CGA Éphémères Anonymes*, est simple. Elle consiste à lier la durée de vie d'une adresse MAC à celle d'une connexion. C'est à dire que pour chaque nouvelle connexion, une nouvelle adresse MAC est générée. Cette adresse MAC est dérivée de la partie identifiant d'interface de la *CGA Éphémère Anonyme* qui est utilisée pour cette connexion, comme présenté dans la Figure 8.5. Cette adresse MAC suit les standards et est allouée dans l'espace d'adresse réservé aux adresses "administrées localement".

Ce dernier point permet de différencier les *CGA Éphémères Anonymes* des adresses IPv6 classiques, mais peut être facilement modifié afin que les *CGA Éphémères Anonymes* se camouflent dans l'espace d'adresse réservé aux constructeurs. Ce dernier ne contient en principe que des adresses globalement uniques à travers le monde. Cependant, en pratique, cette unicité est difficilement vérifiable.

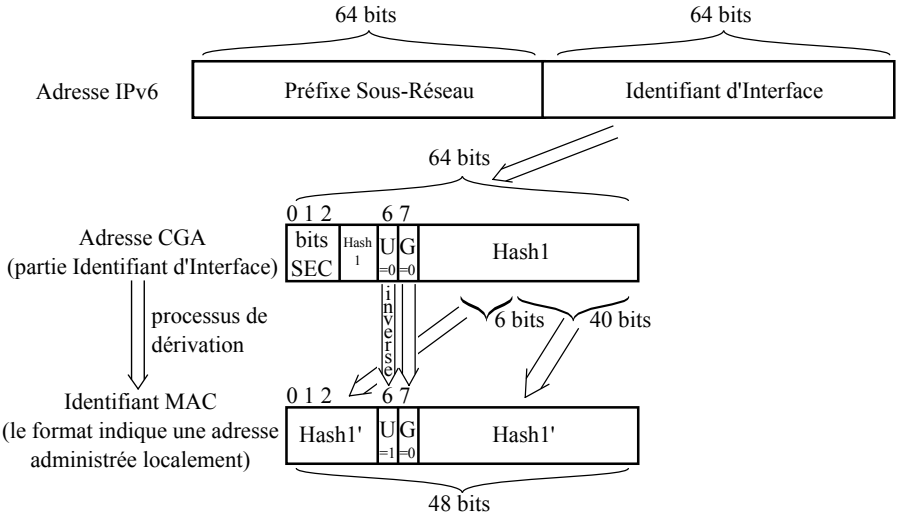


FIGURE 8.5: Dérivation d'un identifiant MAC de 48 bits à partir d'une adresse IPv6 CGA

Un nouvelle adresse CGA est également calculée pour chaque nouvelle connexion. Contrairement aux *CGA Éphémères Pseudo-Anonymes*, elle est ici dérivée d'une nouvelle clé publique. La procédure DAD permet de s'assurer de l'unicité de l'adresse IPv6 sur le lien, mais ne permet pas de s'assurer de l'unicité de l'adresse MAC. Même

si le risque de collision d'adresse MAC est faible (tout comme le risque de collision d'adresse IPv6), une collision non détectée pourrait perturber les communications des nœuds concernés. À cet effet, nous avons mis au point une technique nous permettant de détecter les adresses MAC dupliquées sur le lien, celle-ci est présentée Section 8.3.9.

Tout comme pour la solution *CGA Éphémères Pseudo-Anonymes*, la durée de vie de l'adresse IPv6 est limitée à la durée de la connexion sortante. Cela implique que la nouvelle adresse MAC et la nouvelle clé publique ne doivent plus être utilisées quand la connexion est fermée.

D'un point de vue technique, changer l'adresse MAC d'une carte réseau peut être réalisé de manière logicielle, avec par exemple la commande *ifconfig* sur les systèmes UNIX. Ici, nous changeons l'adresse MAC avant l'envoi de chaque paquet sortant pour refléter l'adresse source IPv6 qui lui correspond. Afin de recevoir les paquets entrants destinés à ces multiples adresses MAC "virtuelles", le nœud active le mode promiscuité (*promiscuous mode*) de sa carte réseau. Par ailleurs, les nœuds correspondants dans le voisinage engagent des procédures de *résolution d'adresse* pour chaque nouvelle adresse avec laquelle ils communiquent, les nœuds implémentant les *CGA Éphémères Anonymes* doivent alors être modifiés pour s'adapter au changement rapide d'adresses MAC. En effet, chaque nouvelle adresse MAC implique l'envoi préalable d'un nouveau message NDP indiquant le lien entre l'adresse MAC et l'adresse *CGA Éphémères Anonyme* associée avant l'ouverture de la connexion. Enfin, nous notons que les nœuds correspondants n'ont pas besoin d'être modifiés pour communiquer avec les nœuds implémentant les *CGA Éphémères Anonymes*.

Faute de temps, nous n'avons pas implémenté cette solution. Nous pouvons appliquer toutes les techniques d'optimisation des *CGA Éphémères Pseudo-Anonymes* (présentées Section 8.3.5), comme l'envoi préventif de messages NDP lors de l'ouverture de connexions pour diminuer le délai d'envoi. Toutefois, certains aspects nous laissent supposer que les *CGA Éphémères Anonymes* sont plus lente d'utilisation que les *CGA Éphémères Pseudo-Anonymes*. En effet, la difficulté est d'implémenter un changement rapide d'adresses MAC sur le nœud, ce qui implique une modification du système d'exploitation et un ralentissement dans le traitement de tous les paquets reçus et émis. Un piste à explorer est l'utilisation de cartes réseaux qui sont capables d'utiliser plusieurs adresses MAC simultanément (c'est le cas de certaines cartes Broadcom [Bro08]).

8.3.9 Détection de l'unicité de l'adresse MAC sur le lien

Dans cette section, nous nous intéressons à la détection de la duplication d'adresse MAC, sans pour autant qu'il soit possible d'identifier le nœud qui effectue la procédure de détection.

Nous souhaitons que notre méthode de vérification de l'unicité de l'adresse MAC ne nécessite aucune modification sur les nœuds. Pour que notre technique fonctionne, nous partons du postulat suivant : la majorité des nœuds (qu'ils utilisent Windows, Linux ou un BSD quelconque) activent par défaut des adresses IPv6 sur leurs interfaces. Ainsi, même quand aucun routeur IPv6 n'annonce de préfixe sous-réseau, les nœuds configurent des adresses *lien-local* sur chaque interface.

Notre procédure de détection d'adresse dupliquée fonctionne d'une manière similaire au DAD du NDP, nous envoyons une requête et la réception d'une réponse nous permet de détecter une adresse dupliquée. Ainsi, le nœud envoie un message *ICMPv6 Echo Request*, défini dans le document [CDG06], dont l'adresse source n'est pas spécifiée (`::`) et dont l'adresse destination est l'adresse multicast "tous les nœuds" (`ff02::1`). L'adresse MAC source est nulle (tous les bits à 0) et l'adresse destination est l'adresse MAC dont on souhaite vérifier l'authenticité. Si l'adresse MAC est utilisée par un nœud sur le lien, celui-ci reçoit la requête et répond avec un message *ICMPv6 Echo Reply*. Si après un certain délai, aucune réponse n'est reçue, l'initiateur de la requête considère que l'adresse MAC est unique sur le lien.

8.3.10 Analyse de la qualité de l'anonymat fourni par les CGA Éphémères

Dans notre proposition *CGA Éphémères Anonymes*, chaque adresse CGA est générée avec sa propre clé publique. L'algorithme de génération de la clé assure, de par sa conception, que la clé publique est aléatoire. Dans le cas des *CGA Éphémères Pseudo-Anonymes*, c'est le calcul d'un nouveau *modifier*, qui est par définition aléatoire, qui permet d'assurer que les différentes adresses générées à partir de la même clé sont aléatoires. De ce fait, chaque adresse IPv6 est indépendante des autres et un nœud externe peut seulement constater que différentes adresses sont issues d'un même préfixe sous-réseau. Dans le cas des *CGA Éphémères Anonymes*, puisque les adresses MAC sont dérivées des adresses CGA, un nœud externe ne peut pas dire si un paquet reçu est émis à partir d'un nœud en particulier ou de son voisin. Toutefois, si un attaquant arrive à savoir que le nœud est seul sur le lien (par ex. le nœud est connecté directement à un modem), ni les *CGA Éphémères Pseudo-Anonymes* ni les *CGA Éphémères Anonymes* ne permettent de protéger le nœud. De plus, si l'attaquant est sur le lien, alors les *CGA Éphémères Pseudo-Anonymes* n'assurent pas l'anonymat puisque celui-ci peut relier les différentes adresses grâce à leur adresse MAC. Les adresses MAC construites dans l'espace d'adresse "administré localement" permettent également d'identifier le nœud quand il est le seul à disposer de la protection des *CGA Éphémères Anonymes* sur son lien.

D'un point de vue plus technique, notre solution *CGA Éphémères Anonymes* repose sur l'utilisation du *promiscuous mode* sur les cartes réseaux. Ce mode peut généralement être détecté par les nœuds voisins par un test simple : il suffit de *pinguer* une des adresses IPv6 du nœud en utilisant une adresse MAC qui n'est pas la sienne. Un nœud en *promiscuous mode* sera révélé puisqu'il répondra à la requête. Nous proposons, pour éviter cette détection, qu'il soit mis en place sur le nœud un filtrage (c.-à-d. par un pare-feu de niveau 2) pour que le nœud ne réponde seulement qu'aux messages ayant un couple adresse MAC - adresse *CGA Éphémères Anonymes* valide (et de ne pas traiter le paquet dans le cas contraire). Cette vérification peut être réalisée très rapidement puisque l'adresse MAC est dérivée à partir de l'adresse IPv6 du nœud (voir la Section 8.3.8).

Le protocole SEND permet aux nœuds avec une horloge légèrement désynchronisée (par défaut un maximum de 10 minutes de décalage) de communiquer entre eux. Cela

implique qu'il est souvent possible de différencier les nœuds en fonction la valeur de leur horloge indiquée dans le contenu de l'option *horodatage* du protocole SEND. Dans notre solution, les messages NDP émis depuis des adresses *CGA Éphémères Pseudo-Anonymes* et *CGA Éphémères Anonymes* sont protégés par l'option *horodatage* pour éviter les rejeux. Si aucune précaution n'est prise pour protéger le contenu de l'horodatage, un attaquant dans le voisinage d'un nœud protégé avec les *CGA Éphémères* peut capturer les messages émis depuis ses différentes adresses, et par la suite, grâce à une fine analyse du contenu, grouper les adresses lui appartenant (celle qui sont protégées par des options *horodatage* qui indiquent la même valeur d'horloge). Afin d'éviter cette vulnérabilité, nous proposons que chaque *CGA Éphémère Pseudo-Anonyme* et chaque *CGA Éphémère Anonyme* se voit attribuer une valeur aléatoire, conservée tout au long de la connexion, qui sera ajoutée l'option *horodatage* lors de l'envoi de messages sécurisés par SEND.

Nous avons noté qu'en fonction de la technologie de lien, les données émises peuvent être identifiables grâce à leur adresse de couche liaison. À cet effet, nous avons proposé un mécanisme permettant de dériver de nouvelles adresses de couche liaison. Cependant, cette solution n'est pas applicable à tous les types de lien. En effet, il existe des liens, comme les liens Point-à-Point, où la connexion est établie entre deux entités et où chaque extrémité est facilement identifiable (c.-à-d. de part la nature du lien).

Afin de ralentir le travail de corrélation entre les adresses qui peut être effectué grâce aux informations transportées par les couches supérieures à la couche 3 (par ex. un flux HTTP, etc.), nous proposons qu'un nœud utilisant les *CGA Éphémères Pseudo-Anonymes* ou les *CGA Éphémères Anonymes* partage son trafic entre les différents routeurs d'un réseau quand cela est possible (c.-à-d. quand plusieurs routeurs par défaut sont disponibles sur le lien), en prenant soin de ne montrer à chaque routeur qu'un sous-ensemble disjoint de son ensemble d'adresses. Ainsi, il devient plus difficile pour un attaquant ayant corrompu un ou plusieurs routeurs d'exploiter des traces du trafic émis par le nœud pour retracer toutes les *CGA Éphèmères* d'un même nœud.

Pour conclure cette section, nous rappelons que ni les *CGA Éphémères Pseudo-Anonymes* ni les *CGA Éphémères Anonymes* n'empêchent la corrélation d'informations qui peut être réalisée par l'analyse des informations supérieures à la couche 3 (par ex. à la couche application).

8.3.11 Analyse de la sécurité de notre approche

Les documents [Aur05, MC04] présentent les mécanismes de protection des adresses CGA. Nos propositions *CGA Éphémères Pseudo-Anonymes* et *CGA Éphémères Anonymes* étant basées sur les CGA, nous inférons que nous disposons des mêmes propriétés de sécurité.

Cependant, le déploiement "classique" du protocole SEND et nos approches diffèrent dans le niveau de protection qu'elles offrent. Dans le protocole SEND, le nœud ne dispose généralement que d'une seule adresse pour toutes ses connexions. Dans le cas des *CGA Éphémères*, notre nœud utilise une adresse CGA par connexion.

Regardons les types d'attaques qui peuvent se produire dans le modèle de sécurité de SEND et analysons leur impact sur nos propositions :

– une adresse CGA est cassée sans que sa clé privée n'ait été divulguée (c.-à-d. l'attaquant a réussi à trouver une collision avec l'adresse). Dans un déploiement classique du protocole SEND, toutes les connexions se basant sur l'adresse CGA, c'est à dire toutes les connexions du nœud, sont affectées. Dans le cas où une *CGA Éphémère Pseudo-Anonyme* ou une *CGA Éphémère Anonyme* est cassée de la même façon, aucune des autres *CGA Éphémères* n'est affectée. Cela signifie que toutes les connexions, à l'exception de celle associée avec l'adresse CGA qui vient d'être cassée, restent protégées. De plus, la durée de validité de l'attaque dépend de la durée de validité de la *CGA Éphémère* affectée, c'est à dire la durée de vie d'une connexion TCP/UDP ;
– une adresse CGA est cassée et la clé privée associée est révélée. Les conséquences pour un déploiement classique de SEND restent les mêmes : toutes les connexions peuvent être perturbées. Pour les *CGA Éphémères Pseudo-Anonymes*, toutes les adresses CGA (qui sont toutes dérivées à partir de la même clé publique) sont également cassées. Pour les *CGA Éphémères Anonymes*, seule l'adresse CGA liée à la clé privée qui vient d'être divulguée est perturbée. Les autres *CGA Éphémères Anonymes* restent non affectées, puisqu'elles dépendent d'autres clés privées.

8.3.12 Comparaison avec les autres mécanismes de génération d'adresses

Nos deux approches se focalisent principalement sur l'aspect anonymat. La solution DHCVv6 [DBV+03], décrite dans la Section 8.2.1, ne traite pas cet aspect, ce qui l'exclut de la comparaison. Le document *Privacy Extensions for Stateless Address Autoconfiguration in IPv6* [NDK07] offre une protection de la vie privée basique (décrite Section 8.2.3), mais n'offre pas de protection du Pseudo-Anonymat ou de l'Anonymat. En effet, lors de l'utilisation d'adresses *temporaires*, plusieurs connexions peuvent utiliser la même adresse source et permettre à un attaquant de tracer les activités du nœud (corréler la visite de différents sites Web ou la lecture de plusieurs boites). La solution basée sur les *adresses IPv6 éphémères* [KAM09b], décrite dans la Section 8.2.4 n'offre aucune protection de l'anonymat des adresses au niveau du lien, en ce point, elles sont similaires aux *CGA Éphémères Pseudo-Anonymes*. Cependant, elles nécessitent d'importantes modifications dans le système d'exploitation et de profonds changements dans les couches réseau et transport que ce soit au niveau du nœud utilisateur que de ses correspondants. Pour finir, FLASCHE [Zug05] (présenté Section 8.2.5) fournit un mécanisme de génération très proche de notre solution, mais ne discute pas des performances de la solution. De plus, la solution n'offre aucune protection contre l'usurpation d'adresse.

Le Tableau 8.1 offre une comparaison entre les différents mécanismes de génération d'adresses IPv6.

Nom de la solution	Durée de vie des adresses	Anonymat à un saut	Anonymat à plusieurs sauts	Modification des nœuds correspondants	Protection de l'adresse (usurpation, etc.)
DHCPv6 [DBV+03]	fixée lors de l'attribution, possiblement illimitée	non	non	non	non
Autoconfiguration d'Adresse Sans État [TNJ07]	fixée lors de la découverte du préfixe, possiblement illimitée	non	non	non	oui en utilisant les CGA + SEND
Adresses temporaires [NDK07]	durée fixe prédéterminée	non	non, mais difficulté de lier plusieurs connexions à travers une longue durée	non	oui en utilisant les CGA + SEND
Adresses IPv6 Éphémères [KAM09b]	limitée à la durée de vie de la connexion	non	oui	oui	non
FLASCHE [Zug05]	limitée à la durée de vie de la connexion	oui	oui	non	non
CGA Éphémères Pseudo-Anonymes	limitée à durée de vie de la connexion	non	oui	non	oui
CGA Éphémères Anonymes	limitée à durée de vie de la connexion	oui	oui	non	oui

Tableau 8.1: Comparatif entre les différents mécanismes de génération d'adresses IPv6

8.4 Synthèse du chapitre

Ce chapitre présente deux mécanismes de génération d'adresses en IPv6. Le premier, *CGA Éphémères Pseudo-Anonymes*, fournit l'anonymat au delà du premier saut en liant une adresse CGA (adresse IPv6) à une connexion tandis que le second, *CGA Éphémères Anonymes*, offre un anonymat de la connexion dès le premier saut en proposant un identifiant de niveau 2 (adresse MAC) unique par connexion. Ces nouveaux mécanismes sont comparés aux mécanismes déjà existants (*Autoconfiguration d'Adresse Sans État*, DHCPv6, etc.). Par rapport à ceux-ci, les *CGA Éphémères* permettent d'assurer la sécurité de l'adresse (protection contre l'usurpation, etc.) grâce à l'utilisation des CGA et du protocole SEND. De plus, nos mécanismes ne nécessitent pas de modifications sur les nœuds correspondants et intermédiaires et restent totalement transparents pour les couches supérieures à la couche 3 (par ex. UDP, TCP, HTTP, etc.), ce qui permet une adoption immédiate de la solution.

Une implémentation dans un greffon de NDprotector prouve la faisabilité des *CGA Éphémères Pseudo-Anonymes*. Ce résultat est appuyé par une étude des performances qui confirme que l'utilisation des adresses *CGA Éphémères* est réaliste. Nous prévoyons également de réaliser une implémentation des *CGA Éphémères Anonymes* afin de pouvoir évaluer leurs performances et de déterminer leur pertinence.

9 Utilisation des identifiants cryptographiques pour protéger les applications à plusieurs sauts

Comme nous l'avons mentionné dans le Chapitre 2, les adresses CGA sont dérivées du concept de *Crypto-Based Identifiers* (CBIDs). Il s'agit là d'un concept vaste qui inclut les identifiants *Statistical Uniqueness and Cryptographic Verifiability* (dont 125 des 128 bits sont composés d'un condensât de la clé publique) et les adresses *Statistical Uniqueness and Cryptographic Verifiability* (SUCV) (dont 61 bits des 64 bits de l'identifiant d'interface sont composés d'un condensât de la clé publique). Les CBIDs ont été largement étudiés dans la littérature et de nombreuses propositions pour sécuriser les applications à plusieurs sauts en s'appuyant sur eux ont été proposées. Ces solutions s'appliquent principalement quand aucune infrastructure de sécurité classique n'est présente (PKI, tierce partie, etc.).

La première partie du chapitre présente les applications des CBIDs fonctionnant à plus d'un saut qui sont décrites dans la littérature : la gestion sécurisée des abonnements sur les groupes anycast et multicast (Section 9.1.1), la création de groupe de confiance dans les réseaux ad-hoc (Section 9.1.2), le chiffrement opportuniste (Section 9.1.3), la preuve de l'origine des messages (Section 9.1.4), un mécanisme d'échange de clés publiques (Section 9.1.5) et la protection du stockage de données distribué (Section 9.1.6).

Une seconde partie du chapitre présente notre contribution sur la diffusion de clés publiques dans les réseaux sans infrastructure : celle-ci est nommée *Trustful Authentication and Key Exchange Scheme* (TAKES). TAKES crée un lien entre le nœud (l'équipement physique), son adresse, sa clé publique et son propriétaire (personne physique). Nous verrons comment tirer avantage de ce lien pour sécuriser de multiples applications (routage, tunnel sécurisé, email, etc.) dans un réseau sans infrastructure. Tout d'abord nous présentons les travaux similaires (Section 9.2.1). Puis, nous positionnons la solution parmi ces travaux (Section 9.2.2) et en fournissons une vue d'ensemble (Section 9.2.3). Nous décrivons ensuite les différents scénarios d'applications (Section 9.2.4) ainsi que les messages TAKES qui permettent de les réaliser (Section 9.2.5 et 9.2.6). Afin de prouver la robustesse de notre solution, nous analysons sa sécurité (Section 9.2.7) et fournissons une preuve formelle de ses messages à l'aide de la logique BAN (Section 9.2.8). Pour finir, nous présentons une implémentation de TAKES (Section 9.2.9).

9.1 Solutions basées sur les CGA pour sécuriser les connexions à plusieurs sauts

9.1.1 Sécuriser la gestion de groupes avec les CGA

Le document [CM03] propose un mécanisme sécurisé de gestion des abonnements des groupes anycast et multicast basé sur une variante des adresses CGA nommée pour l'occasion Group-CGA (G-CGA). Les auteurs proposent de protéger la version 2 du protocole de *Multicast Listener Discovery* (MLD). Ce protocole est utilisé par les routeurs multicast pour déterminer sur lequel de leurs liens sont directement attachés les abonnés de leur groupe multicast dans les réseaux IPv6. Quand un abonné signale sa présence au routeur multicast, ce dernier commence à recopier les flux à destination du groupe multicast de l'abonné sur le lien qui les relie. Quand l'abonné signale son départ, le routeur vérifie s'il n'existe plus d'autres abonnés sur le même lien. Dans le cas de l'absence d'abonnés sur le lien, le routeur arrête la recopie du flux. On peut rapidement imaginer que, si le MLD n'est pas protégé, il est vulnérable à de nombreuses attaques. Par exemple, un attaquant peut faire abonner de nombreux liens à un groupe multicast afin de saturer le réseau.

Le document propose également de sécuriser les groupes anycast. Il ne s'agit pas ici d'adressage anycast au niveau du lien (comme présenté Section 7.2.1) mais de routage anycast. Le fonctionnement est similaire au routage multicast (et également géré par le MLD, voir [HT02]) si ce n'est que le message n'est distribué qu'à un seul des abonnés (généralement, celui situé le plus près géographiquement de la source). Par ailleurs, on constate ici qu'un attaquant qui s'abonne à un groupe a plus de pouvoir que précédemment : il capture des paquets qui, au contraire des abonnés des groupes multicast, ne sont pas répliqués sur les autres abonnés et peut ainsi démarrer un déni de services.

Les G-CGA sont divisées en deux catégories : les Multicast-CGA, ou M-CGA (à ne pas confondre avec les adresses M-CGA définies dans la Section 7.1.4.3), qui protègent les groupes multicast et les Anycast-CGA, ou A-CGA, qui protègent les groupes anycast.

Les M-CGA suivent le format d'une adresse multicast, illustré Figure 9.1, tel que défini dans le document "Architecture d'adressage de l'IPv6" [HD06]. Les auteurs proposent ici de lier la M-CGA à une clé publique en stockant un condensât de la clé dans la partie *identifiant du groupe* de l'adresse. Les A-CGA sont quant à elles construites comme les adresses CGA normales [Aur05].

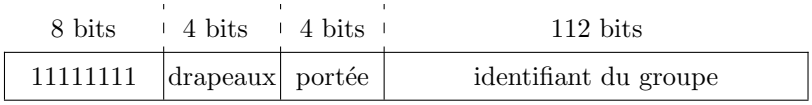


FIGURE 9.1: Format d'une adresse multicast tel que décrit dans le document [HD06]

Par la suite, la mise en place d'un groupe (multicast ou anycast) fonctionne de la

manière suivante :

- le contrôleur du groupe construit une paire de clés et en dérive l'adresse G-CGA (M-CGA ou A-CGA) du groupe à partir de la clé publique ;
- pour chaque abonné autorisé, le contrôleur du groupe génère un certificat SPKI (*Simple Public Key Infrastructure*) [EFL+99] [1] qui autorise l'adresse CGA de l'hôte à joindre le groupe. Ce certificat contient, entre autres, l'adresse M-CGA ou A-CGA du groupe, une période de validité et l'adresse CGA du membre qui reçoit l'autorisation ;
- lorsqu'un hôte souhaite joindre ou quitter le groupe, il émet un message indiquant sa requête depuis sa CGA et y joint son certificat, sa clé publique (celle liée à son adresse CGA) et signe le message avec la clé privée associée ;
- le routeur multicast qui reçoit un tel message vérifie la preuve de l'appartenance du nœud au groupe. Cela implique que le certificat est valide, que l'adresse CGA contenue dans le certificat est la même que l'adresse d'origine du message et que la signature du message est bien réalisée par la clé privée associée à cette CGA.

À la fin de cet échange, l'abonné est ajouté au groupe.

9.1.2 Sécurisation des réseaux ad-hoc et des réseaux pair-à-pair

Dans les réseaux ad-hoc et pair-à-pair, chaque nœud agit comme un routeur et relaye des paquets pour ses voisins. Ainsi, le nœud est dépendant de ses voisins pour assurer le routage de son propre trafic. Cela implique qu'il lui est nécessaire de faire confiance à ses voisins. La cryptographie apporte une réponse à cet épineux problème en créant des groupes au sein du réseau. Les nœuds appartenant au même groupe établissent une relation de sécurité et se font confiance. Il s'agit bien souvent de mécanismes basés sur les PKI ou sur une distribution de clés et de secrets. Ainsi, les messages de routage qui émanent des membres du groupes sont privilégiés par rapport aux messages émis par les nœuds n'appartenant pas au groupe.

L'intérêt de la solution décrite dans le document [MC04] est de protéger l'identité des nœuds et la création de groupes dans les réseaux mobiles ad-hoc grâce à l'utilisation des *Crypto-Based Identifiers* (CBIDs). Les nœuds se voient alors attribuer des CBIDs (les réseaux ad-hoc sont plus laxistes quant au format de l'adresse) qui sont utilisés pour prouver l'origine des messages de routage (signature avec la clé privée correspondant au CBID, etc.). D'une manière similaire à la Section 9.1.1, la création et la gestion des groupes sont sécurisées par l'utilisation de contrôleurs de groupes qui génèrent des certificats autorisant les nœuds à faire partie du groupe. Ainsi, le groupe est lui-même un CBID (CBID_G) et le contrôleur de groupe se sert de la clé privée associée à ce CBID (CBID_G) pour signer les certificats distribués aux nœuds. Chaque certificat contient le CBID (CBID_N) du nœud à autoriser, le CBID du groupe (CBID_G) et une durée de validité. Ces éléments permettent par la suite au nœud de prouver son appartenance au groupe en exhibant son certificat et en signant grâce à la clé privée associée à son CBID (CBID_N).

1. le format du certificat SPKI est un format qui se veut plus simple que les certificats *X.509* et ne définit pas d'autorité de certification.

9.1.3 Chiffrement opportuniste de la connexion

Les documents [LM08] et [CM02] présentent deux mécanismes de chiffrement opportuniste des connexions. Le terme “opportuniste” indique que la connexion sécurisée s'établit entre deux nœuds qui n'ont aucune relation de confiance et aucun secret partagé. Cependant, il est possible de prouver à partir d'une des extrémités que ce qui transite de manière chiffrée est bien émis à partir de l'autre extrémité de la connexion. Le terme “opportuniste” implique également que si la connexion authentifiée ne peut être établie (par ex. une des extrémités de la connexion ne fournit pas de mécanismes de sécurité), alors une connexion non-authentifiée est réalisée.

Les deux documents proposent une approche très similaire. Le premier [LM08] se base sur les adresses CGA et la deuxième version du protocole *Internet Key Exchange* (IKE) [HC98] pour l'échange de clé. Il combine une authentification IKE à base de CGA avec l'utilisation du protocole “opportunistic IPsec” [RR05] pour établir une connexion IPsec. Le second [CM02] utilise les adresses *Statistical Uniqueness and Cryptographic Verifiability* (SUCV) (un CBID dont les CGA sont dérivées) et le protocole d'échange de clé *Just Fast Keying* (JFK) [ABB+04] pour établir une session IPsec.

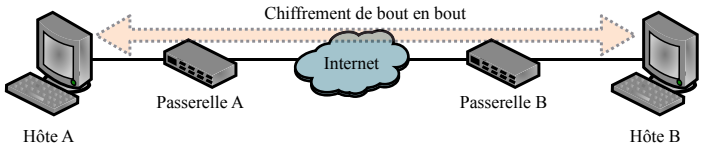


FIGURE 9.2: Chiffrement opportuniste de bout en bout entre deux hôtes

Deux modes de communication sont possibles :

- un mode transport de bout en bout par les nœuds (illustré Figure 9.2). Ce mode comporte comme inconvénient de laisser les adresses des deux nœuds visibles sur tout le chemin. La mise en place est très simple avec une négociation directe entre les deux extrémités ;
- un mode tunnel, où le tunnel est établi entre les passerelles placées sur les réseaux de chacun des nœuds. Chaque passerelle dispose d'adresses anycast réservées, ce qui permet à la passerelle de l'initiateur de la connexion d'apprendre quelle adresse contacter pour communiquer avec l'une des passerelles de sécurité du nœud correspondant. Chacune des passerelles dispose également d'un CBID, la paire de clés associée à celui-ci servira à négocier le tunnel.

Comme illustré par la Figure 9.3, c'est le premier paquet transmis d'un réseau vers l'autre qui déclenche la construction du tunnel lorsque les passerelles de sécurité sont configurées pour faire du chiffrement opportuniste. En effet, si ce paquet est émis à partir d'un CBID (par ex. depuis l'hôte A vers l'hôte B), il est intercepté par la passerelle (A) et est gardé en tampon pendant que la passerelle (A) essaye de joindre la passerelle présente sur le réseau de destination (passerelle B, contactée via son adresse anycast). Si une passerelle de sécurité est présente à l'autre extrémité, la preuve de possession du CBID de chaque nœud en extrémité est apportée par la passerelle

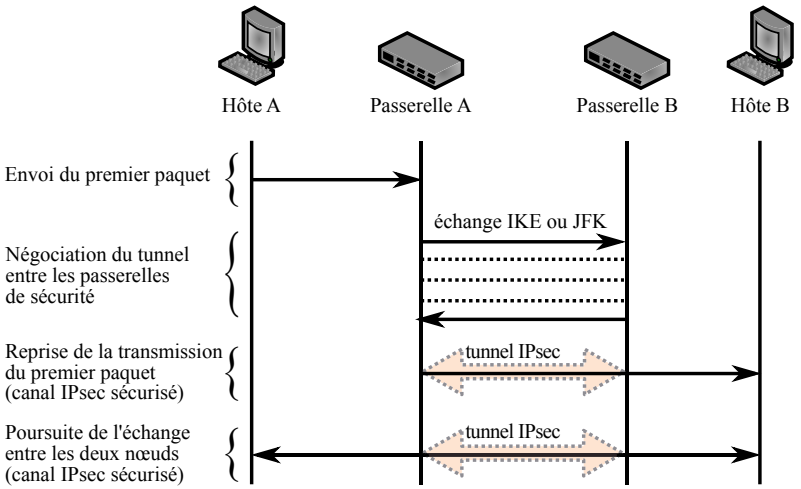


FIGURE 9.3: Chiffrement opportuniste entre deux passerelles de sécurité

présente sur son réseau (c.-à-d. passerelle A pour hôte A ou passerelle B pour hôte B). C'est cette preuve de possession qui différencie les approches [LM08] et [CM02] :

- le document [LM08] se base sur une preuve à distance de la possession de la CGA. Cette preuve est ensuite apportée à la passerelle de sécurité du réseau voisin ;
- cela contraste avec le document [CM02] qui exhibe un certificat SPKI [EFL+99] émis par le nœud et signé par la clé privée associée à son CBID pour permettre à la passerelle de sécurité située sur son réseau d'agir comme tel.

9.1.4 Preuve de l'origine de paquets IPv6 : la solution CGA-SEC

Le document [ZNW10] introduit un nouvel en-tête IPv6, nommé "extension CGA", qui permet d'ajouter les protections CGA (structure de données *CGA Parameters*, signature numérique, etc.) à chaque paquet IPv6. La motivation principale de cette solution vient du constat que le contrôle d'accès basé sur l'adresse source, bien que n'offrant aucune sécurité, est encore largement utilisé dans les réseaux. Les auteurs pensent qu'il serait alors intéressant de combiner la simplicité de ce type de contrôle d'accès à la preuve de possession d'adresses fournie par les CGA.

Toutefois, l'ajout de l'en-tête "extension CGA" est généralement coûteux (quand le calcul de la signature est nécessaire) et, bien que la vérification de la protection ne soit effectuée que par le destinataire du paquet, cela limite la solution à des scénarios particuliers où peu de messages sont émis. Ainsi, le document [ZNW10] indique que la solution est principalement contrainte à :

- permettre, la création d'une liste de contrôles d'accès pour les imprimantes et les équipements à faible puissance de calcul ne pouvant implémenter des mécanismes de sécurité complexes comme IPsec ;

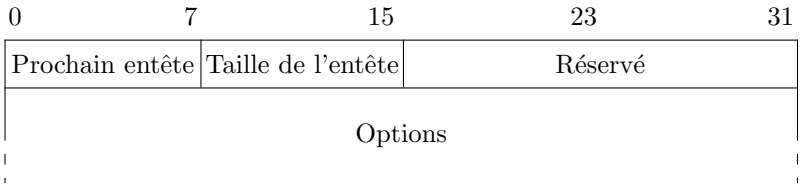


FIGURE 9.4: Format de l'entête "extension CGA"

- protéger la diffusion de messages dans les groupes multicast. La solution permet de vérifier que la source est bien autorisée à émettre avant de répliquer l'information qu'elle émet sur les différents routeurs abonnés au groupe ;
- protéger chaque message dans les réseaux cellulaires ou sans fils où les ressources en bande passante sont limitées ;
- prouver l'origine des messages de type *trap* dans le *Simple Network Management Protocol* (SNMP) ;
- mettre à jour de manière dynamique une entrée DNS. Cette mise à jour est effectuée par un nœud qui dérive ses multiples adresses CGA à partir de la même clé publique, et prouve alors dans ses messages de mise à jour DNS la possession de la clé privée ;
- et, de manière générale, protéger les messages de tous les nœuds qui émettent peu de trafic et qui n'ont pas les ressources pour intégrer un mécanisme de sécurité plus complexe.

L'en-tête d'extension CGA, illustré dans la Figure 9.4, permet le transport de trois options différentes :

- l'option "CGA Params" permet le transport de la structure de données *CGA Parameters*. Celle-ci contient, comme dans le protocole SEND [AKZN05], la clé publique qui permet la vérification de la signature ;
- l'option "CGA Signature" contient une signature IPv6 effectuée sur les champs adresse source, adresse destination et la charge du paquet IPv6. La signature est effectuée avec la clé privée associée à la clé publique transportée dans l'option "CGA Params" ;
- l'option "requête CGA" demande à un correspondant de joindre à son prochain message une option "CGA Signature". Un champ "numéro de séquence" est présent, il permet d'associer l'option "requête CGA" à l'option "CGA Params" de la réponse.

En combinant ces trois options, deux modèles de communication sont possibles : vérification uniquement de l'adresse source du correspondant, lorsque le nœud joint une option "requête CGA" seule, ou vérification bidirectionnelle de l'échange, lorsque les deux nœuds joignent chacune des trois options à chaque paquet.

La solution propose également l'usage de nouveaux messages ICMP pour indiquer les erreurs lors du décodage de l'en-tête "extension CGA" (signature non valide, etc.).

Il est à noter qu'un des points forts de l'utilisation de liste de contrôle d'accès est

que celles-ci se basent généralement sur l'autorisation d'un sous-ensemble d'adresses au sein du réseau, ce qui fait de la mise en place d'un filtre, une opération très rapide. Ici, l'utilisation des CGA force à autoriser chaque adresse explicitement puisque celle-ci est générée de manière aléatoire. Cela diminue l'intérêt de la solution qui se propose initialement de fournir une solution simple pour sécuriser les listes de contrôle d'accès.

9.1.5 Distribution de clés publiques

Le document [MC04] présente un mécanisme d'échange de clés publiques pour les réseaux ad-hoc basé sur les adresses et les identifiants *Statistical Uniqueness and Cryptographic Verifiability* (SUCV) [2].

Cette méthode de distribution de clés fait intervenir les opérateurs humains qui possèdent les équipements à authentifier. Ces opérateurs, à la différence des machines, peuvent facilement évaluer l'authenticité et reconnaître le contenu de leurs échanges.

La méthode proposée fait utilisation de ce canal et le combine à l'utilisation d'un dictionnaire de mots de passe à usage unique (*One Time Password dictionary*, OTP) dont le fonctionnement est décrit dans le document [HMNS98]. Ce dictionnaire contient 2048 (2^{11}) mots qui peuvent chacun être codé sur une valeur de 11 bits. Ainsi, une phrase (suite de mots) peut habilement transporter plusieurs centaines de bits de contenu.

Ensuite, l'échange de messages, ici illustré entre les utilisateurs A et B, est de la forme suivante :

- le nœud de A diffuse un message indiquant qu'il souhaite apprendre la clé publique de son correspondant B ;
- le nœud de B répond alors au nœud de A avec un message contenant sa clé publique, et signé par sa clé privée ;
- à la réception de ce message, le nœud de A peut vérifier la signature et savoir qu'elle a bien été effectuée avec la clé privée de B ;
- pour terminer l'échange, seul le canal authentifié entre les deux utilisateurs (qui dans notre exemple est la voix) est utilisé. L'utilisateur A demande à B de lui lire la phrase (au sens OTP) correspondant à l'identifiant de son nœud (c.-à-d. son adresse ou identifiant SUCV). Cela confirme ainsi que le message authentifié qu'il a reçu a bien été envoyé par B.

À la fin de cet échange, le nœud A a appris la clé publique du nœud B. De manière triviale, le document [MC04] propose également d'étendre le mécanisme de façon à ce qu'il devienne symétrique et que l'échange de clé soit bidirectionnel.

Par ailleurs, la connaissance de la clé publique et de l'adresse du nœud permettent d'améliorer d'autres applications de sécurité basées sur les SUCV, comme le chiffrement opportuniste de la connexion (Sec. 9.1.3).

2. Les adresses SUCV sont très proches des adresses CGA et seuls quelques détails, comme l'absence des bits SEC, les différencient.

9.1.6 Protection du protocole de stockage distribué IBP

Le document [BL03] décrit une combinaison de plusieurs mécanismes permettant la sécurisation du protocole de stockage partagé et distribué *Internet Backplane Protocol* (IBP) [BBM+03]. Dans ce protocole, une application a la possibilité de demander, quand nécessaire, l'allocation d'un espace de stockage, ici appelé "dépôt". Toutefois, cette allocation n'a qu'une durée de vie temporaire et une fois celle-ci écoulée, l'espace est désalloué. Dans ce document, la sécurité de la fonction d'allocation est étudiée et améliorée. En effet, celle-ci est par défaut protégée par un mécanisme de liste de contrôle d'accès, connu pour ses faiblesses en terme de sécurité. Les auteurs traitent rapidement de la sécurité des autres fonctions de manipulation de l'espace de stockage (méthodes *get* et *put*) : ces messages sont considérés comme moins vulnérables car protégés par un secret obtenu lors de l'allocation de l'espace.

La solution combine trois mécanismes de sécurité : les adresses CBID, la protection d'IPsec au niveau transport (protocole *Encapsulating Security Payload*, ESP) et les certificats SPKI. L'utilisation des couples CBID/SPKI et CBID/IPsec rappelle par beaucoup d'aspects les Sections 9.1.1 et 9.1.3.

Lorsque le possesseur du dépôt IBP souhaite protéger sa ressource, il génère un CBID qui représente son dépôt. Ce CBID est lié à un certificat SPKI (en signant celui-ci grâce à la clé privée associée au CBID). Par la suite, ce certificat lui permet signer de nouveaux certificats qui autorisent leur propriétaire à faire une demande d'allocation de ressources auprès de ce dépôt. Ces certificats contiennent la durée de vie et la capacité maximale du futur espace alloué. Afin que le possesseur du dépôt n'ait pas à signer lui-même tous ces certificats, il lui est également possible de déléguer à certains certificats la capacité de signer de nouveaux certificats pour qu'ils soient autorisés à accéder au dépôt. Ces nouveaux certificats peuvent eux-mêmes déléguer à nouveau cette autorisation. Le possesseur du dépôt peut de cette façon donner naissance à une chaîne de certification dont il est l'origine (ou l'ancre). Dans ce cas, pour vérifier l'autorisation d'un nœud, il est nécessaire de vérifier chaque certificat de la chaîne de certification depuis le certificat du demandeur de la ressource jusqu'au certificat associé au CBID du dépôt.

En parallèle, une politique de sécurité est mise en place pour utiliser le mode transport du protocole ESP lors de l'envoi d'un premier paquet à destination du port TCP correspondant au protocole IBP. Ainsi, lors de la demande d'allocation, le CBID du nœud (par ex. son adresse CGA) lui permet de démarrer une association IPsec (comme nous l'avons vu dans la Section 9.1.3). Ce tunnel assurera l'intégrité et la confidentialité des messages durant la demande d'allocation de la ressource. Conjointement à cette utilisation d'IPsec, le nœud joint son certificat SPKI à chaque demande afin de prouver son droit d'accès à la ressource.

9.2 Diffusion de clés publiques à plusieurs sauts : Trustful Authentication and Key Exchange Scheme (TAKES)

Cette section décrit notre contribution nommée *Trustful Authentication and Key Exchange Scheme* (TAKES), une application multi-sauts des CBID (et des CGA) qui permet la diffusion de clés publiques au sein d'un réseau mobile sans infrastructure (*Mobile Ad Hoc Network*, MANET).

9.2.1 Travaux similaires

La sécurité des réseaux ad-hoc déconnectés (c.-à-d. du réseau Internet) est un sujet de recherche vaste qui s'est considérablement développé durant ces dernières années. La principale difficulté, l'absence de tierce partie de confiance (TTP), rend impossible la mise en place d'un système de PKI classique et donc l'utilisation des systèmes de sécurité à base de clés publiques. Afin de lever cet obstacle, de nouveaux mécanismes de distribution de clés publiques dans les réseaux ad-hoc ont été étudiés. Ces solutions sont divisées en trois catégories : celles basées sur la cryptographie à seuil [YK03, DMA04], ou plus récemment les solutions faisant usage de l'*Identity Based Cryptography* [DMA04] (IBC) et de l'utilisation de canaux hors-bande (*Out Of Band Channels*, OOBC) [PS99, FAWD02, BSSW02, MW07, SU08, MC04].

Basé sur la cryptographie à seuil, MOCA [YK03] fournit une solution pour la création et l'utilisation d'une PKI distribuée dans un réseau MANET déconnecté. Le rôle de l'autorité de confiance est distribuée à un sous ensemble de terminaux mobiles composant le réseau. Dans le document [DMA04], Deng et al. se basent, d'une part, sur la cryptographie à seuil pour mettre en place un *Private Key Generator* distribué, et sur l'*Identity Based Cryptography*, d'autre part, pour que chaque nœud puisse prouver l'origine de ses messages et sécuriser ses communications avec les autres nœuds. Enfin, Larafa et al. [LLM09] proposent un mécanisme de contrôle d'accès dans un réseau MANET. La cryptographie à seuil permet de distribuer le service AAA actif à l'arrivée d'un nouveau noeud dans le réseau. Les noeuds AAA réalisent l'authentification du noeud nouvellement arrivée et lui génèrent un jeton incluant l'adresse CGA du noeud. Ensuite, chaque paquet émis par ce noeud contient le jeton, ce qui permet aux noeuds du réseau de vérifier l'autorisation de ce noeud d'émettre du trafic dans le réseau MANET.

Toutefois, ces solutions supposent l'existence d'un lien de confiance entre les nœuds d'un réseau (c.-à-d. que la majorité des nœuds soient honnêtes), ce qui est une hypothèse forte et rarement vérifiable.

Quand les nœuds d'un réseau ad-hoc non connecté ne peuvent pas se faire confiance, il reste un vecteur par lequel la sécurité peut être apportée : il s'agit de l'humain, l'utilisateur associé à chaque nœud qui compose le réseau ad-hoc. C'est ce facteur qui va déterminer le niveau de sécurité du système. En effet, il faut faire un compromis entre la sécurité et la facilité d'utilisation et donc le taux d'acceptation de la solution auprès des utilisateurs. Par exemple, un mot de passe simple à retenir sera vulnérable

aux attaques par force brute tandis qu'un mot de passe sûr sera difficile à retenir. Toutefois, les humains sont meilleurs à d'autres aspects, comme la reconnaissance de formes, l'authentification d'un autre humain lors d'un réunion face à face, etc. Ainsi, des solutions exploitant ces qualités ont été définies. Dans leur travail sur les *Random Arts* [PS99], Perrig et Song définissent une solution où une clé publique est transformée en image afin d'être validée par l'utilisateur.

Plus robustes, les solutions basées sur le couplage et l'utilisation de canaux hors-bande authentifiés permettent également l'échange de clés publiques. Les canaux hors-bande, de part leur nature (voix, équipement optique en visibilité directe, etc.), sont authentifiés et permettent le transport d'informations en clair. Feeny et al.[FAWD02] utilisent une communication infrarouge dans le but de transporter le matériel cryptographique (clé publique, etc) entre deux nœuds désirant établir un lien de confiance. Balfanz et al. [BSSW02] proposent un mécanisme qui repose également sur les infrarouges, mais cette fois-ci le canal hors-bande transporte seulement des informations sensibles durant le début de l'échange. Par la suite, un canal non-sécurisé (par ex. canal sans fil) est utilisé pour terminer l'authentification et réaliser un échange de clés. Ces techniques sont généralisables à de nombreux canaux, que ce soit l'utilisation de LEDs et d'une Webcam [SU08] ou l'utilisation d'un LASER [MW07]. L'intérêt de ces solutions est de pouvoir contrôler la distance (physique) maximale entre les différents participants.

Finalement, d'autres solutions reposent sur l'utilisateur comme canal hors-bande. La solution que nous allons développer en fait partie. En ce sens, elle rejoint la solution présentée Section 9.1.5 ([MC04]).

9.2.2 Positionnement de la solution

Nous souhaitons contribuer aux mécanismes d'échange de clés publiques à travers les réseaux Ad Hoc déconnectés. Plus précisément, nous voulons permettre la mise en place de nouvelles applications, où la diffusion d'une seule clé publique (et non pas l'échange de clés) à plusieurs nœuds d'un réseau est nécessaire. Ainsi, notre mécanisme ne diffuse qu'une seule clé publique à la fois, bien qu'il soit possible de l'exécuter plusieurs fois quand plusieurs nœuds souhaitent procéder à un échange de clés. Nous souhaitons également créer une association forte entre l'utilisateur, sa clé publique, son équipement et son adresse.

Afin de faciliter la mise en place de notre solution, nous tenons également compte des limitations suivantes : la solution ne doit pas nécessiter de matériel spécialisé (à l'inverse de [SU08, MW07]), de contrainte physique comme être en visibilité directe (contrairement à [FAWD02, BSSW02]), ne doit être composée que d'actions simples à réaliser par un opérateur humain (contrairement à [MC04]), et qu'il soit possible de diffuser la clé à une assemblée d'une centaine d'utilisateurs (par ex. cas d'une conférence).

9.2.3 Vue d'ensemble de TAKES

Le mécanisme TAKES propose la diffusion de clés publiques et de l'identité de l'utilisateur à un ou plusieurs sauts au sein d'un réseau MANET déconnecté. Ici, nous désignons par le terme "participant" les nœuds qui diffusent leur clé publique avec TAKES, mais également ceux qui souhaitent apprendre les clés publiques des autres nœuds. Chaque participant est identifié par son CBID lié à la clé publique qu'il veut partager.

Lorsque l'un des participants souhaite diffuser sa clé publique, il émet successivement deux messages. Comme nous le verrons dans la Section 9.2.7.1, l'ordre d'émission de ces messages est important pour éviter les attaques de type homme du milieu. Le premier message, envoyé par le participant à tous les nœuds du réseau MANET (émission de type *broadcast*), diffuse la clé publique et est protégé par une signature effectuée avec la clé privée liée au CBID et une *passphrase* secrète à usage unique. Le succès de l'envoi de ce premier message dépend des mécanismes de diffusions multi-sauts mis en place au niveau du réseau MANET multicast [LWLL07]. Le second message est envoyé sur un canal hors-bande (*Out of Band Channel*, OOBC), tel que la voix, et contient seulement la *passphrase* secrète qui permet de vérifier l'authenticité du premier message et de l'associer à la personne physique ayant émis le second message. Puisque ce second message est envoyé à travers un canal hors-bande où les messages peuvent être facilement ignorés (par ex. la voix), nous considérons que le participant souhaitant partager sa clé attire l'attention des autres participants avant de transmettre son message afin que ceux-ci se préparent à recevoir le message. Après la réception de ces deux messages, les autres participants peuvent authentifier le contenu du premier message, protégé à la fois par le CBID et par le secret. Dans le cas où la validité du message est prouvée, le participant peut lier l'émetteur des messages (c.-à-d. une personne physique), sa clé publique, son équipement et son adresse. L'association de ces quatre éléments est stockée (c.-à-d. dans une base de données locale) par chaque participant pour une utilisation ultérieure.

Nous soulignons que TAKES repose sur une communication unidirectionnelle (à travers deux canaux différents) initiée depuis le participant qui diffuse sa clé publique à destination du ou des participants qui vont apprendre sa clé. Par conséquent, ceux-ci n'émettent aucune confirmation quant à la réception ou même la validité des messages.

Comme nous le verrons dans la section suivante, de nombreuses applications sont rendues possible grâce à cette diffusion de la clé publique.

9.2.4 Scénario et applications dans les réseaux MANET

Cette section détaille plusieurs scénarios d'utilisation de TAKES dans la vie de tous les jours. Nous montrons que de nombreuses applications peuvent bénéficier de la diffusion de la clé publique au sein d'un réseau MANET.

Dans les scénarios présentés, nous faisons les hypothèses suivantes :

– un niveau minimal de sécurité est requis en imposant une valeur minimale à la taille des clés utilisées par les CBID ;

– aucun accès à l'Internet et aucune infrastructure réseau (aucun point d'accès, routeur fixe, etc.) ne sont présents ;
– aucun service de confiance n'est accessible sur le réseau (DNS, DHCP, serveur de certification, etc.).

Pour chaque scénario, nous considérons qu'il existe un nombre **n** de participants sur un total **m** d'utilisateurs présents dans le réseau MANET.

Scénario I (1 $\rightarrow$ n) : Il s'agit d'une salle de conférence avec n personnes. L'orateur, qui est bien connu du public, souhaite rendre publiques des ressources contenues dans son ordinateur personnel. Toutefois, il souhaite prouver que les ressources qu'il offre sont bien celles qu'il a mises à disposition et non pas celles d'un attaquant présent dans l'assistance. Pour se faire, il utilise un serveur *web* installé sur son ordinateur, qui contient les documents qu'il souhaite partager, signés avec sa clé privée (c.-à-d. le serveur contient le fichier du document et un fichier de signature). Néanmoins, puisqu'il n'existe pas d'autorité de confiance pour gérer son identité, l'orateur doit trouver un nouveau moyen pour diffuser sa clé publique.

$\Rightarrow$ TAKES permet à l'orateur de diffuser sa clé publique dans l'assistance. Un seul message est diffusé sur le lien et chacun des n participants n'a qu'une seule opération à effectuer : entrer la *passphrase*.

Les applications qui peuvent directement bénéficier de cet échange peuvent être : les échanges de média et de fichiers, diffusion de messages simples, vérification d'emails protégés par la clé publique de l'orateur, etc.

Scénario II (1 $\leftrightarrow$ 1) : Deux employés appartenant à deux entreprises différentes collaborent sur un projet commun. Ces deux personnes n'ont jamais échangé de données (numériques) avant cette rencontre. Toutefois, ils se sont déjà rencontrés dans le passé et par conséquent connaissent et font confiance à l'identité (physique) de leur collaborateur. Ils souhaitent s'échanger des données tout en s'assurant de l'authenticité et de la confidentialité de l'échange, en établissant, par exemple, un tunnel sécurisé entre eux. Pour ce faire, ils souhaitent apprendre la clé publique de leur correspondant d'une manière simple et sécurisée.

$\Rightarrow$ chaque collaborateur procède à une exécution de TAKES (soit deux exécutions au total). À la fin de ces exécutions, chacun des collaborateurs connaît la clé publique de son collègue.

Les applications décrites dans le scénario I peuvent également être mises en place ici. D'autres applications sont aussi envisageables , comme l'établissement d'un tunnel sécurisé ou la négociation d'un secret partagé.

Scénario III (n $\leftrightarrow$ n, n $\leq$ m avec n participants sur les m utilisateurs) : Dans une salle de réunion, m utilisateurs sont interconnectés. Bien qu'il s'agisse pour les participants de leur première rencontre, ils ont prévu d'échanger différents types de données (document, audio, vidéo, etc.) de manière sécurisée. S'il n'est pas nécessaire que le flux soit chiffré, ils désirent toutefois que les données soient au moins authentifiables. D'autre part, cette salle de réunion fait parti d'un réseau MANET plus vaste.

Les personnes présentes souhaitent pouvoir former un sous-groupe protégé au sein du MANET afin d'améliorer la robustesse de leur protocole de routage.

⇒ chacun des participants procède à une exécution de TAKES (soit m exécutions au total). À la fin de ces échanges, chacun des membres connaît la clé publique des m - *1* autres membres.

Les applications décrites dans les scénarios I et II peuvent également être mises en place ici. De nouvelles applications sont possibles, comme la mise en place d'un routage sécurisé au sein du MANET ou la génération d'une clé de groupe.

Scénario IV, les participants ne sont plus réunis : Une fois la collaboration entre les membres du MANET terminée, il est souhaitable de réutiliser le matériel cryptographique précédemment appris. Par exemple, les utilisateurs peuvent utiliser une clé publique d'un participant pour lui envoyer un courrier électronique. Ou encore, la clé publique peut servir à établir des tunnels chiffrés ou des réseaux privés virtuels, même lorsque les participants ne sont plus présents sur le même réseau. En réalité, la plupart des applications qui font usage de la cryptographie à clé publique peuvent bénéficier de la diffusion de la clé publique par TAKES.

Par ailleurs, il peut être intéressant, même quand aucun système de nom n'est présent ou qu'un ancien participant n'est inscrit dans aucun système de nom, de pouvoir le contacter quand il s'est déplacé dans un autre réseau.

⇒ l'exécution de TAKES dans les scénarios I, II et III a permis à TAKES d'apprendre les éléments publiques générant le CBID, dont la clé publique. Ces éléments peuvent être utilisés, même quand les participants ne sont plus présents dans le même réseau. En outre, s'il est possible de savoir de manière authentifiée qu'un de ces paramètres a été modifié, il est également possible de calculer le nouveau CBID. Ainsi, quand un participant change de réseau, l'un des avantages de l'utilisation de TAKES est de pouvoir connaître son nouvel identifiant en recalculant son CBID avec son nouveau préfixe réseau.

9.2.5 Détails du protocole TAKES

La Figure 9.5 présente les messages envoyés pour assurer la diffusion de la clé publique. Il s'agit là d'une communication unidirectionnelle qui s'adresse à tous les participants du protocole TAKES. Au terme de cette succession de messages, chaque récepteur apprend la clé publique de l'initiateur. De plus, il est capable de lier l'initiateur (une personne, ici référencé *user A*), son nom (*nameA*), son équipement communiquant (par ex. un ordinateur portable, PDA, téléphone intelligent, nommé *equipA*), son adresse IP (*@A*) et sa clé publique (*pkA*).

Le mécanisme de diffusion de la clé publique de TAKES se fait en 4 étapes :

1. l'adresse de l'initiateur est générée. L'adresse *@A* est un CBID, elle résulte de la concaténation des éléments suivants : un préfixe sous-réseau, *prefixA* - le résultat de l'application d'une fonction de hachage *F()* sur le préfixe *prefixA*, le nom de *user A nameA*, le nom de son équipement *equipA* (par ex. 'Notebook-A', 'PDA-A', 'Smartphone-A', etc) et sa clé publique *pkA* (de type RSA, ECC ou

autre). En d'autres termes, l'adresse *@A* est construite de la manière suivante (où *trunc64* est une fonction de troncature d'une chaîne à 64 bits et "|" est la fonction de concaténation) :

$$@A = [\text{prefixA} \mid \text{trunc64}(F(\text{prefixA, nameA, equipA, pkA}))]$$

Il est à noter que cette étape est généralement réalisée avant le début de l'échange. Elle n'a donc aucun impact sur le temps de déroulement de TAKES ;

2. le message est envoyé sur le réseau MANET : quand l'utilisateur *user A* souhaite partager sa clé publique, il envoie un message multicast (signé avec sa clé privée) sur le réseau. Ce message contient les éléments suivants : l'adresse de l'hôte A (*@A*), le nom de son propriétaire (*nameA*), le nom de l'équipement (*equipA*), un horodatage pour protéger des rejeux (*timestampA*), sa clé publique (*pkA*) et un HMAC [BCK96] calculé sur les données *pkA* et *timestampA* et protégé par le secret *secretA*. *secretA* est une *passphrase* à usage unique que l'utilisateur *user A* révélera par la suite aux participants via le canal authentifié hors bande ;
3. la transmission de la *passphrase* secrète sur le canal hors-bande (OOBC) : les récepteurs se voient proposer d'enregistrer la clé publique contenue dans le message. Pour se faire, ils doivent entrer la *passphrase* (*secretA*) et connaître le nom de l'utilisateur *user A* (*nameA*, qui sont tous les deux transmis de manière sécurisée au travers de l'OOBC. Cette OOBC peut revêtir la simple forme d'une communication orale (par ex. "Bonjour, mon nom est *userA* et ma *passphrase* est *secretA*") ;
4. le récepteur peut ensuite vérifier le message multicast en vérifiant deux éléments : que le HMAC est valide (ce qui prouve également le lien entre l'utilisateur *userA* et la clé publique *pkA*) et que la signature du message (sig_{pkA}) a bien été calculée avec la clé privée associée à la clé publique (*pkA*) contenue dans le message.

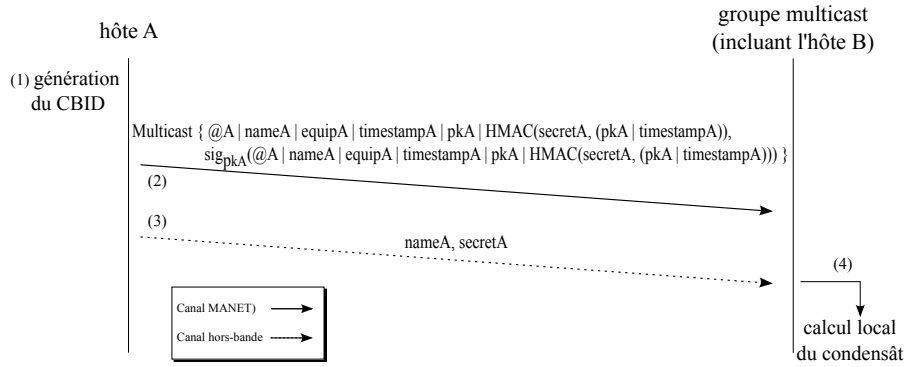


FIGURE 9.5: Messages envoyés pour effectuer la diffusion de la clé publique dans TAKES

Il est à noter que l'utilisation de l'horodatage nécessite une synchronisation entre les horloges de chaque nœud. Celle-ci peut-être réalisée par des protocoles spécialisés,

comme présenté dans les documents [SCHS07] et [CWH06]. Assurer la sécurité de ces protocoles de synchronisation est toutefois un problème à part entière. Une alternative plus souple serait d'utiliser l'algorithme de vérification de l'horodatage détaillé dans la spécification de SEND [AKZN05]. Les nœuds acceptent les messages dont l'horodatage ne présente pas une différence trop importante avec leur horloge. Cette différence entre l'horodatage et l'horloge est conservée pour les échanges futurs où le nœud vérifie que cette différence ne varie pas de manière plus importante qu'une dérive d'horloge classique.

9.2.6 Mise à jour et révocation de la clé

Nous complétons notre proposition par un mécanisme de mise à jour et de révocation de clés. Nous considérons la révocation comme un cas particulier de la mise à jour. En effet, le message de mise à jour permet la mise en place d'une nouvelle clé (*pkA'*) ainsi que la révocation de l'ancienne clé (*pkA*). Si aucune nouvelle clé n'est fournie, il s'agit alors d'une révocation simple de l'ancienne clé sans aucune mise à jour.

La Figure 9.6 illustre le contenu de ce message. Dans celui-ci, il n'est pas nécessaire de fournir tous les composants qui ont permis d'établir initialement l'identité, puisque celle-ci (par ex. *@A, nameA, equipA, pkA*) est déjà connue des participants dont on souhaite mettre à jour ou révoquer la clé. Les récepteurs du message utilisent l'adresse (source) *@A* pour trouver l'identité de l'émetteur et la clé publique *pkA* associée. Les récepteurs n'ayant aucune entrée correspondant à *@A* ne sont pas concernés par la mise à jour ou la révocation de la clé. L'authenticité du message est assurée par une signature effectuée par la clé privée correspondant à l'ancienne clé publique (*pkA*). À noter que le mécanisme de révocation n'a pas pour vocation d'être complet puisqu'il n'existe aucune autorité ou dépôt central permettant à un participant de s'assurer qu'une clé enregistrée en local n'a pas été révoquée.

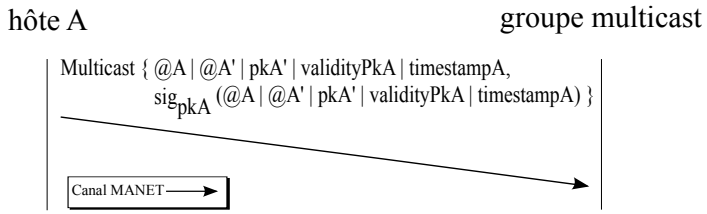


FIGURE 9.6: Message de mise à jour (et de révocation) de la clé

Les éléments de ce message sont les suivants :

1. l'adresse *@A* fournit aux récepteurs du message un moyen d'identifier l'identité de l'utilisateur *user A* dont la clé est mise à jour (ou révoquée) ;
2. l'adresse (source) *@A'* correspond à la nouvelle adresse de l'utilisateur *user A* générée à partir de sa nouvelle clé publique. Cette adresse est différente de *@A* si la clé publique est mise à jour et reste identique dans le cas d'une révocation ;

3. la clé publique *pkA'* correspond à la nouvelle clé publique de l'utilisateur *user A*. Dans le cas d'une révocation, il s'agit de la clé *pkA*. Dans le cas contraire, après la durée *validityPkA*, la clé est considérée comme active et utilisable immédiatement ;
4. la date de validité *validityPkA* indique quand l'ancienne clé publique *pkA* doit expirer. Si la date est une date passée ou l'heure actuelle, la clé *pkA* n'est plus valide. Si la date est une date dans le futur, l'ancienne clé peut encore être utilisée pour sécuriser des applications, en parallèle de la nouvelle clé (si fournie) ;
5. un horodatage, *timestampA*, protège des attaques par rejeu ;
6. la signature sig_{pkA} assure l'authenticité du message. À cet effet, la clé *pkA* qui permet de vérifier la signature reste valide au moins jusqu'à ce que le message soit accepté, et ce, même si le message indique que la clé doit être révoquée immédiatement.

9.2.7 Analyse de la sécurité de TAKES

Afin de mieux couvrir les différents aspects de TAKES, nous scindons notre analyse en plusieurs parties. Tout d'abord, nous analysons les diffusions des messages dans TAKES, la gestion et le stockage des clés (paire du clé du nœud, clés récemment apprises, ...) et l'aspect vie privée et anonymat de la solution. Puis nous étudions les attaques sur dépendances de TAKES : les réseaux MANET et les CBID.

9.2.7.1 Diffusion des messages

Dans cette section, nous discutons des protections et des vulnérabilités des messages utilisés dans TAKES. Pour se faire, nous nous plaçons du point de vue de l'attaquant et adoptons le modèle de Dolev-Yao [DY02], c'est à dire, que nous considérons qu'un attaquant est capable d'écouter et modifier n'importe quel message émis ainsi que de les rejouer ou d'en générer de nouveaux, en étant toutefois limité par les protections cryptographiques appliquées au message (il est impossible pour un attaquant de forger une signature ou de décoder le contenu chiffré).

Le message de TAKES (2) émis sur le canal non-authentifié (réseau MANET) lors de la diffusion de la clé (voir message (2) de la Figure 9.5) ne révèle pas d'informations à l'attaquant. L'écoute du trafic ne lui permet d'apprendre que des éléments qui sont publiques. La seule information sensible transmise est la *passphrase* (c.-à-d. *secretA*) associée au HMAC et qui n'est donc pas transmise en clair. Ajouter ou modifier les données du message ne donne pas non plus d'avantages à l'attaquant, puisque cela corromprait la signature du message. De plus, une modification de la clé publique serait détectée par le HMAC. Quelles que soient les modifications effectuées sur le message, ce dernier sera rejetté pour cause de signature invalide ou de HMAC erroné.

L'attaquant peut perturber l'envoi des messages TAKES transmis sur le réseau MANET, ce qui mène à un déni de services. Néanmoins, cette attaque est facilement détectable par les participants, puisque la réception d'un deuxième message sur le canal hors-bande indique que le premier message n'a pas été reçu. Il est également possible pour l'attaquant de rejouer plusieurs fois le message TAKES de diffusé sur le lien. Les

messages reçus sont alors stockés sur le nœud. Toutefois, puisqu'aucun message n'est traité sans action de la part du participant, aucune action cryptographique lourde n'est effectuée. Ainsi, il n'y a pas de consommation inutile des ressources de calcul.

Si l'ordre d'envoi des messages n'est pas respecté, un attaquant peut être amené à connaître la *passphrase* avant que la clé publique ne soit diffusée sur le réseau. Il est ainsi aussi capable de construire un nouveau message, incluant sa clé publique et protégé par un HMAC dont le secret est la *passphrase* qui vient d'être obtenue. C'est pourquoi, il est important de respecter l'ordre des messages dans TAKES. De plus, la *passphrase* est à usage unique. Dans le cas contraire, le participant qui diffuse sa clé s'expose à un attaquant qui pourrait également générer des messages considérés comme valides.

L'OOBC est le point le plus critique dans TAKES puisque celui-ci est totalement dépendant de l'utilisateur. Cela impose que les participants soient capables de prouver leur identité dans le cas où il s'agit d'une première rencontre (par ex. dans le cas d'une réunion entre plusieurs partenaires de travail). Ainsi, le niveau de confiance attribué à une personne permettra de dériver le niveau de confiance que l'on attribue à sa clé publique, son adresse et son équipement.

9.2.7.2 Gestion et stockage des clés

La génération de la clé est un aspect important de TAKES pour définir un niveau minimal de sécurité. Ainsi, nous suivons les recommandations du NIST [Nat07] concernant la taille minimale de clés : celles-ci doivent faire au moins 2048 bits quand l'algorithme RSA est utilisé ou 224 bits quand l'algorithme ECC est utilisé (ou une taille équivalente quand d'autres algorithmes sont utilisés). De plus, aucun autre mécanisme de sécurité ne doit faire un usage direct de la clé. En d'autres termes, cette clé doit seulement servir à dériver de nouveaux secrets.

Puisque les clés publiques sont transmises en clair dans le réseau, elles sont supposés être connues publiquement. Toutefois, un aspect à ne pas négliger dans TAKES est le stockage de la clé sur l'hôte. L'administration des clés est une tâche difficile. D'une part, la pair de clé du nœud ainsi que les clés nouvellement acquises doivent pouvoir être utilisées dans les applications souhaitant faire usage pour leurs mécanismes de sécurité. D'autre part, ce sont ces clés qui permettent d'établir la correspondance entre l'identité de la personne et son adresse. Une corruption de ces clés peut mener à une usurpation d'identité. C'est pourquoi, le système de stockage de clé doit être capable de garantir leur intégrité.

9.2.7.3 Vie privée et anonymat

Sécuriser les communications et fournir l'anonymat sont souvent deux problèmes difficiles dans les réseaux MANET. L'un est souvent obtenu en détriment de l'autre. TAKES fournit la sécurité mais n'offre pas de fonction d'anonymat. En effet, un des intérêts de TAKES est de lier fortement une adresse à l'identité de l'utilisateur du nœud. Toutefois, il est possible d'utiliser plusieurs adresses en parallèle sur une même interface réseau, et de fournir des adresses supplémentaires (non gérées par TAKES)

aux applications nécessitant l'anonymat. Cependant, ces adresses ne bénéficient d'aucune protection, et ne peuvent notamment pas participer au routage sécurisé mis en place à l'aide des clés publiques diffusées par TAKES. Dans ce cas, il faut se tourner vers des solutions comme celle présentée dans le document [KM07] qui utilise la cryptographie symétrique pour offrir un routage sécurisé ainsi que l'anonymat.

9.2.7.4 Attaques sur les dépendances de TAKES

Dans la Section 9.2.7.1, nous avons montré que le message en lui même n'était pas vulnérable. Cependant, TAKES repose fortement sur les réseaux MANET pour transmettre le message de diffusion de clés ainsi que sur les CBID pour sécuriser l'envoi. Toutefois, les réseaux MANET sont vulnérables à de nombreuses attaques qui nuisent à la propagation des messages. Par ailleurs, les CBIDs souffrent également de quelques faiblesses. Dans cette section, nous discutons de l'influence de ces menaces sur la sécurité globale de TAKES.

Attaques sur les MANET

Les MANET sont vulnérables à de nombreux types d'attaques : JellyFish (ralentissement ou réordonnancement du trafic), trou noir [AHK08] (disparition d'une partie du trafic) ou trou de vers [GMW+06] (messages de signalisation recopiés d'une extrémité à l'autre du réseau, perturbant ainsi l'établissement de la topologie au sein de chaque nœud). Ces attaques ont été étudiées par le passé et des parades pour chacune de celles-ci existent. Malheureusement, comme nous ne contrôlons pas le type de réseau MANET, nous devons considérer que celui-ci n'est pas protégé contre ces attaques.

L'attaquant peut alors effectuer les actions suivantes : modifier les routes pour retarder un message ou perturber le routage pour provoquer la perte d'un message. Comme nous l'avons vu Section 9.2.7.1, la perte d'un message est détectable lors de la réception du message transmis sur OOBC. L'ajout d'un délai n'est pas problématique. Quand le délai reste suffisamment court pour que le message OOBC (3) soit reçu après le message sur le lien (2), les deux messages peuvent être traités normalement (puisque les messages sont reçus dans l'ordre). Dans le cas contraire, aucun des deux messages ne doit être traité.

Attaques sur les CBID

Un attaquant souhaitant prendre le contrôle d'un CBID peut procéder de deux façons. Soit il obtient la clé privée associée au CBID (par ex. en cassant l'algorithme de chiffrement). Soit il calcule une nouvelle clé qui permet une collision avec le CBID actuel. Ces deux actions sont très difficiles à réaliser :

– la difficulté pour obtenir la clé privée dépend de l'algorithme cryptographique utilisé. Dans TAKES, nous utilisons les algorithmes RSA ou ECC qui, à ce jour, n'ont pas de faiblesses connues. L'attaquant n'a donc pas d'autres choix que d'essayer des techniques de factorisation (RSA), de résoudre le problème des logarithmes discrets sur les courbes elliptiques (ECC) ou d'effectuer une recherche exhaustive

de la paire de clés. Le choix d'une taille de clés raisonnable (2048 bits pour les clés RSA ou de 224 bits pour les clés ECC ; voir Section 9.2.7.2) rend ce genre d'attaques très difficilement réalisable ;

– l'alternative est d'essayer de produire une collision sur le CBID. Nous considérons ici que la taille de la sortie de la fonction de hachage, et donc du CBID, est n. Si la fonction de hachage est robuste aux attaques, il faut en moyenne 2^{n-1} essais en faisant varier l'entrée de la fonction de hachage avant de trouver une collision avec le CBID. Il n'est toutefois pas possible de faire varier n'importe quelle partie des données fournies en entrée à la fonction de hachage. En effet, le CBID que nous utilisons est dérivé à partir du préfixe sous-réseau, du nom de l'équipement et de la clé publique du nœud. Cependant, le préfixe réseau est utilisé durant la vérification du CBID, et doit donc rester fixe. Il en est de même pour le nom de l'équipement qui est une chaîne de caractères et qui doit donc être lisible. Ainsi, l'attaquant ne peut faire varier que la clé publique, ce qui impose le calcul d'une nouvelle paire de clés à chaque essai. Si nous appliquons ces résultats à un CBID où la sortie de la fonction de hachage fait 59 bits, il faut 2.8×10^{17} essais en moyenne avant de trouver une paire de clés valide. Même si nous considérons un attaquant puissant qui peut générer 100 millions de paire de clés et de condensâts par secondes, il lui faut plus de 9 milliards d'années.

Nous constatons que ces attaques sont purement théoriques et n'influencent pas en pratique le niveau de sécurité de TAKES. Le document [MC04] contient également un grand nombre d'exemples décrivant la (faible) faisabilité des attaques sur les CBID.

9.2.8 Vérification formelle des messages de TAKES

Lors de la vérification formelle de TAKES, nous avons souhaité conserver le sens original des messages et prendre en compte le concept abstrait que sont les canaux hors-bande. La modélisation de ce canal n'est pas prévue par les outils de preuve formelle automatique comme Avispa [Vig06]. Afin de modéliser ce canal, nous nous sommes orientés vers la logique BAN [BAN90] (de Burrows, Abadi et Needham), connue pour son extensibilité. Cette logique permet de modéliser les protocoles de sécurité à travers différents objets distincts : les participants, les aléas, les clés et les postulats. Les notations que nous utilisons pour la preuve de TAKES sont fournies dans le Tableau 9.1.

Nous procédons à la vérification du message de diffusion de clés ainsi que du message de mise à jour/révocation de la clé.

Authentification et diffusion de la clé

Afin de pouvoir appliquer les différents postulats de la logique BAN, nous devons transcrire les messages d'authentification et de diffusion de clés TAKES dans le formalisme de la logique BAN. La Figure 9.7 illustre les messages initiaux légèrement simplifiés et adaptés aux notations du Tableau 9.1.

Le message transmis sur le canal lien inclut @A, Na et Ea. En d'autres termes, il transporte l'identité électronique du participant A. Dans la logique BAN, nous

TAKES	Logique BAN	Signification réelle
hostA	A	Entité A (participant)
nameA	Na	Nom de l'utilisateur A
equipA	Ea	Nom de l'équipement de l'utilisateur A
pkA	Ka	Clé publique de l'utilisateur A
pkA'	Ka'	Nouvelle clé publique de l'utilisateur A
pkA^{-1}	Ka^{-1}	Clé privée de l'utilisateur A
timestampA	Ta	Horodatage émis par A
secretA	S	*passphrase* de l'utilisateur A
validityPkA	Va	Date jusqu'à laquelle Ka/Ka^{-1} sont valides

TABLEAU 9.1: Correspondances entre les notations de TAKES et de la logique BAN

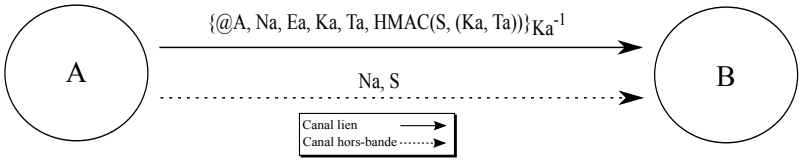


FIGURE 9.7: Messages initiaux de TAKES adaptés aux notations du Tableau 9.1

considérons l'ensemble de ces éléments comme un seul élément que nous nommons Xa. Par ailleurs, le *HMAC(S,(Ka,Ta))* permet d'authentifier les éléments Ka et Ta. Toutefois, cette vérification nécessite la connaissance des éléments Ka, Ta et S (qui sont ici contenus dans le même message). Nous modélisons ce HMAC en logique BAN en utilisant la notation $\langle \overset{Ka}{\mapsto} A, Ta \rangle_S$ qui signifie que (Ka,Ta) est protégé par S.

Sur le canal hors-bande, les messages transmis sont tous considérés comme des messages protégés par le participant qui les émet. Ainsi, faire confiance à l'identité du participant (par ex. A) suffit pour faire confiance à tous les messages émis par ce participant. Nous modélisons cette preuve d'origine dans la logique BAN en associant l'identité du participant IDa à une clé privée K^{-1}_{IDa}. Ainsi, chaque message émis par le participant est *signé* par sa clé privée K^{-1}_{IDa}. La clé publique correspondante, $\overset{K_{IDa}}{\mapsto}$, est connue de tous les autres participants qui font confiance à l'identité IDa.

Nous pouvons maintenant écrire les deux messages de TAKES à partir des éléments de la logique BAN :

- Sur le réseau MANET (c.-à-d. sur le lien) : A $\longrightarrow B$: $\{Xa,\ \overset{Ka}{\mapsto} A,\ Ta,\ \langle \overset{Ka}{\mapsto} A,\ Ta \rangle_S\}_{Ka^{-1}}$
- Sur le canal hors-bande : A $\longrightarrow B$: $\{A \overset{S}{\leftrightharpoons} B\}_{K^{-1}_{IDa}}$

Nous devons également définir les hypothèses de départ suivantes :

1. $B \models \overset{K_{IDa}}{\mapsto} IDa$: B *croit* la *clé publique* K_{IDa}. Cela signifie que, si B fait confiance

à l'identité du participant A, il fait explicitement confiance à la clé publique abstraite qui correspond à l'identité du participant A (IDa) sur le canal hors-bande ;

2. $B \models A \overset{Ka}{\mapsto} A$: B *croit* que A a *autorité* sur la clé publique de A (Ka) ;
3. $B \models \sharp(Ta)$: B *croit* que l'horodatage Ta est *nouveau* (ou récent). Cette hypothèse est valide puisque les horloges sont synchronisées (voir Section 9.2.5) ;
4. $B \models \sharp(A \overset{S}{=\!=} B)$: B *croit* que le secret partagé S est *nouveau* (première utilisation). Dans TAKES, la *passphrase* doit être à usage unique par définition (c.-à-d. elle ne doit pas être réutilisée pour une autre exécution du protocole) ;
5. $B \models A \Rightarrow Xa$: B *croit* que A a *autorité* sur l'élément Xa, cela signifie que B croit que A est le possesseur de l'élément Xa ;
6. $B \models IDa \Rightarrow A \overset{S}{=\!=} B$: B *croit* que l'identité de A (IDa) a *autorité* sur le secret partagé entre A et B (A $\overset{S}{=\!=}$ B).

Le Tableau 9.2 décrit les étapes de la vérification du protocole. Le message émis sur le lien est considéré comme déjà reçu et l'analyse commence à partir du message reçu par le canal hors-bande. Notons que l'aspect CBID n'est pas couvert par la logique BAN.

A $\longrightarrow B : \{A \overset{S}{=\!=} B\}_{K_{IDa}^{-1}}$
7. $B \triangleleft \{A \overset{S}{=\!=} B\}_{IDa^{-1}}$
8. $B \models IDa \mid\sim A \overset{S}{=\!=} B$ // (1), règle "*message meaning*"
9. $B \models IDa \models A \overset{S}{=\!=} B$ // (4)
10. $B \models A \overset{S}{=\!=} B$ // (6), règle "*jurisdiction*" (autorité)
A $\longrightarrow B : \{Xa, \overset{Ka}{\mapsto} A,\ Ta,\ \langle \overset{Ka}{\mapsto} A,\ Ta\rangle_S\}_{Ka^{-1}}$
11. $B \triangleleft \{Xa, \overset{Ka}{\mapsto} A,\ Ta,\ \langle \overset{Ka}{\mapsto} A,\ Ta\rangle_S\}_{Ka^{-1}}$
12. $B \triangleleft \langle \overset{Ka}{\mapsto} A,\ Ta\rangle_S$ // (11)
13. $B \models A \mid\sim (\overset{Ka}{\mapsto} A, Ta)$ // (10), (12)
14. $B \models \sharp(\overset{Ka}{\mapsto} A, Ta)$ // (3), règle "*freshness*" (nouveauté)
15. $B \models \overset{Ka}{\mapsto} A$ // (2), (13), (14)
16. $B \models A \mid\sim (Xa, \overset{Ka}{\mapsto} A,\ Ta,\ \langle \overset{Ka}{\mapsto} A,\ Ta\rangle_S)$ //(11), (15)
17. $B \models A \mid\sim (Xa, \overset{Ka}{\mapsto} A,\ Ta)$ // (16), règle "*once-said*"
18. $B \models \sharp(Xa, \overset{Ka}{\mapsto} A,\ Ta)$ // (3), (17)
19. $B \models A \models (Xa, \overset{Ka}{\mapsto} A,\ Ta)$ // (18), (19)
20. $B \models Xa$ // (5), (19), règle "belief" (croyance)

TABLEAU 9.2: Étapes de la vérification du protocole de diffusion

Les résultats du Tableau 9.2 prouvent qu'à la fin de l'exécution de notre protocole, le participant B croit que Xa est vrai. C'est à dire que chaque élément composant Xa (@A, Na et Ea) est vrai et donc que l'identité électronique de A est bien réelle.

Message de mise à jour/révocation de la clé

Comme nous l'avons vu dans la Section 9.2.6, les actions de mise à jour et de révocation de la clé sont toutes les deux effectuées par le même message. Il s'agit pour nous de valider formellement ce message et ainsi prouver qu'il est sécurisé. La Figure 9.8 illustre ce message adapté aux notations de la logique BAN contenues dans le Tableau 9.1.

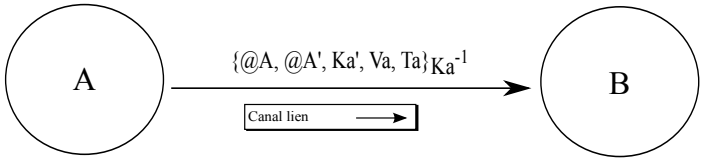


FIGURE 9.8: Message de mise à jour/révocation de la clé adapté aux notations du Tableau 9.1

Afin de diminuer la taille de nos postulats durant la vérification, nous définissons un élément Xa qui comprend (@A, @A', Ka', Va). Nous pouvons ainsi écrire le message de mise à jour/révocation avec le formalisme de la logique BAN de la manière suivante : A $\longrightarrow B : \{Xa,\ Ta\}_{Ka^{-1}}$

Pendant l'envoi de ce message, les hypothèses suivantes s'appliquent :

1. $B \models \overset{K_a}{\mapsto} A$: B *croit* la *clé publique* Ka. Cette hypothèse a été prouvée dans la section précédente ;
2. $B \models \sharp(Ta)$: B *croit* que l'horodatage Ta est *nouveau* (récent). Cette hypothèse est valide puisque les horloges sont synchronisées ;
3. $B \models A \Rightarrow Xa$: B *croit* que A a *autorité* sur l'élément Xa, c'est à dire que B croit que A est le possesseur de l'élément Xa.

A $\longrightarrow B : \{Xa,\ Ta\}_{Ka^{-1}}$
4. $B \triangleleft \{Xa,\ Ta\}_{Ka^{-1}}$
5. $B \models A \mid\sim (Xa, Ta)$ // (1), (4), règle "*message-meaning*"
6. $B \models \sharp(Xa, Ta)$ // (2), règle "*freshness*" (nouveauté)
7. $B \models A \models (Xa, Ta)$ // (5), (6), règle "*nonce-verification*"
8. $B \models Xa$ // (3), (7), opérateur de croyance (*belief operator*)

TABLEAU 9.3: Vérification logique des étapes de mise à jour et révocation de la clé

La vérification formelle du message est donnée dans le Tableau 9.3. La conclusion de la vérification est que B fait confiance à Xa. C'est à dire, le participant B fait confiance à l'adresse @A, l'adresse @A', la nouvelle clé publique Ka' et la date de validité Va.

9.2.9 Implémentation de TAKES

Afin d'implémenter le protocole TAKES dans un environnement réel, nous avons appliqué le concept de CBID aux adresses IPv6 et avons décidé d'utiliser les adresses CGA. Le protocole TAKES a été implémenté en Python et l'interface graphique a été réalisée avec GKTBuilder. Cette implémentation est actuellement restreinte au système d'exploitation Linux du fait de l'utilisation de NDprotector pour la génération et la vérification des adresses CGA. Elle a été réalisée sous notre responsabilité par Andrei Vlad Şambra durant un stage effectué à Télécom SudParis. Elle est publiquement disponible sur une page dédiée [3]. Par faute de temps, seuls les messages de diffusion de la clé publique ont été implémentés. Ceux-ci sont transportés par des messages UDP.

Cette implémentation a été testée dans un environnement MANET avec trois ordinateurs. Comme décrit sur la Figure 9.9, les hôtes sont interconnectés par des liens hétérogènes (lien Ethernet et lien sans-fil). Chaque hôte est configuré pour exécuter TAKES sur son interface MANET. Nous avons retenue l'implémentation B.A.T.M.A.N. [4] qui est intégrée par défaut dans les versions récentes du noyau Linux. Cette implémentation établit un réseau ad-hoc en utilisant un routage proactif et une approche de construction de route stigmergique [PB04]. Nos tests préliminaires ont montrés que chaque hôte pouvait diffuser un message qui était reçu par tous les autres hôtes. De plus, nous avons vérifié que les messages unicast envoyés depuis l'hôte A vers l'hôte C transitaient bien par l'hôte B.

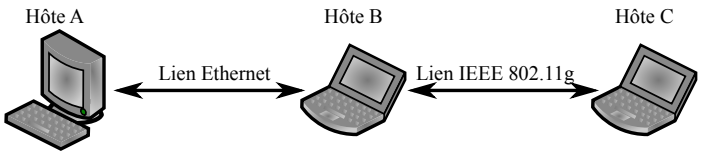


FIGURE 9.9: Topologie du banc d'essai

L'envoi d'un message se déroule comme nous l'avons décrit dans la Section 9.2.5 : après avoir configuré ses paramètres publiques (clé publique, nom d'utilisateur, nom de l'équipement), l'utilisateur est capable de transmettre sa clé publique aux autres participants du réseau MANET. Afin d'envoyer sa clé publique à travers le lien, l'utilisateur doit entrer la *passphrase* à usage unique à travers la boite de dialogue présentée dans la Figure 9.10.

Dès que les participants reçoivent le message TAKES, une notification est présentée à l'écran (voir Figure 9.11). Plusieurs messages peuvent être affichés simultanément tant qu'ils n'ont pas été acceptés. Un double-clic sur l'un de ces messages permet à l'utilisateur de consulter des informations sur le message (adresse IPv6 source, clé publique, etc), puis d'entrer la *passphrase* afin de pouvoir sauvegarder la clé publique associée.

3. http://gitorious.org/takes
4. A Better Approach To Mobile Ad-hoc Networking - http://www.open-mesh.org.

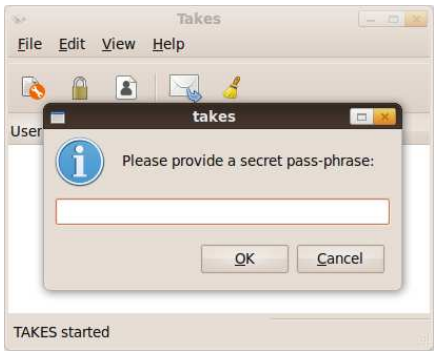


FIGURE 9.10: *Popup* indiquant à l'utilisateur d'entrer sa *passphrase* avant la diffusion de sa clé publique

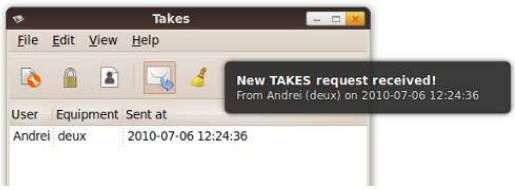


FIGURE 9.11: Notification de l'arrivée d'un message TAKES

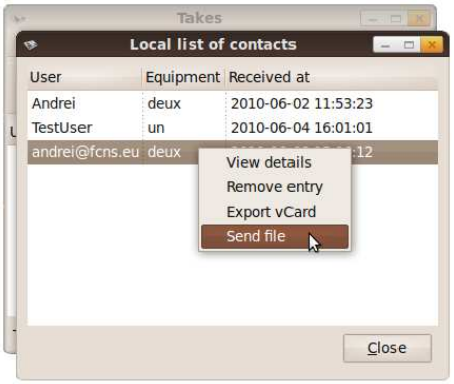


FIGURE 9.12: Interface pour l'envoi de fichiers vers un utilisateur de confiance (dont la clé publique a été avalisée par TAKES)

Nous avons également développé une des applications du scénario II présenté dans la Section 9.2.4 : le transfert de fichiers. Nous supposons que les deux utilisateurs ont préalablement échangé leurs clés publiques via TAKES. Cette application est très simple et opère sur le modèle du *File Transfer Protocol* (FTP) : c.-à-d. elle sépare les messages de contrôle des messages de données. Comme illustré dans la Figure 9.12, lorsqu'un participant souhaite envoyer un fichier à l'un de ses correspondants, il le sélectionne depuis la liste des clés publiques acceptées. La base de données de TAKES permet de retrouver l'adresse IPv6 du correspondant et de se passer totalement d'un service de correspondances nom/adresse (qui peut ne pas exister du tout dans un réseau MANET).

Les services mis en place pour sécuriser le transfert de fichiers sont l'authentification, l'intégrité et la confidentialité du fichier. À la réception, après la vérification de l'adresse CGA source et de la signature des messages, une boite de dialogue permet à la personne destinataire d'accepter ou non le fichier reçu.

9.3 Synthèse du chapitre

Ce chapitre présente les mécanismes basés sur les CGA et les CBIDs pour assurer la sécurité des communications à plusieurs sauts. Dans une première partie, nous faisons un état de l'art des différentes solutions existantes. Dans une seconde partie, nous proposons un nouveau mécanisme, appelé TAKES, qui permet la distribution de clés publiques à plusieurs sauts. Cette proposition a été implémentée et les messages utilisés ont été vérifiés formellement grâce à la logique BAN. La solution permet l'usage des clés publiques pour protéger les applications, même quand aucune infrastructure n'est disponible. Cette solution permet également d'associer un nom à une adresse IPv6, ce qui offre l'avantage de remplacer le service de noms comme DNS dans un environnement sans infrastructure.

10 Conclusion et perspectives de recherche

Dans les années à venir, les adresses CGA et le protocole SEND sont amenés à être déployés pour sécuriser l'accès au lien. Cependant, le protocole SEND, comme la majorité des protocoles IPv6, souffre d'un défaut d'attention de la part de la communauté réseau. Dans cette thèse, nous avons essayé de définir les limites d'utilisation des CGA et du protocole SEND et d'étendre leurs usages à de nouveaux scénarios.

Le premier problème que nous avons souhaité régler est le problème de l'utilisation des adresses CGA et du protocole SEND dans les environnements à faible capacité de calculs et dans les environnements mobiles. Ce problème est multiple, d'un côté les opérations cryptographiques réalisées dans SEND ne sont pas adaptées aux environnements à faible capacité de calculs, de l'autre, SEND doit être adapté pour permettre le partage de l'adresse requis par les protocoles de Mobilité. Plusieurs questions peuvent alors se poser : existe-t-il des algorithmes cryptographiques adaptés capables de jouer le rôle de RSA dans les adresses CGA et le protocole SEND ? Est-il possible d'étendre le protocole SEND pour qu'il devienne indépendant du type d'algorithme utilisé et donc facilement extensible dans le futur à de nouveaux algorithmes cryptographiques plus légers ? Quelles sont les modifications à appliquer au protocole SEND et aux adresses CGA pour permettre le partage d'une adresse entre plusieurs nœuds ?

Le second problème que nous avons identifié est la réutilisation des adresses CGA pour les applications à plusieurs sauts. En effet, si nous avons besoin des adresses CGA pour protéger les applications à un seul saut, comme dans le protocole SEND pour assurer la protection d'une adresse au niveau du lien, ne pouvons nous pas réutiliser les propriétés de sécurité de ces adresses pour définir de nouveaux mécanismes de sécurité ? Si oui, quel type de mécanisme est-il possible de définir ?

Notre première contribution est l'identification d'une attaque par rejeu de messages dans le protocole SEND [CC08]. Cette attaque empêche tous les nœuds présents sur un lien attaqué d'obtenir une nouvelle adresse. Ainsi, un nouveau nœud joignant le lien ne pourra obtenir d'adresse et ne pourra alors pas communiquer. Nous avons reporté cette attaque au groupe de travail chargé de la maintenance des CGA et de SEND à l'IETF et avons également proposé les modifications nécessaires au protocole SEND pour contrer cette attaque.

Notre deuxième contribution est une étude comparative des performances des adresses CGA et du protocole SEND [CBL10, BCLM08]. Nous avons testé de multiples vecteurs pour évaluer les performances de SEND : remplacer l'algorithme RSA par défaut dans le protocole SEND par la cryptographie basée sur les courbes elliptiques, remplacer la fonction de hachage SHA-1 utilisée dans les adresses CGA et dans le protocole SEND par de nouvelles fonctions de hachage (SHA-256, TIGER, etc.) et

l'utilisation des ressources des cartes graphiques présentes sur la majorité des systèmes pour améliorer la vitesse de calcul de l'adresse CGA. Nous avons ainsi conclu que l'utilisation des courbes elliptiques permettait aux nœuds à faible capacité de calcul d'utiliser le protocole SEND. Par ailleurs, nous avons constaté que la transition de la fonction de hachage SHA-1, rendue nécessaire à cause de sa récente cryptanalyse, vers une autre fonction de hachage n'aura pas d'influence importante sur les performances. Finalement, la présence de cartes graphiques de plus en plus puissantes couplée à la généralisation des techniques de GPGPU s'est révélée être un élément à privilégier pour réduire significativement les temps de calcul lors de la génération des adresses CGA.

Notre troisième contribution est une modification des adresses CGA et du protocole SEND pour permettre l'utilisation de nouveaux algorithmes cryptographiques. Nous y définissons notre solution nommée *Signature Algorithm Agility* [CBLM09]. Celle-ci permet d'utiliser de nouveaux algorithmes dans SEND, ce qui améliore ses performances, sa robustesse et sa flexibilité. Par ailleurs, certains gouvernements disposent de leur propre suite de protocoles cryptographiques (comme GOST pour la Russie). Dans ce contexte, notre proposition permettrait la définition et l'ajout rapide de ses nouveaux algorithmes au sein de SEND et faciliterait l'interopérabilité entre les différents types de nœuds. Ce travail a également donné naissance à une série de documents au sein de l'IETF [CLSV10b, CLSV10a, CLMSV09]. Malgré certains retours positifs, ces travaux ne sont malheureusement restés qu'à l'état de drafts au sein du groupe de travail CSI. Cependant, nous pensons que nos documents pourront être utilisés comme base par d'autres personnes dans le futur.

Notre quatrième contribution est une nouvelle implémentation des adresses CGA et du protocole SEND [NDp]. Cette implémentation nous a permis d'évaluer la facilité de la mise en place de notre proposition de *Signature Algorithm Agility* et de vérifier l'interopérabilité des nœuds intégrant la *Signature Algorithm Agility* et des nœuds intégrants seulement le protocole SEND. De plus, cette implémentation nous a permis d'évaluer la conformité des implémentations de NTT DoCoMo et de Cisco vis à vis des spécifications du protocole SEND. À cet effet, nous avons pu reporter à Cisco certains problèmes que nous avions pu constater. Par ailleurs, ce travail d'implémentation nous a permis d'appuyer notre proposition sur la *Signature Algorithm Agility* soumises au sein de l'IETF.

Notre cinquième contribution décrit une utilisation dérivée de la *Signature Algorithm Agility* afin d'améliorer la compatibilité du protocole SEND avec les protocoles de Mobilité et les adresses Anycast [CL10]. Nous étendons le protocole SEND et les adresses CGA afin de pouvoir partager une adresse entre plusieurs nœuds. Cela permet une utilisation conjointe du protocole SEND avec les protocoles *Mobile IPv6* et *Proxy Mobile IPv6*. Nous avons clairement identifié les limites de notre solution et avons conclu que celle-ci était particulièrement adaptée quand le nombre de nœuds à partager une adresse restait faible (moins de 10 nœuds). Une implémentation de cette solution dans NDprotector nous a permis d'obtenir des résultats qui confirment que notre solution est pertinente.

Notre sixième contribution décrit deux mécanismes de génération d'adresses offrant l'anonymat à la couche 2 et à la couche 3 nommés *CGA Éphémères Anonymes* et

CGA Éphémères Pseudo-Anonymes. L'usage des adresses CGA dans ce contexte nous permet d'obtenir des adresses aléatoires pour chaque nouvelle connexion établie par le nœud. L'originalité de cette contribution est de protéger chaque adresse contre les attaques sur le lien (grâce à l'utilisation de SEND) tout en rendant difficile l'association des différentes connexions du nœud et donc en empêchant un attaquant de lier les différentes activités du nœud. Un des intérêts de notre contribution est d'avoir pu mesurer concrètement l'impact de notre proposition *CGA Éphémères Pseudo-Anonymes* sur les performances du nœud : le délai ajouté est à peine perceptible dans la majorité des cas.

Notre septième et dernière contribution décrit un mécanisme de distribution de clés publiques dans les réseaux MANET déconnectés nommé TAKES. La sécurité de ce mécanisme repose sur les utilisateurs pour effectuer des opérations simples. Puisque l'utilisateur joue ici un rôle central dans la diffusion de la clé, les opérations qu'il doit réaliser restent simples. Ainsi, la diffusion d'une clé publique à travers un réseau ne nécessite qu'une seule action pour chacun des participants. Par la suite, nous décrivons de nombreux scénarios d'application pour notre solution et indiquons quels usages de la clé publique peuvent être effectués pour sécuriser les communications entre les participants. Les messages définis dans cette proposition ont été validés formellement grâce à la logique BAN.

Perspectives d'évolutions et nouveaux axes de recherche

Lors de notre étude des performances des adresses CGA et du protocole SEND, nous avions volontairement restreint nos choix à l'utilisation des courbes elliptiques recommandées par le NIST. Toutefois, certaines catégories de courbes elliptiques, qui nécessitent un nombre moins important d'opérations complexes durant les calculs, ont récemment reçu plus d'attention de la part de la communauté cryptographique (courbes de Montgomery, Edwards, etc.). Il serait intéressant de ré-évaluer les performances des CGA et de SEND dans ces nouvelles conditions. Par ailleurs, les techniques GPGPU ont également évolué et nous avions évalué lors de l'écriture de notre article que l'utilisation de nouvelles techniques pourrait encore gagner en efficacité (réductions des temps de calculs) par rapport aux techniques de GPGPU actuelles. Il serait intéressant de vérifier le gain réel qui peut être obtenu avec les cartes de moyenne gamme actuelles.

Nous devons également compléter notre contribution sur la génération d'adresses permettant l'anonymat sur les réseaux IPv6 avant d'essayer de la faire publier. En effet, nous avons seulement implémenté la proposition sur les *CGA Éphémères Pseudo-Anonymes* et il nous reste à implémenter les *CGA Éphémères Anonymes*, beaucoup plus intrusives sur le fonctionnement du nœud, afin de vérifier leur influence sur les performances des connexions (notamment sur les délais d'envoi et de réception de paquets).

Nous avons validé les messages de TAKES grâce aux outils de la logique BAN. Toutefois, ces outils ne permettent pas de modéliser toutes les capacités d'un attaquant et certaines attaques peuvent ne pas être détectées. Nous pensons qu'il pourrait être

judicieux d'effectuer une nouvelle validation du protocole grâce à de nouveaux outils de validation de protocoles (SPAN, Proverif, etc.). Nous pensons qu'il est également possible d'étendre TAKES vers de nouveaux usages et ainsi de le positionner comme une infrastructure décentralisée de sécurité dans les réseaux MANET déconnectés.

Finalement, nous prévoyons d'étendre notre implémentation NDprotector afin qu'elle soit utilisable par une communauté plus large d'utilisateurs.

Influence de nos recherches sur la communauté

Parmi nos différentes contributions, nous avons présenté des solutions pour rendre le protocole SEND compatible avec les nœuds à faible capacité de calculs et une nouvelle implémentation de SEND orientée vers la facilité d'utilisation. Nous pensons que ces deux contributions vont permettre dans le futur un déploiement plus vaste des CGA et du protocole SEND dans les réseaux IPv6.

Après l'analyse de son très bon comportement, il ne fait pas de doute que l'algorithme cryptographique ECC sera à terme intégré au protocole SEND.

Table des figures

Liste des tableaux

Liste des contributions et des publications

Conférences

[CL10] Cheneau, T. et M. Laurent: *Étude des solutions de proxy Neighbor Discovery sécurisées et proposition basée sur la Signature AgilityÉtude des solutions de proxy Neighbor Discovery sécurisées et proposition basée sur la Signature Agility*. Dans *Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI) 2010*, mai 2010. https://amnesiak.org/me/papers/article-SAR-SSI-2010.pdf.

[CBLM09] Cheneau, T., A. Boudguiga et M. Laurent-Maknavicius: *Amélioration des performances des adresses CGA et du protocole SEND : étude comparée de RSA et d'ECC/ECDSAAmélioration des performances des adresses CGA et du protocole SEND : étude comparée de RSA et d'ECC/ECDSA*. Dans *Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI) 2009*, juin 2009. https://amnesiak.org/me/papers/article-SAR-SSI-2009.pdf.

[CC08] Cheneau, T. et J M Combes: *Une attaque par rejeu sur le protocole SENDUne attaque par rejeu sur le protocole SEND*. Dans *Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI) 2008*, octobre 2008. https://amnesiak.org/me/papers/article-SAR-SSI-2008.pdf.

Journaux internationaux

[CBL10] Cheneau, T., A. Boudguiga et M. Laurent: *Significantly improved performances of the cryptographically generated addresses thanks to ECC and GPGPUSignificantly improved performances of the cryptographically generated addresses thanks to ECC and GPGPU*. Computers & Security, 29(4) :419 – 431, 2010, ISSN 0167-4048. http://www.sciencedirect.com/science/article/B6V8G-4Y0D62K-2/2/6703743d0756cc39b4bcd383a3e6d5ed.

Activités de normalisation

[CLMSV09] Cheneau, T., M. Laurent-Maknavicius, S. Shen et M. Vanderveen: *Support for Multiple Signature Algorithms in Cryptographically Generated Addresses (CGAs)Support for Multiple Signature Algorithms in Cryptographically*

Generated Addresses (CGAs). Internet-draftInternet-Draft draft-cheneau-cga-pk-agility-00, Internet Engineering Task Force, juin 2009. http://www.ietf.org/internet-drafts/draft-cheneau-cga-pk-agility-00.txt, Work in progress.

[CLSV10b] Cheneau, T., M. Laurent, S. Shen et M. Vanderveen: *Signature Algorithm Agility in the Secure Neighbor Discovery (SEND)ProtocolSignature Algorithm Agility in the Secure Neighbor Discovery (SEND)Protocol*. Internet-draftInternet-Draft draft-cheneau-csi-send-sig-agility-02, Internet Engineering Task Force, juin 2010. http://www.ietf.org/internet-drafts/draft-cheneau-csi-send-sig-agility-02.txt, Work in progress.

[CLSV10a] Cheneau, T., M. Laurent, S. Shen et M. Vanderveen: *ECC public key and signature support in Cryptographically GeneratedAddresses (CGA) and in the Secure Neighbor Discovery (SEND)ECC public key and signature support in Cryptographically GeneratedAddresses (CGA) and in the Secure Neighbor Discovery (SEND)*. Internet-draftInternet-Draft draft-cheneau-csi-ecc-sig-agility-02, Internet Engineering Task Force, juin 2010. http://www.ietf.org/internet-drafts/draft-cheneau-csi-ecc-sig-agility-02.txt, Work in progress.

Rapports de recherche

[BCLM08] Boudguiga, A., T. Cheneau et M. Laurent-Maknavicius: *Usage and Performance of Cryptographically Generated Addresses, Research report TELECOM and Management SudParis, 08-014 LORUsage and Performance of Cryptographically Generated Addresses, Research report TELECOM and Management SudParis, 08-014 LOR*, octobre 2008.

Bibliographie

[AB96] Anderson, R. et E. Biham: *Tiger : a fast new hash function*. Dans *Fast Software Encryption, Third International Workshop Proceedings*, pages 89–97. Springer-Verlag, 1996.

[ABB+04] Aiello, W., S. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. Keromytis et O. Reingold: *Just Fast Keying : Key Agreement in a Hostile Internet*. ACM Trans. Inf. Syst. Secur, 7, 2004.

[AHK08] Aad, I., J. P. Hubaux et E.W. Knightly: *Impact of Denial of Service Attacks on Ad Hoc Networks*. Transactions on Networking, IEEE/ACM, 16(4) :791–802, 2008, ISSN 1063-6692.

[AKZN05] Arkko, J., J. Kempf, B. Zill et P. Nikander: *SEcure Neighbor Discovery (SEND)*. RFC 3971, Internet Engineering Task Force, mars 2005. http://www.rfc-editor.org/rfc/rfc3971.txt.

[ANN07] Arkko, J., P. Nikander et M. Näslund: *Enhancing Privacy with Shared Pseudo Random Sequences*. Dans Christianson, Bruce, Bruno Crispo, James Malcolm et Michael Roe (rédacteurs) : *Security Protocols*, tome 4631 de *Lecture Notes in Computer Science*, pages 187–196. Springer Berlin / Heidelberg, 2007. http://dx.doi.org/10.1007/978-3-540-77156-2_22.

[Aur05] Aura, T.: *Cryptographically Generated Addresses (CGA)*. RFC 3972, Internet Engineering Task Force, mars 2005. http://www.rfc-editor.org/rfc/rfc3972.txt.

[BA] Biham, E. et R. Anderson: *Tiger : A Fast New Hash Function website*. http://www.cs.technion.ac.il/~biham/Reports/Tiger/.

[BA06] Bagnulo, M. et J. Arkko: *Cryptographically Generated Addresses (CGA) Extension Field Format*. RFC 4581, Internet Engineering Task Force, octobre 2006. http://www.rfc-editor.org/rfc/rfc4581.txt.

[BA07] Bagnulo, M. et J. Arkko: *Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGAs)*. RFC 4982, Internet Engineering Task Force, juillet 2007. http://www.rfc-editor.org/rfc/rfc4982.txt.

[Bag09] Bagnulo, M.: *Hash-Based Addresses (HBA)*. RFC 5535, Internet Engineering Task Force, juin 2009. http://www.rfc-editor.org/rfc/rfc5535.txt.

[BAN90] Burrows, M., M. Abadi et R. Needham: *A logic of authentication*. ACM Transactions on Computer Systems (TOCS), 8(1) :18–36, 1990.

[BBM+03] Bassi, A., M. Beck, T. Moore, J. Plank, M. Swany, R. Wolski et G. Fagg: *The Internet Backplane Protocol : a study in resource sharing*. Future Generation Computer Systems, 19(4) :551–561, 2003, ISSN 0167-739X. http://www.sciencedirect.com/science/article/B6V06-487V024-1/2/961c4825334d230a6552443383165c12.

[BCK96] Bellare, M., R. Canetti et H. Krawczyk: *Keying hash functions for message authentication*. Dans *Advances in Cryptology—CRYPTO'96*, pages 1–15. Springer, 1996.

[BCLM08] Boudguiga, A., T. Cheneau et M. Laurent-Maknavicius: *Usage and Performance of Cryptographically Generated Addresses, Research report TELECOM and Management SudParis, 08-014 LOR*, octobre 2008.

[Bel96] Bellovin, S.: *Defending Against Sequence Number Attacks*. RFC 1948, Internet Engineering Task Force, mai 1996. http://www.rfc-editor.org/rfc/rfc1948.txt.

[BHK05] Barbeau, M., J. Hall et E. Kranakis: *Detecting Impersonation Attacks in Future Wireless and Mobile Networks*. Dans *In Proceedings of MADNES 2005 - Workshop on Secure Mobile Ad-hoc Networks and Sensors - Held in conjunction with ISC'05*. SVLNCS, 2005.

[BL03] Bassi, A. et J. Laganier: *Towards an IPv6-Based Security Framework for Distributed Storage Resources*. Dans *Communications and Multimedia Security (CMS 2003)*, tome 2828 de *Lecture Notes in Computer Science*, pages 54–64. Springer, 2003, ISBN 3-540-20185-8.

[Bro08] Broadcom: *BCM57XX Programmer's guide : Host Programmer Interface Specification for the NetXtreme Family of Highly Integrated Media Access Controllers*, 2008.

[BSSW02] Balfanz, D., D.K. Smetters, P. Stewart et H.C. Wong: *Talking to strangers : Authentication in ad-hoc wireless networks*. Dans *Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS)*, pages 7–19, 2002.

[CBL10] Cheneau, T., A. Boudguiga et M. Laurent: *Significantly improved performances of the cryptographically generated addresses thanks to ECC and GPGPU*. Computers & Security, 29(4) :419 – 431, 2010, ISSN 0167-4048. http://www.sciencedirect.com/science/article/B6V8G-4Y0D62K-2/2/6703743d0756cc39b4bcd383a3e6d5ed.

[CBLM09] Cheneau, T., A. Boudguiga et M. Laurent-Maknavicius: *Amélioration des performances des adresses CGA et du protocole SEND : étude comparée de RSA et d'ECC/ECDSA*. Dans *Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI) 2009*, juin 2009. https://amnesiak.org/me/papers/article-SAR-SSI-2009.pdf.

[CC08] Cheneau, T. et J M Combes: *Une attaque par rejeu sur le protocole SEND*. Dans *Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI) 2008*, octobre 2008. https://amnesiak.org/me/papers/article-SAR-SSI-2008.pdf.

[CDG06] Conta, A., S. Deering et M. Gupta: *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443, Internet Engineering Task Force, mars 2006. http://www.rfc-editor.org/rfc/rfc4443.txt.

[CG09] Chiu, S. et E. Gamess: *Easy-SEND : A Didactic Implementation of the Secure Neighbor Discovery Protocol for IPv6*. Dans *Proceedings of the World Congress on Engineering and Computer Science*, tome 1, 2009.

[CKD10] Combes, J M., S. Krishnan et G. Daley: *Securing Neighbor Discovery Proxy : Problem Statement*. RFC 5909, Internet Engineering Task Force, juillet 2010. http://www.rfc-editor.org/rfc/rfc5909.txt.

[CL10] Cheneau, T. et M. Laurent: *Étude des solutions de proxy Neighbor Discovery sécurisées et proposition basée sur la Signature Agility*. Dans *Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI) 2010*, mai 2010. https://amnesiak.org/me/papers/article-SAR-SSI-2010.pdf.

[CLMSV09] Cheneau, T., M. Laurent-Maknavicius, S. Shen et M. Vanderveen: *Support for Multiple Signature Algorithms in Cryptographically Generated Addresses (CGAs)*. Internet-Draft draft-cheneau-cga-pk-agility-00, Internet Engineering Task Force, juin 2009. http://www.ietf.org/internet-drafts/draft-cheneau-cga-pk-agility-00.txt, Work in progress.

[CLSV10a] Cheneau, T., M. Laurent, S. Shen et M. Vanderveen: *ECC public key and signature support in Cryptographically GeneratedAddresses (CGA) and in the Secure Neighbor Discovery (SEND)*. Internet-Draft draft-cheneau-csi-ecc-sig-agility-02, Internet Engineering Task Force, juin 2010. http://www.ietf.org/internet-drafts/draft-cheneau-csi-ecc-sig-agility-02.txt, Work in progress.

[CLSV10b] Cheneau, T., M. Laurent, S. Shen et M. Vanderveen: *Signature Algorithm Agility in the Secure Neighbor Discovery (SEND)Protocol*. Internet-Draft draft-cheneau-csi-send-sig-agility-02, Internet Engineering Task Force, juin 2010. http://www.ietf.org/internet-drafts/draft-cheneau-csi-send-sig-agility-02.txt, Work in progress.

[CM02] Castelluccia, C. et G. Montenegro: *IPv6 Opportunistic Encryption, Research Report INRIA, 4568*. octobre 2002.

[CM03] Castelluccia, C. et G. Montenegro: *Securing Group Management in IPv6 with Cryptographically Generated Addresses*. Dans *Proceedings of the Eighth IEEE Symposium on Computers and Communications. ISCC 2003*, 2003.

[Coc07] Cochran, M.: *Notes on the Wang et al.* 2^{63} *SHA-1 Differential Path*. Cryptology ePrint Archive, Report 2007/474, 2007. http://eprint.iacr.org/.

[CSF+08] Cooper, D., S. Santesson, S. Farrell, S. Boeyen, R. Housley et W. Polk: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280, Internet Engineering Task Force, mai 2008. http://www.rfc-editor.org/rfc/rfc5280.txt.

[CUD] *CUDA Zone website*. http://www.nvidia.com/object/cuda_home.html.

[CWH06] Chen, G.N., C.Y. Wang et R.H. Hwang: *MTSP : Multi-hop Time Synchronization Protocol for IEEE 802.11 Wireless Ad Hoc Network*. Dans *Wireless Algorithms, Systems, and Applications*, tome 4138 de *Lecture Notes in Computer Science*, pages 664–675. Springer Berlin / Heidelberg, 2006. http://dx.doi.org/10.1007/11814856_62.

[DBP96] Dobbertin, H., A. Bosselaers et B. Preneel: *RIPEMD-160 : A Strengthened Version of RIPEMD*. Dans *Proceedings of the Third International Workshop on Fast Software Encryption*, pages 71–82, London, UK, 1996. Springer-Verlag, ISBN 3-540-60865-6.

[DBV+03] Droms, R., J. Bound, B. Volz, T. Lemon, C. Perkins et M. Carney: *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*. RFC 3315, Internet Engineering Task Force, juillet 2003. http://www.rfc-editor.org/rfc/rfc3315.txt.

[DFH99] Deering, S., W. Fenner et B. Haberman: *Multicast Listener Discovery (MLD) for IPv6*. RFC 2710, Internet Engineering Task Force, octobre 1999. http://www.rfc-editor.org/rfc/rfc2710.txt.

[DH98] Deering, S. et R. Hinden: *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460, Internet Engineering Task Force, décembre 1998. http://www.rfc-editor.org/rfc/rfc2460.txt.

[DMA04] Deng, H., A. Mukherjee et D.P. Agrawal: *Threshold and identity-based key management and authentication for wireless ad hoc networks*. Dans *International Conference on Information Technology : Coding and Computing (ITCC'04)*, tome 1, pages 107–115, 2004.

[DWPT05] Devarapalli, V., R. Wakikawa, A. Petrescu et P. Thubert: *Network Mobility (NEMO) Basic Support Protocol*. RFC 3963, Internet Engineering Task Force, janvier 2005. http://www.rfc-editor.org/rfc/rfc3963.txt.

[DY02] Dolev, D. et A. Yao: *On the security of public key protocols*. IEEE Transactions on Information Theory, 29(2) :198–208, 2002.

[Eck10] Eckersley, P.: *A Primer on Information Theory and Privacy*, janvier 2010. https://www.eff.org/deeplinks/2010/01/primer-information-theory-and-privacy.

[EFL+99] Ellison, C., B. Frantz, B. Lampson, R. Rivest, B. Thomas et T. Ylonen: *SPKI Certificate Theory*. RFC 2693, Internet Engineering Task Force, septembre 1999. http://www.rfc-editor.org/rfc/rfc2693.txt.

[FAWD02] Feeney, L.M., B. Ahlgren, A. Westerlund et A. Dunkels: *Spontnet : Experiences in configuring and securing small ad hoc networks*. Dans *Proceedings of The Fifth International Workshop on Network Applicances (IWNA5), Liverpool, UK*, 2002.

[FGM+99] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach et T. Berners-Lee: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, Internet Engineering Task Force, juin 1999. http://www.rfc-editor.org/rfc/rfc2616.txt.

[GG05] Gruteser, M. et D. Grunwald: *Enhancing location privacy in wireless LAN through disposable interface identifiers : a quantitative analysis*. Mob. Netw. Appl., 10(3) :315–325, 2005, ISSN 1383-469X.

[GGCS02] Gupta, V., S. Gupta, S. Chang et D. Stebila: *Performance analysis of elliptic curve cryptography for SSL*. Dans *WiSE '02 : Proceedings of the 1st ACM workshop on Wireless security*, pages 87–94, New York, NY, USA, 2002. ACM, ISBN 1-58113-585-8.

[GHM+07] Gill, V., J. Heasley, D. Meyer, P. Savola et C. Pignataro: *The Generalized TTL Security Mechanism (GTSM)*. RFC 5082, Internet Engineering Task Force, octobre 2007. http://www.rfc-editor.org/rfc/rfc5082.txt.

[GKK10] Gagliano, R., S. Krishnan et A. Kukec: *Certificate profile and certificate management for SEND*. Internet-Draft draft-ietf-csi-send-cert-06, Internet Engineering Task Force, août 2010. http://www.ietf.org/internet-drafts/draft-ietf-csi-send-cert-06.txt, Work in progress.

[GLD+08] Gundavelli, S., K. Leung, V. Devarapalli, K. Chowdhury et B. Patil: *Proxy Mobile IPv6*. RFC 5213, Internet Engineering Task Force, août 2008. http://www.rfc-editor.org/rfc/rfc5213.txt.

[GMW+06] Gorlatova, M.A., P.C. Mason, M. Wang, L. Lamont et R. Liscano: *Detecting Wormhole Attacks in Mobile Ad Hoc Networks through Protocol Breaking and Packet Timing Analysis*. Dans *Military Communications Conference, 2006. MILCOM 2006. IEEE*, pages 1–7, 2006.

[HC98] Harkins, D. et D. Carrel: *The Internet Key Exchange (IKE)*. RFC 2409, Internet Engineering Task Force, novembre 1998. http://www.rfc-editor.org/rfc/rfc2409.txt.

[HD06] Hinden, R. et S. Deering: *IP Version 6 Addressing Architecture*. RFC 4291, Internet Engineering Task Force, février 2006. http://www.rfc-editor.org/rfc/rfc4291.txt.

[HH05] Hinden, R. et B. Haberman: *Unique Local IPv6 Unicast Addresses*. RFC 4193, Internet Engineering Task Force, octobre 2005. http://www.rfc-editor.org/rfc/rfc4193.txt.

[HMNS98] Haller, N., C. Metz, P. Nesser et M. Straw: *A One-Time Password System*. RFC 2289, Internet Engineering Task Force, février 1998. http://www.rfc-editor.org/rfc/rfc2289.txt.

[HMV04] Hankerson, D., A. Menezes et S. Vanstone: *Guide to elliptic curve cryptography*. Springer, 2004.

[HN09] Haddad, W. et M. Naslund: *On Secure Neighbor Discovery Proxying Using 'Symbiotic' Relationship*. Internet-Draft draft-haddad-csi-symbiotic-sendproxy-01, Internet Engineering Task Force, juillet 2009. http://www.ietf.org/internet-drafts/draft-haddad-csi-symbiotic-sendproxy-01.txt, Work in progress.

[HPFS02] Housley, R., W. Polk, W. Ford et D. Solo: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 3280, Internet Engineering Task Force, avril 2002. http://www.rfc-editor.org/rfc/rfc3280.txt.

[HT02] Haberman, B. et D. Thaler: *Host-based Anycast using MLD*. Internet-Draft draft-haberman-ipngwg-host-anycast, Internet Engineering Task Force, mai 2002. http://tools.ietf.org/html/draft-haberman-ipngwg-host-anycast-01, Work in progress.

[JPA04] Johnson, D., C. Perkins et J. Arkko: *Mobility Support in IPv6*. RFC 3775, Internet Engineering Task Force, juin 2004. http://www.rfc-editor.org/rfc/rfc3775.txt.

[JPBM07] Jeong, J., S. Park, L. Beloeil et S. Madanapalli: *IPv6 Router Advertisement Option for DNS Configuration*. RFC 5006, Internet Engineering Task Force, septembre 2007. http://www.rfc-editor.org/rfc/rfc5006.txt.

[JX09] Jiang, S. et Z. Xia: *Configuring Cryptographically Generated Addresses (CGA) using DHCPv6*. Internet-Draft draft-jiang-csi-cga-config-dhcpv6-01, Internet Engineering Task Force, octobre 2009. http://www.ietf.org/internet-drafts/draft-jiang-csi-cga-config-dhcpv6-01.txt, Work in progress.

[KAM09a] Kitamura, H., S. Ata et M. Murata: *Harmless IPv6 Address State Extension (Uncertain State)*. Internet-Draft draft-kitamura-ipv6-uncertain-address-state-01, Internet Engineering Task Force, juillet 2009. http://tools.ietf.org/html/draft-kitamura-ipv6-uncertain-address-state-01, Work in progress.

[KAM09b] Kitamura, H., S. Ata et M. Murata: *IPv6 Ephemeral Addresses*. Internet-Draft draft-kitamura-ipv6-ephemeral-address-01, Internet Engineering Task Force, juillet 2009. http://tools.ietf.org/html/draft-kitamura-ipv6-ephemeral-address-01, Work in progress.

[Ken05] Kent, S.: *IP Encapsulating Security Payload (ESP)*. RFC 4303, Internet Engineering Task Force, décembre 2005. http://www.rfc-editor.org/rfc/rfc4303.txt.

[KG05] Kempf, J. et C. Gentry: *Secure IPv6 Address Proxying using Multi-Key Cryptographically Generated Addresses (MCGAs)*. Internet-

Draft draft-kempf-mobopts-ringsig-ndproxy-02, Internet Engineering Task Force, août 2005. http://tools.ietf.org/html/draft-kempf-mobopts-ringsig-ndproxy-02, Work in progress.

[KKJ10] Kukec, A., S. Krishnan et S. Jiang: *SEND Hash Threat Analysis*. Internet-Draft draft-ietf-csi-hash-threat-10, Internet Engineering Task Force, juillet 2010. http://www.ietf.org/internet-drafts/draft-ietf-csi-hash-threat-10.txt, Work in progress.

[KLBGM10] Krishnan, S., J. Laganier, M. Bonola et A. Garcia-Martinez: *Secure Proxy ND Support for SEND*. Internet-Draft draft-ietf-csi-proxy-send-04, Internet Engineering Task Force, mai 2010. http://www.ietf.org/internet-drafts/draft-ietf-csi-proxy-send-04.txt, Work in progress.

[KM07] Kao, J.C. et R. Marculescu: *Real-time anonymous routing for mobile ad hoc networks*. Dans *Wireless Communications and Networking Conference, 2007. WCNC 2007. IEEE*, pages 4139–4144. IEEE, 2007.

[KWRG06] Kempf, J., J. Wood, Z. Ramzan et C. Gentry: *IP Address Authorization for Secure Address Proxying Using Multi-key CGAs and Ring Signatures* . Dans *Advances in Information and Computer Security*, pages 196–211. Springer, 2006.

[LKS04] Lynn, C., S. Kent et K. Seo: *X.509 Extensions for IP Addresses and AS Identifiers*. RFC 3779, Internet Engineering Task Force, juin 2004. http://www.rfc-editor.org/rfc/rfc3779.txt.

[LLM09] Larafa, S. et M. Laurent-Macknavicius: *Protocols for Distributed AAA Framework in Mobile Ad-hoc Networks*. Dans *Proceedings of Workshop on Mobile and Wireless Networks Security, MWNS 2009*, pages 75–36, mai 2009, ISBN 978-3-8322-8177-9. http://www-lor.int-evry.fr/%7Emaknavic/articles/Larafa-MWNS2009.pdf.

[LM08] Laganier, J. et G. Montenegro: *Using IKE with IPv6 Cryptographically Generated Address*. Internet-Draft draft-laganier-ike-ipv6-cga-02, Internet Engineering Task Force, janvier 2008. http://tools.ietf.org/html/draft-laganier-ike-ipv6-cga-02, Work in progress.

[LWLL07] Lacharite, Y., M. Wang, L. Lamont et L. Landmark: *A Simplified Approach to Multicast Forwarding Gateways in MANET*. Dans *Wireless Communication Systems, 2007. ISWCS 2007.*, pages 426–430, oct. 2007.

[MC04] Montenegro, G. et C. Castelluccia: *Crypto-based identifiers (CBIDs) : Concepts and applications*. ACM Trans. Inf. Syst. Secur., 7(1) :97–127, 2004, ISSN 1094-9224.

[Moo06] Moore, N.: *Optimistic Duplicate Address Detection (DAD) for IPv6*. RFC 4429, Internet Engineering Task Force, avril 2006. http://www.rfc-editor.org/rfc/rfc4429.txt.

[MOVR97] Menezes, A. J., P. C. Van Oorschot, S. A. Vanstone et R. L. Rivest: *Handbook of Applied Cryptography*, 1997.

[MW07] Mayrhofer, Rene et Martyn Welch: *A Human-Verifiable Authentication Protocol Using Visible Laser Light*. The Second International Conference on Availability, Reliability and Security (ARES'07), pages 1143–1148, 2007.

[NA02] Nikander, P. et J. Arkko: *Delegation of Signalling Rights*. Dans *Security Protocols Workshop*, pages 203–214, 2002.

[Nat07] National Institute of Standards and Technology: *NIST Special Publication 800-57*. NIST Special Publication, 800 :57, 2007. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf.

[Nat08a] National Institute of Standards and Technology: *Draft FIPS 186-3 : Digital Signature Standard (DSS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, novembre 2008. http://csrc.nist.gov/publications/drafts/fips_186-3/Draft_FIPS-186-320_November2008.pdf.

[Nat08b] National Institute of Standards and Technology: *FIPS 180-3 : Secure Hash Standard (SHS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, octobre 2008. http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf.

[NB09] Nordmark, E. et M. Bagnulo: *Shim6 : Level 3 Multihoming Shim Protocol for IPv6*. RFC 5533, Internet Engineering Task Force, juin 2009. http://www.rfc-editor.org/rfc/rfc5533.txt.

[NDK07] Narten, T., R. Draves et S. Krishnan: *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*. RFC 4941, Internet Engineering Task Force, septembre 2007. http://www.rfc-editor.org/rfc/rfc4941.txt.

[NDp] *Site web du projet NDprotector*. http://amnesiak.org/NDprotector/.

[NDS02] NDSS'02: *Statistically Unique and Cryptographically Verifiable (SUCV) Identifier and Addresses*. The Internet Society, février 2002.

[NIS] *Commentaires du NIST sur la cryptanalyse de SHA-1*. http://csrc.nist.gov/groups/ST/hash/statement.html.

[NKN04] Nikander, P., J. Kempf et E. Nordmark: *IPv6 Neighbor Discovery (ND) Trust Models and Threats*. RFC 3756, Internet Engineering Task Force, mai 2004. http://www.rfc-editor.org/rfc/rfc3756.txt.

[NM02] Nakajima, J. et M. Matsui: *Performance Analysis and Parallel Implementation of Dedicated Hash Functions*. Dans *EUROCRYPT '02 : Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, pages 165–180, London, UK, 2002. Springer-Verlag, ISBN 3-540-43553-0.

[NNSS07] Narten, T., E. Nordmark, W. Simpson et H. Soliman: *Neighbor Discovery for IP version 6 (IPv6)*. RFC 4861, Internet Engineering Task Force, septembre 2007. http://www.rfc-editor.org/rfc/rfc4861.txt.

[PB04] Parunak, H. et S.A. Brueckner: *Stigmergic Learning for Self-Organizing Mobile Ad-Hoc Networks (MANET's)*. Dans *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, tome 3, page 1325. IEEE Computer Society, 2004.

[PH10] Pfitzmann, A. et M. Hansen: *A terminology for talking about privacy by data minimization : Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, août 2010.

[Plu82] Plummer, D.: *Ethernet Address Resolution Protocol : Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. RFC 0826, Internet Engineering Task Force, novembre 1982. http://www.rfc-editor.org/rfc/rfc826.txt.

[Pos80] Postel, J.: *User Datagram Protocol*. RFC 0768, Internet Engineering Task Force, août 1980. http://www.rfc-editor.org/rfc/rfc768.txt.

[Pos81] Postel, J.: *Transmission Control Protocol*. RFC 0793, Internet Engineering Task Force, septembre 1981. http://www.rfc-editor.org/rfc/rfc793.txt.

[PS99] Perrig, A. et D. Song: *Hash visualization : A new technique to improve real-world security*. Dans *International Workshop on Cryptographic Techniques and E-Commerce*, pages 131–138, 1999.

[Res00] Rescorla, E.: *HTTP Over TLS*. RFC 2818, Internet Engineering Task Force, mai 2000. http://www.rfc-editor.org/rfc/rfc2818.txt.

[RR05] Richardson, M. et D.H. Redelmeier: *Opportunistic Encryption using the Internet Key Exchange (IKE)*. RFC 4322, Internet Engineering Task Force, décembre 2005. http://www.rfc-editor.org/rfc/rfc4322.txt.

[RSA02] RSA Laboratories: *PKCS #1 v2.1 : RSA Cryptography Standard*, 2002.

[RST01] Rivest, R. L., A. Shamir et Y. Tauman: *How to Leak a Secret*. Dans *ASIACRYPT '01 : Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information S ecurity*, pages 552–565, London, UK, 2001. Springer-Verlag, ISBN 3-540-42987-5.

[SCHS07] Sheu, J.P., C.M. Chao, W.K. Hu et C.W. Sun: *A clock synchronization algorithm for multihop wireless ad hoc networks*. Wireless Personal Communications, 43(2) :185–200, 2007. http://dx.doi.org/10.1007/s11277-006-9217-4.

[Sta95] Standard, Secure Hash: *Technical Report FIPS PUB 180-1*. US Department of Commerce/National Institute of Standards and Technology, 1995.

[SU08] Saxena, N. et M.D.B. Uddin: *Device Pairing Using Unidirectional Physical Channels*. Dans *Mobile and Wireless Networks Security : Proceedings of the MWNS 2008 Workshop*, page 27, 2008.

[TNJ07] Thomson, S., T. Narten et T. Jinmei: *IPv6 Stateless Address Autoconfiguration*. RFC 4862, Internet Engineering Task Force, septembre 2007. http://www.rfc-editor.org/rfc/rfc4862.txt.

[TTP06] Thaler, D., M. Talwar et C. Patel: *Neighbor Discovery Proxies (ND Proxy)*. RFC 4389, Internet Engineering Task Force, avril 2006. http://www.rfc-editor.org/rfc/rfc4389.txt.

[Vig06] Viganò, Luca: *Automated Security Protocol Analysis With the AVISPA Tool*. Electronic Notes in Theoretical Computer Science, 155 :61–86, 2006, ISSN 1571-0661. Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI).

[whi] *The WHIRLPOOL Hash Function website*. http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html.

[WYY05] Wang, X., Y. Lisa Yin et H. Yu: *Finding Collisions in the Full SHA-1*. Dans *In Proceedings of Crypto*, pages 17–36. Springer, 2005.

[XyS] *XySSL website*. http://xyssl.org.

[YK03] Yi, S. et R. Kravets: *MOCA : Mobile certificate authority for wireless ad hoc networks*. Dans *2nd Annual PKI Research Workshop Pre-Proceedings*, tome 51, page 65, 2003.

[ZNW10] Zhang, D., P. Nallur et M. Wasserman: *Cryptographically Generated Address (CGA) Extension Header for Internet Protocol version 6 (IPv6)*. Internet-Draft draft-dong-savi-cga-header-03, Internet Engineering Task Force, juillet 2010. http://www.ietf.org/internet-drafts/draft-dong-savi-cga-header-03.txt, Work in progress.

[Zug05] Zugenmaier, A.: *FLASCHE–A Mechanism Providing Anonymity for Mobile Users*. Dans *Privacy Enhancing Technologies*, pages 121–141. Springer, 2005.

Glossaire des Acronymes

ARP : Address Resolution Protocol
CGA : Cryptographically Generated Addresses
ECC : Elliptic Curve Cryptography
GPGPU : General-Purpose Computing on Graphical Processing Units
IPv4 : Internet Protocol, version 4
IPv6 : Internet Protocol, version 6
MANET : Mobile Ad-hoc Network
MIPv6 : Mobile IP, version 6
MTU : Maximum Transfert Unit, unité de transfer maximale
NDP : Neighbor Discovery Protocol, protocole de Découverte de Voisins
PMIPv6 : Proxy Mobile IP, version 6
SEND : SEND, protocole de Découverte de Voisins Sécurisée