# Amélioration des performances des adresses CGA et du protocole SEND: étude comparée de RSA et d'ECC/ECDSA

Tony Cheneau (tony.cheneau@it-sudparis.eu)*
Aymen Boudguiga (aymen.boudguiga@it-sudparis.eu)*
Maryline Laurent-Maknavicius
(maryline.maknavicius@it-sudparis.eu)*

**Résumé :** Les adresses générées de manière cryptographique (CGA) servent essentiellement aujourd'hui au protocole de découverte de voisins sécurisée (SEND). Les CGA ne peuvent être générées qu'à l'aide de l'algorithme RSA et la fonction de hachage SHA-1. Pour une meilleure évolutivité des CGA, une robustesse étendue, et des performances améliorées, cet article propose d'intégrer d'autres algorithmes en alternative, et prend l'exemple de la cryptographie sur les courbes elliptiques (ECC). Pour cela, il décrit une adaptation des mécanismes de SEND avec le principe de CGA multi-clés. Une évaluation des performances des CGA entre ECC et RSA démontre des performances améliorées avec ECC dans le mécanisme de génération des CGA et avec ECDSA (Elliptic Curve DSA) pour la génération de signatures. Un gain en performances est conforté dans un environnement mobile et prouve la viabilité de l'utilisation des CGA ECC et de SEND dans ce contexte en tenant compte de nos nouvelles améliorations.

**Mots Clés :** CGA,ECC,ECDSA,RSA,performance

## 1 Introduction

Le protocole de découverte de voisins (Neighbor Discovery Protocol ou ND) [NNSS07] a été développé pour le protocole IPv6 [DH98]. Il permet aux hôtes connectés à un même lien (c.-à-d. dans le même voisinage) de découvrir les routeurs voisins, de résoudre les adresses logiques en adresses physiques, de détecter les nœuds indisponibles et les adresses dupliquées. Il s'est assez rapidement avéré que le protocole ND était vulnérable à de nombreuses attaques, en particulier les dénis de services [NKN04]. Pour se prémunir de ces attaques, un groupe de travail de l'IETF a défini le protocole SEND (Secure Neighbor Discovery) [AKZN05] qui se présente sous la forme de plusieurs extensions à ND. Le protocole SEND se base essentiellement sur l'utilisation des adresses CGA (Cryptographically Generated Addresses) [Aur05]. Une adresse CGA est une adresse IPv6 générée de manière cryptographique et qui sert uniquement à prouver que l'émetteur d'un message SEND est bien propriétaire de son adresse CGA. Les CGA servent en quelque sorte à introduire un mécanisme décentralisé permettant de gérer l'association entre une clé publique et son

*. Institut TELECOM, TELECOM SudParis, CNRS Samovar UMR 5157, 9 rue Charles Fourier, 91011 Évry, France

propriétaire. Le protocole SEND complète ce mécanisme en signant des messages ND à l'aide d'une signature RSA utilisant la clef privée associée à la clef publique de la CGA.

Les CGA ne devraient pas se limiter au seul contexte de SEND. L'IETF prévoit d'autres usages possibles comme les réseaux mobiles, et travaille à adapter les CGA en conséquence. De nouvelles fonctionnalités sont donc en cours de définition, comme le "proxying" du protocole ND [Kem07] dans un contexte MIPv6 [JPA04]. De plus, les CGA ont été proposées en couplage avec les *Hash Based Addresses* [Bag07] (HBA) afin de résoudre le problème de l'ajout d'une nouvelle adresse au sein de réseaux alliant plusieurs fournisseurs de services internet (cas de réseaux multihomés).

L'utilisation de l'algorithme RSA et la fonction de hachage SHA-1 dans la génération des CGA est actuellement imposée par SEND. Notre proposition développée dans cet article concerne l'utilisation des courbes elliptiques (Elliptic Curves Cryptography, ECC) [HMV04]. Cette idée est motivée essentiellement par les gains en performances très encourageants trouvés dans les domaines des réseaux de capteurs et des réseaux ad-hoc. Aussi proposons-nous d'adapter SEND et les CGA afin de rendre possible l'usage des courbes elliptiques.

Dans cet article, nous présentons dans la section 2 les CGA et le protocole SEND. La section 3 détaille les algorithmes de génération et de vérification des CGA. La section 4 expose les mécanismes que nous préconisons en vue d'intégrer l'utilisation des ECCs dans les CGA et le protocole SEND. Nous présentons dans la section 5 les résultats d'une étude comparative de performances entre des CGA générées avec RSA et des CGA générées avec ECC, et ce, sur deux types d'équipements : des ordinateurs de type station de travail et des périphériques mobiles à faibles capacités de calcul. Enfin, la section 6 donne nos conclusions.

## 2 Protocole de découverte de voisins sécurisée (SEND)

Le protocole ND (Neighbor Discovery) [NNSS07] repose sur 5 messages de ICMPv6 pour assurer ses fonctionnalités. Dans cet article, nous nous référons aux messages suivants :

- le message *Neighbor Solicitation* permet à un nœud de demander à l'un de ses voisins son adresse physique ;
- le message *Neighbor Advertisement* est la réponse au message *Neighbor Solicitation*, et contient une option transportant l'adresse physique d'un nœud ;
- le message *Router Solicitation* permet à un nœud de demander à un (ou plusieurs) routeur(s) situé(s) sur le même lien des informations sur le réseau ;
- le message *Router Advertisement* est la réponse au message *Router Solicitation*. Il contient des informations sur le réseau telles que : le préfixe de sous-réseau, le MTU (*Maximum Transfert Unit*) du lien, etc. Ces messages peuvent être envoyés spontanément par exemple pour avertir de la présence d'un nouveau préfixe disponible sur le réseau.

Le protocole SEND [AKZN05] spécifie 2 nouveaux messages et 6 nouvelles options aux messages ND. Le but de ces extensions est de sécuriser les échanges d'informations de ND entre les routeurs et les hôtes se trouvant sur le même lien. Un hôte contrairement à un routeur est un nœud n'assurant pas la fonction de routage.

## 2.1 Messages CPS et CPA

SEND offre ainsi de nouvelles fonctionnalités telles que ADD (*Authorization Delegation Discovery*) qui est un mécanisme se basant sur l'échange de messages ICMPv6 pour permettre à un hôte d'authentifier un routeur. Cette fonctionnalité intervient au début des échanges entre les deux entités. Le routeur est autorisé à agir comme tel par une entité de confiance (ou "ancre de confiance") qui lui délivre un certificat. Ce certificat du routeur contient, entre autres, le préfixe de sous-réseau sur lequel le routeur est autorisé à assurer la fonction de routage. Ce dernier, vérifié par l'hôte, permet de s'assurer que le routeur n'est pas un routeur pirate installé par un attaquant.

Les deux messages assurant cette fonctionnalité sont les messages *Certification Path Solicitation* (CPS) et *Certification Path Advertisement* (CPA). Pour pouvoir authentifier un routeur, un hôte envoie un message CPS dans lequel il demande au routeur de lui fournir un chemin de certification jusqu'à son ancre de confiance (*trust anchor*) qui peut prendre la forme d'une autorité de certification). Après réception d'un CPS, le routeur génère un message CPA contenant plusieurs certificats formant le chemin de certification depuis le certificat de l'ancre de confiance jusqu'à son propre certificat de sorte que l'hôte puisse vérifier la chaîne de certification et s'assurer de la légitimité du routeur. C'est donc la validité du certificat d'un routeur qui lui permet d'assurer au sein d'un réseau sa fonction de routage.

## 2.2 Options de SEND

Le protocole SEND introduit 6 options à ajouter aux messages ICMPv6 de ND pour sécuriser les échanges de messages ND. Ces options, illustrées sur les figures 1 et 2 sont les suivantes :

- l'*option CGA* contient la structure de données *paramètres CGA* (présentée dans la section 3.1) ;
- l'*option Signature RSA* contient la signature RSA de chaque message ND qui va être envoyée. Cette signature est effectuée (et vérifiée) sur les champs suivants : les adresses source et destination contenues dans l'en-tête IP, les champs type, code et somme de contrôle de l'entête ICMPv6, tous les champs ICMPv6 se trouvant après la somme de contrôle et avant le début des options, et toutes les options ND et SEND qui se trouvent avant cette option de signature RSA (qui doit être la dernière du message) ;
- l'*option Horodatage* pour contrer les attaques par rejeu. Elle contient la valeur de l'horloge du nœud au moment de la génération du paquet ;
- l'*option Nonce* contient un nombre aléatoire servant à contrer les attaques par rejeu. Cette option permet de s'assurer qu'une réponse à un message est bien liée à ce dernier (et n'est pas rejouée hors contexte) ;
- l'*option Ancre de confiance* inclut le nom et les informations relatives à l'entité de confiance ;
- l'*option Certificat* est utilisée uniquement avec les messages de type CPA et contient un certificat à envoyer à l'hôte (cette option pourra être jointe plusieurs fois dans le même message).

## 2.3 Fonctionnement général de SEND

Quand un nœud se connecte à un réseau, il commence par envoyer un message ND *Router Solicitation* pour obtenir des informations sur les préfixes de sous-réseau disponibles afin de générer son adresse CGA. Quand, en retour, il reçoit un message *Router Advertisement* d'un routeur, il ne peut être sûr ni de l'identité de ce dernier, ni de la validité du préfixe de sous-réseau que ce dernier prétend router. Afin d'authentifier ce routeur, comme l'illustre la figure 1, l'hôte envoie un message CPS comprenant l'option *Ancre de confiance* (en référence à une ancre de confiance que l'hôte connaît). Le routeur répond alors par un message CPA contenant une liste (chaîne) de certificats le reliant à cette entité de confiance.

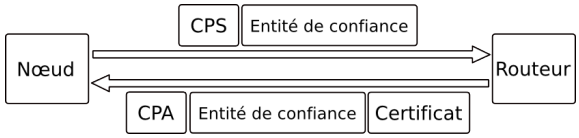


**Figure 1:** Échange de messages Certification Path Solicitation et Certification Path Advertisement entre un nœud et un routeur

Après l'authentification du routeur (via son certificat), le nœud choisit un préfixe pour calculer son adresse CGA et forme la structure de données *paramètres CGA* correspondantes (cf. section 3). Le nœud est par la suite en mesure de communiquer avec ses voisins en utilisant les messages ND complétés par les options de SEND.

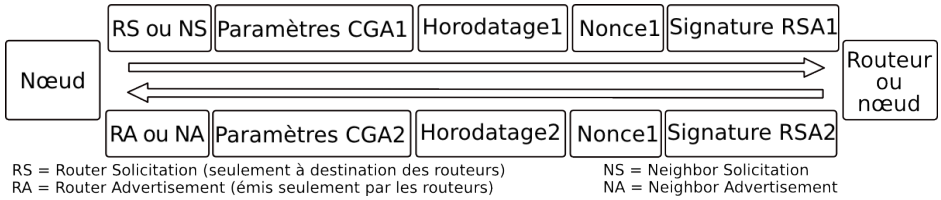


**Figure 2:** Exemple d'échange de messages de découverte de voisins dans un réseau protégé par SEND

Les scénarios d'usage les plus fréquents du protocole SEND (illustrés par la figure 2) sont les suivants :

- Un nœud veut résoudre l'adresse de l'un de ses voisins. Il envoie un message *Neighbor Solicitation* (à destination d'un routeur ou d'un hôte) et y inclut les options suivantes (dans cet ordre) : l'option *CGA*, l'option *Horodatage*, l'option *Nonce* et l'option *Signature RSA*. L'utilisation de l'option *Nonce* est ici obligatoire, car le message envoyé est de type *solicitation*. La réponse correspondante contient la même valeur nonce afin de prouver qu'elle est liée au message *Neighbor Solicitation* (voir figure 2).
- Quand un nœud ou un routeur envoie une annonce sans être sollicité (messages de type *Advertisement*), il n'y a aucun besoin d'y inclure un nonce. Cette situation se présente quand un nœud change d'adresse (physique ou CGA) sur l'une de ses interfaces ou lorsqu'un routeur diffuse de nouvelles informations concernant ses préfixes.

# 3 Processus de génération et de vérification des adresses CGA

L'algorithme de génération des adresses de type CGA se base principalement sur le calcul de deux hashs (ou condensats). Ces hashs sont calculés à partir de certains champs de la structure de données *paramètres CGA* (voir figure 3) et en fonction d'une valeur SEC comprise entre 0 et 7. Cette valeur SEC sert à préciser le niveau de robustesse de l'adresse CGA face à des attaques par force brute.

À l'heure actuelle, les standards CGA imposent la fonction de hachage SHA-1 [Nat08b] et des clés de type RSA. La taille minimale des clés est fixée à 384 bits.

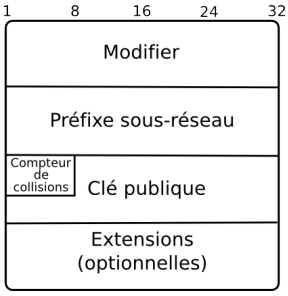


**Figure 3:** Structure de données *paramètres CGA*

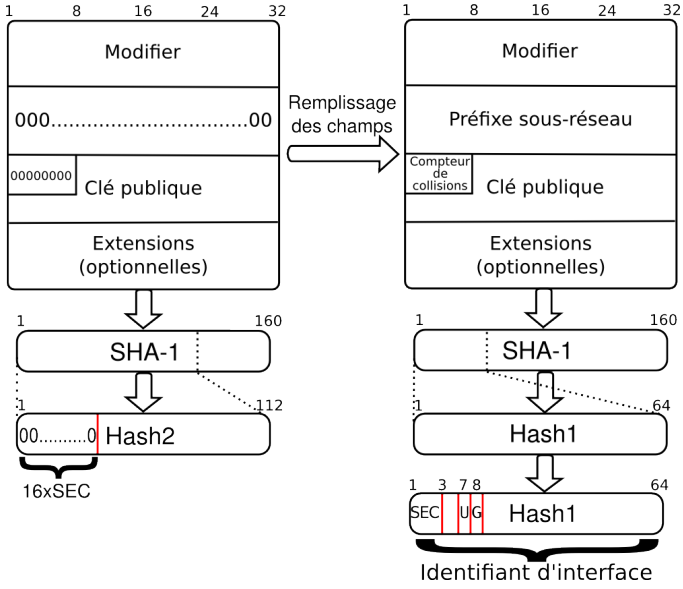


**Figure 4:** Construction de *hash1* et *hash2*

## 3.1 Structure de données paramètres CGA

La figure 3 détaille la structure de données *paramètres CGA* telle que définie dans le document [Aur05]. Cette structure est composée d'un *modifier*, nombre aléatoire sur 128 bits qui ajoute un caractère privé à l'adresse (i.e. *modifier* permettra de générer des adresses CGA différentes pour une même clé publique), d'un *Préfixe de sous-réseau*, d'un *Compteur de collisions* (utile à la génération d'une autre adresse CGA si une duplication d'adresse est détectée sur le réseau), d'une *Clé publique* du nœud encodée au format DER (Distinguished Encoding Rules) et d'un champ *Extensions* facultatif qui permet d'étendre les fonctionnalités des CGA.

## 3.2 Génération des CGA

L'algorithme de génération des CGA est illustré à la figure 4 qui fait appel au calcul de deux hashs :

- Le premier, *hash1*, contient les 64 bits de poids fort de la sortie de la fonction de hachage appliquée à la structure de données *paramètres CGA*. Les trois bits de poids fort y sont remplacés par la valeur de SEC (détaillée plus loin) et les bits $U$ et $G$ sont mis à zéro (pour plus d'informations concernant les bits $U$ et $G$ se référer à [HD06]). Ce hash ainsi modifié représente alors l'identifiant d'interface de l'adresse CGA générée.
- Le second, *hash2*, contient les 112 bits de poids fort du hash résultant de l'application de la fonction de hachage à la structure de données *paramètres CGA* où les champs *Préfixe de sous-réseau* et *Compteur de collisions* sont mis à zéro. La valeur SEC intervient ici et impose au 16×SEC bits de poids fort du *hash2* d'être égaux à 0. Cette condition augmente la complexité de génération des CGA. Elle fut introduite pour augmenter le coût de l'attaque par force brute.

L'algorithme de génération des CGA se décompose en deux grandes étapes (cf. figure 5) :

1. Le calcul du *hash2* et la génération du *Modifier final*. Dans cette boucle, la valeur du champ *modifier* est incrémentée de 1 à chaque fois que *hash2* ne vérifie pas la condition imposée par la valeur de SEC (i.e. les 16×SEC premiers bits du *hash2* ne sont pas à 0). La valeur du *Modifier final* est alors atteinte lorsque *hash2* vérifie cette condition. Cette boucle est très coûteuse en terme de temps et peut représenter un partie importante des calculs effectués lors de la génération d'une CGA. Si ce calcul est parfois coûteux, il est à noter qu'il n'est effectué qu'une seule fois par adresse CGA. Ainsi, un nœud rejoignant un nouveau réseau (avec un nouveau préfixe de sous-réseau), doit calculer une nouvelle adresse CGA mais ne doit pas recalculer le *hash2*.
2. Le calcul de *hash1* et la construction de l'identifiant d'interface de la CGA sont effectués après le calcul du *hash2*, quand la structure de données *paramètres CGA* est complète. L'identifiant d'interface est directement déduit de la valeur de *hash1* (comme nous l'avons présenté précédemment). Ce hash sera obligatoirement recalculé pour adapter l'adresse CGA à chaque changement de préfixe de sous-réseau.

Une fois l'adresse CGA générée, la procédure de détection d'adresse dupliquée classiquement opérée dans IPv6 est lancée conformément au document RFC 4862 [TNJ07]. S'il s'avère que l'adresse nouvellement générée existe déjà, le champ *Compteur de collisions* est incrémenté de 1 et un nouvel identifiant d'interface est calculé sur la structure de données *paramètres CGA* ainsi créée. Après 3 collisions d'adresse détectées, la procédure de génération d'adresse est abandonnée. Lorsque la procédure de détection d'adresse dupliquée se termine correctement (i.e il n'y a pas de conflit d'adresse détecté pour cette adresse), la structure de données *paramètres CGA* contient les valeurs finales des champs *modifier* et *Compteur de collisions*.

## 3.3 Vérification des CGA

À la réception d'un paquet ND, un nœud peut vérifier l'adresse CGA de l'émetteur (cf. figure 6). En effet, le paquet SEND contient dans l'*option CGA* la structure de données *paramètres CGA* et il lui est donc possible de calculer *hash1* et *hash2*.

Plus précisément, le nœud doit vérifier que la valeur du champ *Compteur de collision* n'est pas supérieure à 2. Il vérifie aussi que le préfixe de sous-réseau extrait de l'adresse de

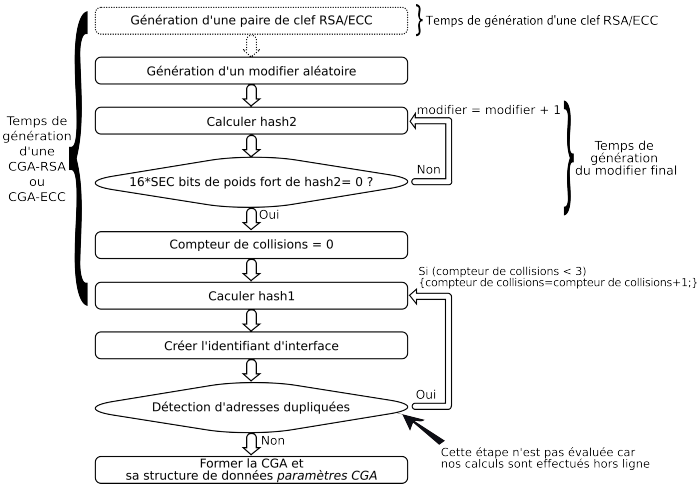


**Figure 5:** Algorithme de génération d'une CGA

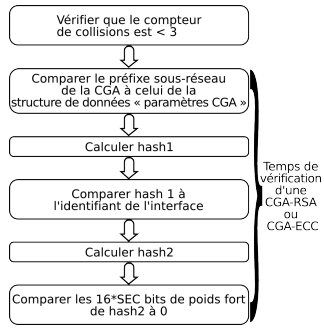


**Figure 6:** Algorithme de vérification d'une CGA

l'émetteur (dans l'en-tête IP) est identique au champ *Préfixe de sous-réseau* contenu dans les *paramètres CGA*. La valeur *hash1* est recalculée et comparée à l'identifiant d'interface (en ignorant les 3 bits de poids fort ainsi que les bits $U$ et $G$). La vérification porte aussi sur les valeurs de Timestamp et de Nonce. La valeur de SEC est extraite de l'adresse (3 bits de poids fort de l'identifiant d'interface) et *hash2* est recalculé. Ce dernier doit vérifier la condition sur ses 16×SEC bits de poids fort (qui doivent être à 0). Enfin, la signature RSA est vérifiée à l'aide de la clé publique explicitée dans la structure de *paramètres CGA*. Si une seule de ses vérifications échoue, alors la procédure de vérification de CGA échoue.

# 4 Utilisation des courbes elliptiques dans CGA et SEND

Comme le décrit la section précédente, les adresses CGA sont fortement liées à l'algorithme de signature numérique qu'elles utilisent et sont basées aujourd'hui sur le RSA et SHA-1. Pourtant, il pourrait être utile de laisser la possibilité de changer d'algorithmes, et ce, pour de multiples raisons : en vue d'un meilleur niveau de sécurité, de meilleures performances pour une intégration sur des équipements de faibles capacités, une réduction du volume de stockage...

Notre réflexion nous a amenés à étudier les possibilités d'intégration de divers algorithmes de signature numérique dans les CGA, en particulier ECC, et ce, de façon rétrocompatible, sans mécanisme de transition brutale. En effet, une première approche naïve d'intégration de ECC serait de reprendre les spécifications de SEND et des CGA et de remplacer toutes les références à RSA par ECC et ECDSA. Cette première approche très simple pourrait fonctionner dans un réseau homogène. Néanmoins, les nœuds implémentant la version précédente des CGA (RFC 3972 [Aur05]) et de SEND (RFC 3971 [AKZN05]) ne pourraient pas comprendre la présence d'une clé ECC dans le champ *clé Publique* (celle-ci n'étant pas RSA). Cette approche non rétrocompatible pourrait empêcher deux nœuds de différentes générations de standard de communiquer, dans le cas du choix exclusif de

ECC par le nœud de nouvelle génération qui souhaite utiliser un protocole. Par exemple, le nœud de nouvelle génération pourrait décider de n'utiliser que ECC pour le faible coût de génération de signature. Aussi, pour résoudre ce problème, nous étendons les CGA afin qu'elles puissent transporter plusieurs clefs publiques. Ces mécanismes font l'objet de deux documents de travail au sein de l'IETF ([CLMSV09b] et [CLMSV09a]).

## 4.1 CGA multi-clés

Les CGA telles que définies dans le document RFC 3972 [Aur05] offrent la possibilité d'étendre les CGA de manière générique. En effet, la structure de données *paramètres CGA* comprend un champ appelé *Extensions* qui suit le format générique donné par le 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*. Le même format est classiquement utilisé dans les certificats X.509 [CSF+08] et permet de déterminer le type de clé publique aisément lors de sa lecture.

Ainsi, un nœud peut être associé à plusieurs clés publiques de différents types. Pour cela, il précisera dans le champ *Clé publique* (lui aussi au format *SubjectPublicKeyInfo*) de la structure de données *paramètres CGA* 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 RSA dans le champ *Clé publique*, ceci dans le but de permettre la lecture de cette clé par les nœuds de génération plus ancienne.

Cette idée de CGA multi-clés est particulièrement intéressante pour les routeurs, car un même routeur peut ainsi communiquer avec des nœuds de différentes générations ayant des exigences quant à l'usage de certains types de clés.

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

## 4.2 Adaptation de SEND à de multiples algorithmes de signature

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* proposée par SEND en utilisant le champ réservé de 16 bits pour préciser à quelle clé, stockée dans la structure *paramètres CGA*, la signature se réfère. Par exemple, la valeur 0 indique que la clé publique est contenue dans le champ *Clé publique*, la valeur 1 indique que la clé publique est contenue dans la première extension de type *Public Key extension*, etc.

Avec l'introduction de multiples algorithmes de signature, il est clair que certains scénarios suivant les 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 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 algorithmes pour des raisons de consommation énergétique. Il est donc nécessaire de passer par une étape de négociation entre les nœuds.

Afin de pouvoir négocier et comprendre le type de clés disponible en chacun des nœuds, nous avons été amenés à définir une nouvelle option pour les messages ND (sécurisés par SEND). Cette option est appelée *Supported Signature Algorithm* (SSA) et contient la liste des algorithmes de signature disponibles sur le nœud. Pour chacun de ces algorithmes, une distinction est faite selon que le nœud supporte uniquement la vérification de signature ou bien les deux : signature et vérification. Cela permet aux nœuds d'annoncer qu'ils comprennent certains algorithmes de signature durant la phase de négociation alors qu'ils ne souhaitent pas signer avec ces algorithmes.

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, chaque nœud détermine si la (les) signature(s) a pu être comprise de son correspondant et, si besoin, il envoie à nouveau le message avec un autre type de signature. Nous avons également défini un nouveau message d'erreur ICMP qui sert à indiquer la raison de l'échec de la découverte des voisins, comme une taille de clé trop faible.

Pour les messages spontanés (comme les *Router Advertisement*), "diffusés" sur le réseau et n'ayant donc pas de destinataire spécifique, il est possible de recevoir en retour un message d'erreur ICMP émis par l'un des multiples récepteurs. Ce message permettra à l'émetteur de rediffuser après coup son message *Router Advertisement*, mais en signant avec un autre algorithme afin de satisfaire (si possible) l'émetteur du message d'erreur. Afin d'ajuster ses messages aux capacités des récepteurs, le routeur pourra poursuivre ses émissions de *Router Advertisement* en les signant avec plusieurs signatures (plusieurs *Signature RSA*).

## 4.3 Extensions de SEND

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 :

- Si cette entité est un routeur, 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 *paramètres CGA*. Pour permettre l'utilisation de CGA à clés publiques multiples (comme le suggère la section 4.1), 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 impraticable (déployer un seul certificat par routeur est déjà assez complexe), nous étendons les certificats X.509 afin de leur faire transporter plusieurs clefs publiques. À l'image des CGA, nous définissons un champ *Extensions* pour transporter nos clefs publiques. Celles-ci sont alors automatiquement signées par le certificat.
- Deux nœuds n'ayant aucun algorithme de signature en commun peuvent-ils rentrer 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, que cette fonction soit certifiée par une autorité dans son certificat de clés publiques X.509 et que deux nouveaux messages ICMP *Signature Check*

*Request* (SCR) et *Signature Status* (SS) soient définis.

### 4.4 Vulnérabilité de la solution CGA multi-clés

La solution de CGA multi-clés ne doit pas dégrader la qualité de la 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. Pour éviter ce type d'attaque, nous supposons que lorsqu'un algorithme de signature ou une fonction de hachage en vient à être compromis, 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.

## 5 Tests de performance et analyse des résultats

Cette section présente les différents tests que nous avons effectués pour évaluer l'apport des courbes elliptiques aux CGA. Des résultats plus détaillés sont aussi disponibles dans le rapport de recherche [BCL08].

### 5.1 Environnement de test

Nous avons mesuré d'une part les temps de génération et de vérification des adresses CGA. Cette évaluation est importante, car en cas de lenteur excessive de la génération ou la vérification des adresses CGA, il est clair que l'usage dont il sera fait de ces adresses CGA dans le protocole SEND influencera ou même restreindra l'utilisation possible.

D'autre part, nous avons mesuré les temps de génération et de vérification des CGA sur deux équipements différents : un Pentium 4 et un Tablet PC. Le premier correspond à une station de travail classique. Le Tablet PC, quant à lui, simule un nœud mobile de faibles capacités de calcul. Il s'agit d'un Nokia N800 avec un processeur compatible ARMv6 cadencé à 400 MHz. Le principal but de l'utilisation du Tablet PC est de savoir si un équipement mobile, avec des performances limitées, est capable de générer une adresse CGA suffisamment rapidement pour envisager son utilisation dans un contexte SEND et de mobilité IPv6 [JPA04].

Pour évaluer finement les temps de génération et de vérification d'une CGA, nous avons utilisé une instruction d'assembleur RDTSC (Read Timestamp Counter) qui est disponible sur le Pentium 4. Celle-ci retourne le nombre de cycles d'horloge interne du processeur à un instant donné. En comparant deux exécutions de cette fonction et en divisant par la fréquence du processeur, on obtient le temps CPU écoulé entre les deux instructions avec une résolution d'un ordre proche de l'instruction.

La méthode précédente présente quelques inconvénients. En effet, il est utile de s'assurer que notre processus n'est pas interrompu par l'ordonnanceur pour donner la main à un autre processus, sinon cette interruption serait comptabilisée dans la mesure. Pour y remédier, nos machines de tests ont été lancées en mode single de façon à limiter le nombre de processus concurrents et donc d'interruptions possibles durant nos phases d'évaluation. L'architecture ARM ne pouvant bénéficier de cette méthode, nous avons été contraints sur Tablet PC d'utiliser la fonction *gettimeofday()* de la bibliothèque *time*. Certes cette fonction est moins précise et comptabilise le temps consommé par les processus concurrents,

néanmoins, les résultats obtenus étant plus grands de plusieurs ordres, il a été décidé que la précision restait convenable.

Les tests ont été réalisés avec des clés ECC de 163, 224, 256, 384 et 571 bits. Le comparatif des performances entre ECC et RSA dans les tableaux de cette section est fourni à niveau de sécurité équivalent, c'est-à-dire pour des longueurs de clés RSA et ECC équivalentes en robustesse (RSA-1024 équivaut à ECC-163). À noter que la bibliothèque OpenSSL utilisée ne proposait pas de ECC 512 bits qui se trouve être équivalent à RSA 15360 bits ; nous avons donc choisi des clés ECC de 571 bits et indiqué dans les tableaux le décalage de niveau de sécurité considéré par une taille de clé RSA de "15360+".

Pour déterminer le nombre d'échantillons à considérer, nous avons d'abord considéré un estimateur de Monte Carlo. Celui-ci s'est révélé inadéquat tant les temps que nous relevions étaient aléatoires du fait de leurs dépendances avec la génération des clés et le calcul du *Modifier final*. Nous avons alors choisi de réaliser 10000 échantillons sur chaque mesure.

Les différents temps mesurés sont :

- *le temps de génération d'une clé RSA/ECC* représente le temps de génération d'une paire de clés RSA ou ECC ;
- *le temps de génération du Modifier final* (avec RSA ou ECC) représente le temps nécessaire à l'obtention du *Modifier final*. S'y trouvent inclus le temps de calcul de chacune des exécutions de SHA-1 ayant permis de satisfaire la condition sur les bits SEC du *hash2* ;
- *le temps de génération d'une CGA-RSA* ou *CGA-ECC* inclut le temps de génération de la clef, l'initialisation du champ *modifier* et le calcul de *hash1* et de *hash2* ;
- *le temps de vérification d'une CGA-RSA* ou *CGA-ECC* inclut la comparaison du préfixe de sous-réseau de l'entête IP à celui contenu dans la structure de données *paramètres CGA*, le calcul de *hash1*, la comparaison de celui-ci avec l'identifiant d'interface de l'adresse source contenu dans l'entête IP, le calcul de *hash2* et la comparaison de ce dernier avec 16×SEC zéros ;
- *le temps de génération de la signature RSA* ou *ECDSA* correspond à la durée de génération de la signature RSA (ou ECDSA) telle que décrite dans la section 3 ;
- *le temps de vérification de la signature RSA* ou *ECDSA* correspond à la durée de la vérification de la signature RSA (ou ECDSA) telle que décrite dans la section 3.

## 5.2 Temps de génération d'une CGA

Dans cette section, nous comparons le temps de génération d'une adresse CGA avec RSA et avec ECC, pour différentes longueurs de clés et pour des valeurs de SEC égales à 0 et à 1. Notons qu'une valeur SEC=0 ne nécessite aucun calcul de *hash2*. Par ailleurs, les clés RSA utilisées ont un exposant public égal à 3 (signatures plus rapides).

Le tableau 1 résume les résultats des tests obtenus sur un Pentium 4 cadencé à une fréquence de 2593 MHz.

Le tableau 1 montre que le temps de génération 0.22 sec d'une adresse CGA avec une clé ECC de 571 bits et SEC=1 est nettement inférieur (ordre de 2) au temps de génération 1.05 sec d'une adresse CGA obtenue avec une clé RSA de 2048 bits et SEC=0 et qui correspond à un niveau de sécurité bien moindre (valeur de SEC différente, clés non équivalentes en termes de robustesse). Il est ainsi facile de déduire que l'emploi des ECC augmente le niveau de sécurité tout en diminuant le temps de génération des adresses.

| Valeur de SEC | 0 | | | | |
|---|---|---|---|---|---|
| Longueur de clé RSA (bits) | 1024 | 2048 | 3072 | 7680 | 15360+ |
| Longueur de clé ECC équivalente (bits) | 163 | 224 | 256 | 384 | 571 |
| Temps de génération d'une CGA-RSA | 0.163964 | 1.055813 | 3.457668 | 92.610627 | - |
| Temps de génération d'une CGA-ECC | 0.006449 | 0.012602 | 0.012622 | 0.020802 | 0.111246 |
| Valeur de SEC | 1 | | | | |
| Temps de génération d'une CGA-RSA | 0.281018 | 1.194061 | 3.601473 | 92.899951 | - |
| Temps de génération d'une CGA-ECC | 0.096317 | 0.108551 | 0.106154 | 0.135056 | 0.224526 |

**Table 1:** Temps de génération (en secondes) d'une CGA utilisant une clé RSA ou ECC sur un Pentium 4 cadencé à 2593 Mhz

De manière générale, dans le cas où SEC=1, le temps de génération d'une adresse est plus grand que celui trouvé quand SEC=0. Cela est prévisible et évident puisqu'un SEC non nul ajoute le calcul du *Modifier final* à l'algorithme de génération des CGA. Pour chaque série de tests avec SEC=1, sont réalisés en moyenne $2^{16}$ calculs de hashs avant de trouver le *Modifier final*.

Évaluer le nombre de hashs calculés avant d'obtenir le *Modifier final* est un problème équivalent au problème de calcul de complexité des attaques par pré-image que l'on trouve dans la littérature [MOVR97]. Nous extrapolons alors la formule du nombre de hashs calculés pour obtenir une pré-image et nous obtenons qu'il faut $2^{SEC*16}$ hashs en moyenne pour trouver le *Modifier final*.

Ces résultats sont confortés par le tableau 2 qui présente, pour SEC=1, le détail des temps de générations des clés RSA et ECC avec les temps de génération du *Modifier final*. En particulier, il nous indique que le temps de génération du *Modifier final* est plus petit lors de l'utilisation de clés ECC. Ceci est logique puisque, comme nous l'avons vu précédemment, le temps de génération du *Modifier final* dépend du temps de calcul de $2^{16}$ hashs en moyenne. Or, comme ces hashs sont calculés en appliquant la fonction de hachage SHA-1 à la structure de données *paramètres CGA* qui contient la clé publique, plus la clef est petite, plus le calcul du hash sera rapide. Les clés ECC, quant à elles, sont plus petites que les clés RSA, et ont clairement un avantage ici.

Il est à noter que nous avons initialement effectué des tests pour des valeurs de SEC=2 sur 10 échantillons (non représentatifs). Nous avons alors constaté qu'il fallait en moyenne 1.8 heure pour calculer une adresse CGA avec une clé RSA de 1024 bits (résultats détaillés dans [BCL08]). En théorie, $2^{32}$ hashs, pour SEC=2, sont nécessaires en moyenne avant d'obtenir une valeur de *Modifier final*, ce qui, sur notre machine, équivaut approximativement à 2.1 heures. Les résultats sont du même ordre et prouvent que cette durée est trop importante pour être utilisable en pratique.

## 5.3 Temps de vérification d'une CGA

La vérification de l'adresse CGA est la première étape effectuée à la réception d'un message SEND. C'est une opération qui se doit d'être rapide. L'algorithme de vérification de la CGA (illustré figure 6) ne contient que deux calculs de hashs (*hash1* et *hash2*) et quelques opérations de comparaisons.

Le tableau 3 fournit les résultats sur la durée de vérification des adresses CGA. Comme

| Longueur de clé RSA (bits) | 1024 | 2048 | 3072 | 7680 | 15360+ |
|---|---|---|---|---|---|
| Longueur de clé ECC équivalente (bits) | 163 | 224 | 256 | 384 | 571 |
| Temps de génération d'une clé RSA | 0.163959 | 1.055806 | 3.457661 | 92.610616 | - |
| Temps de génération d'une clé ECC | 0.006458 | 0.012521 | 0.012679 | 0.020876 | 0.113482 |
| Temps de gén. du *Modifier final* avec RSA | 0.117230 | 0.159470 | 0.202715 | 0.388356 | - |
| Temps de gén. du *Modifier final* avec ECC | 0.089857 | 0.096027 | 0.093472 | 0.114177 | 0.113482 |

**Table 2:** Comparaison entre les temps de génération d'une clé RSA vs ECC, et entre les temps de génération du *Modifier final* RSA vs ECC, et ce pour SEC=1 (en secondes)

dans la section précédente, la taille des clés ECC est à leur avantage et leur permet d'obtenir des temps de vérification plus faibles que les clés RSA. Les fonctions de hachage étant les opérations les plus consommatrices en temps, durant cette phase de vérification des CGA, il y a tout intérêt à fournir moins de données en entrée des fonctions de hachage.

| Valeur de SEC | 0 | | | | |
|---|---|---|---|---|---|
| Longueur de clé RSA (bits) | 1024 | 2048 | 3072 | 7680 | 15360+ |
| Longueur de clé ECC équivalente (bits) | 163 | 224 | 256 | 384 | 571 |
| Temps de vérification d'une CGA-RSA | 0.000004 | 0.000005 | 0.000005 | 0.000009 | - |
| Temps de vérification d'une CGA-ECC | 0.000003 | 0.000004 | 0.000003 | 0.000004 | 0.000004 |
| Valeur de SEC | 1 | | | | |
| Temps de vérification d'une CGA-RSA | 0.000007 | 0.000008 | 0.000009 | 0.000016 | - |
| Temps de vérification d'une CGA-ECC | 0.000004 | 0.000005 | 0.000005 | 0.000005 | 0.000007 |

**Table 3:** Temps de vérification (en secondes) d'une CGA sur un Pentium 4 cadencé à 2593 Mhz

## *5.4 Temps de génération des signatures RSA et ECDSA*

Cette section étudie les résultats concernant les temps de génération et de vérification des signatures RSA et ECDSA. Nous avons pour cela simulé les calculs liés à l'utilisation de l'option de signature RSA définie par SEND. Nous avons généré aléatoirement un message ayant la même taille qu'un message *Neighbor Solicitation* comportant toutes les options SEND (sauf l'option de signature RSA). La taille de ce message varie en fonction de la clé publique utilisée (puisqu'elle est encodée dans l'*option CGA*). Nous obtenons ainsi un message ayant les mêmes propriétés que celui sur lequel est effectuée la signature RSA dans le protocole SEND. Il est à noter que l'adresse CGA ayant déjà été générée à cette étape, la valeur de SEC n'interagit pas dans le calcul de la signature.

Le tableau 4 indique que le temps de génération de la signature RSA augmente en fonction de la taille de la clé RSA, ce qui est prévisible puisque le calcul de la signature dépend de la taille du modulo. Il montre aussi que la vérification d'une signature ECDSA prend plus de temps qu'une signature RSA. Là aussi, des résultats théoriques [GGCS02] confirment ces résultats.

Durant la procédure de Duplicate Address Detection, lors d'une collision, un nœud va recevoir un message de type Neighbor Solicitation (qui indique qu'un nœud désire

| Longueur de clé RSA (bits) | 1024 | 2048 | 3072 | 7680 | 15360+ |
|---|---|---|---|---|---|
| Longueur de clé ECC équivalente (bits) | 163 | 224 | 256 | 384 | 571 |
| Temps de génération signature RSA | 0.004585 | 0.022217 | 0.053573 | 0.609111 | - |
| Temps de génération signature ECDSA | 0.002231 | 0.004396 | 0.004460 | 0.007365 | 0.037403 |
| Temps de vérification signature RSA | 0.000070 | 0.000167 | 0.000322 | 0.001425 | - |
| Temps de vérification signature ECDSA | 0.004379 | 0.005249 | 0.005352 | 0.008850 | 0.074777 |

**Table 4:** Temps de génération et de vérification des signatures RSA et ECDSA sur un Pentium 4 cadencé à 2593 Mhz (en secondes)

s'attribuer une adresse qu'il possède) et devra ensuite répondre par un message de type Neighbor Advertisement (indiquant que l'adresse est déjà prise). Pour le nœud qui assure ses opérations, cela se traduit par une vérification de signature (message NS) et par une génération de signature (message NA). Le ND protocole [NNSS07] indique qu'il faut cette réponse ait lieu dans la seconde suivant l'émission de la requête de type Neighbor Solicitation. Ce temps de réponse inclue le temps de vérification de la signature du message NS et le temps de génération de la signature du message NA. Une attention particulière devra être portée au fait que la somme de ces deux temps ne doit pas dépasser la seconde. Or, nos résultats montrent que les signatures RSA pour des tailles de clés de 7680 peuvent atteindre les 0.609111 seconde.

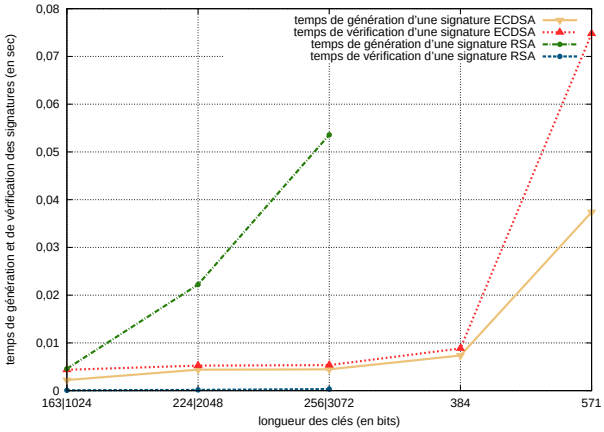


**Figure 7:** Comparaison des temps de génération et de vérification des signatures RSA et ECDSA

La figure 7 montre que les vérifications de signatures ECDSA prennent toujours plus de temps que celles des signatures RSA. Cet inconvénient est compensé par une vitesse de génération de signature plus rapide, ce qui semble indiquer que selon l'usage, l'utilisation d'ECDSA pourrait avoir un impact négatif sur les performances. En effet, si on considère la consommation globale sur l'ensemble des nœuds du réseau, alors nous pouvons considérer deux cas d'utilisation de SEND :

– one-to-one : deux nœuds communiquent ensemble, par exemple, lors de la résolution

d'adresses. Dans ce cas-là, il sera préférable d'utiliser ECDSA, car ECDSA offre un niveau de sécurité supérieur à RSA avec des clés plus petites. Le ratio génération de signature/vérification de signature étant 1 : 1, ECDSA aboutit globalement à de meilleures performances ;

– one-to-many : un routeur diffuse ses messages *Advertisement* à ses voisins. Ici, la signature RSA est avantageuse, car elle est plus rapide à vérifier (par tous les récepteurs).

Suite à ce constat, sachant qu'un même nœud peut signer avec la signature de son choix (cf. section 4), on peut imaginer dans un réseau hétérogène que les routeurs signent leurs messages avec RSA et que les nœuds y préfèrent l'usage d'ECDSA.

## 5.5 Génération d'une adresse CGA sur un Tablet-PC

Dans cette section nous présentons les résultats obtenus lors de la génération d'adresses CGA sur un Tablet-PC. Le but de cette étude est d'étudier le réalisme de SEND dans un environnement mobile.

Pour un plus grand panel de tests sur Tablet PC, nous avons considéré des clés RSA de tailles 384 et 512 bits, bien que ces clés offrent un faible niveau de sécurité. À savoir, le NIST recommande actuellement d'utiliser des clés RSA de taille supérieure à 1024 bits [Nat08a].

| Valeur de SEC | 0 | | | |
|---|---|---|---|---|
| Longueur de clé RSA (bits) | 384 | 512 | 1024 | 2048 |
| Temps de génération d'une CGA/RSA | 0.473715 | 0.694189 | 2.902132 | 18.004494 |
| Temps de génération d'une clé RSA | 0.473634 | 0.694106 | 2.902089 | 18.004389 |
| Temps de génération signature RSA | 0.008384 | 0.011975 | 0.051369 | 0.288404 |
| Temps de vérification signature RSA | 0.000340 | 0.000418 | 0.000852 | 0.002295 |
| Temps de génération du *Modifier final* | 0.000003 | 0.000003 | 0.000003 | 0.000003 |
| Valeur de SEC | 1 | | | |
| Temps de génération du *Modifier final* | 1.114100 | 1.050757 | 1.424263 | 1.892901 |

**Table 5:** Temps de génération d'une CGA-RSA sur un Nokia N800 à 400 MHz (en secondes)

| Valeur de SEC | 0 | | | | |
|---|---|---|---|---|---|
| Longueur de clé ECC (bits) | 163 | 224 | 256 | 384 | 571 |
| Temps de gén. d'une CGA-ECC | 0.088265 | 0.137280 | 0.151946 | 0.302067 | 0.981322 |
| Temps de gén. d'une clé ECC | 0.088172 | 0.137182 | 0.151850 | 0.301966 | 0.981217 |
| Temps de gén. signature ECDSA | 0.000024 | 0.000028 | 0.000028 | 0.000028 | 0.000032 |
| Temps de vérif. signature ECDSA | 0.000102 | 0.000103 | 0.000100 | 0.000102 | 0.000107 |
| Temps de gén. du *Modifier final* | 0.000003 | 0.000003 | 0.000003 | 0.000003 | 0.000003 |
| Valeur de SEC | 1 | | | | |
| Temps de gén. du *Modifier final* | 0.965827 | 1.042549 | 1.056985 | 1.337056 | 1.319589 |

**Table 6:** Temps de génération d'une CGA-ECC sur un Nokia N800 à 400 MHz (en secondes)

Le tableau 5 nous montre que dans le cas SEC=0 (i.e. *hash2* non calculé), le temps de génération d'une adresse CGA avec une clé RSA de 1024 bits dépasse les 2 secondes, ce

qui est très important. Cependant, il est tout à fait possible de rendre ce temps d'attente invisible à un utilisateur en générant les clés à l'avance.

Le temps de génération du *Modifier final* pour SEC=0 (tableau 5 et 6) exprime le temps lié au calcul de *hash2*. Comme il n'y a aucun calcul de *hash2* (SEC=0), les valeurs indiquées correspondent ici à un simple appel de fonction, un test et un retour de fonction.

Pour RSA, dans le cas où SEC=1, le temps de génération du *Modifier final* dépasse la seconde, ce qui implique que le temps total de génération de l'adresse est supérieur à 1 seconde. Il serait donc difficile d'utiliser les CGA avec $SEC \geq 1$ sur des équipements restreints en puissance de calcul sans avoir recours à la pré-génération des valeurs.

Le tableau 6 met en avant l'intérêt des clés ECC. Avec SEC=1, la génération d'une adresse CGA est plus rapide que pour son équivalent RSA. La comparaison peut même être poussée plus loin. Pour des tailles de clés RSA de 1024 ou 2048 bits, et leurs équivalents ECC de 163 et 224 bits, on obtient des temps de génération d'une adresse CGA équivalents mais avec des valeurs de SEC=0 pour RSA et SEC=1 pour ECC, ce qui, pour une même puissance calculatoire, permet d'améliorer sensiblement la résistance aux attaques par force brute sur la fonction de hachage.

On retrouve là encore les mêmes spécificités que pour la station de travail (cf. section 5.2), c'est-à-dire la différence de temps de génération pour SEC=1 entre RSA et ECC due à la différence de taille des *paramètres CGA* sur lesquels se trouve exécutée la fonction de hachage.

Dans le cas SEC=0, il serait réaliste de vouloir générer des CGA avec ECC en générant des clés ECC à la volée si celles-ci ont une taille inférieure ou égale à 256 bits. Au-delà, le temps d'attente pour la génération d'une adresse excéderait les 0.15 seconde et ne serait plus aussi "transparent".

On constate ici que le temps de vérification des signatures ECDSA est inférieur à celui des RSA, ce qui semble contredire la section précédente. On peut attribuer cette différence de performance à l'implémentation sous évaluation (OpenSSL). Cette anomalie justifie notre choix d'une étude pratique sur une étude purement théorique.

Pour conclure sur l'usage des CGA et SEND en environnement mobile, il en ressort qu'il est très faisable d'utiliser des CGA basées sur ECC, ou même sur RSA, dès lors que la taille de la clé reste raisonnable ou que les clés sont générées à l'avance. Les clés ECC et l'algorithme de signature ECDSA montrent ici encore l'intérêt à les fournir pour les CGA et SEND.

# 6 Conclusion

Les CGA et le protocole SEND ont été définis avec l'usage exclusif du RSA et SHA-1. Cependant, avec des tailles de clés recommandées qui ne cessent d'augmenter, le RSA ne sera plus longtemps viable. Pour prévenir le problème, nous avons proposé un mécanisme de transition qui permet l'introduction d'autres algorithmes de signature et fonctions de hachage dans SEND et les CGA. Nous nous sommes intéressés aux performances que pourrait offrir l'usage combiné d'ECC/ECDSA. Nous avons comparé les performances entre RSA et ECC en mesurant les temps de génération des CGA, les temps de génération des signatures et les temps de vérification des signatures, et ce, sur un Pentium 4 et un Tablet PC. Il ressort de notre analyse que les courbes elliptiques s'avèrent très prometteuses aussi bien sur des équipements de puissances importantes que des équipements à faibles

capacités. Ceci est essentiellement dû à la petite taille de clés ECC qui permettent d'offrir un bon niveau de sécurité. Cette étude nous permet d'affirmer que les CGA (à base de courbes elliptiques essentiellement) pourraient être introduites de façon réaliste dans les réseaux filaires et sans fil et satisfaire également le contexte très contraignant de la mobilité des terminaux.

En parallèle, on voit se profiler au sein de l'IETF des améliorations de SEND prônant l'utilisation d'ECDSA pour les signatures et de la seconde génération de fonctions de hachage SHA (SHA-256, SHA-512, etc.). Prochainement, nous soumettrons à l'IETF la solution de transition pour les CGA et SEND en vue d'étendre le choix des algorithmes de signature et des fonctions de hachage.

## Remerciements

Nous remercions Michaela Vanderveen et Sean Shen pour leur aide à la co-écriture du draft ayant permis de présenter le mécanisme décrit en section 4. Nous remercions aussi les relecteurs anonymes de SARSSI 2009 pour leurs remarques et conseils.

Nous tenons également à remercier l'ANR (Agence Nationale de la Recherche) pour son soutien financier apporté au projet TLCOM MobiSEND (Étude de la sécurité de l'auto-configuration et utilisation des outils associés pour les protocoles de sécurité et de mobilité en IPv6).

## Références

[AKZN05] J. Arkko, J. Kempf, B. Zill, and P. Nikander. SEcure Neighbor Discovery (SEND). RFC 3971 (Proposed Standard), March 2005.

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

[BA06] M. Bagnulo and J. Arkko. Cryptographically Generated Addresses (CGA) Extension Field Format. RFC 4581, Internet Engineering Task Force, October 2006.

[Bag07] M. Bagnulo. Hash Based Addresses (HBA) : draft-ietf-shim6-hba-05, December 2007.

[BCL08] A. Boudguiga, T. Cheneau, and M. Laurent Maknavicius. Usage and Performance of Cryptographically Generated Addresses, Research report TELECOM and Management SudParis, 08-014 LOR, October 2008.

[CLMSV09a] T. Cheneau, M. Laurent-Maknavicius, S. Shen, and M. Vanderveen. Signature Algorithm Agility in the Secure Neighbor Discovery (SEND) Protocol. Internet-Draft draft-cheneau-send-sig-agility-00, Internet Engineering Task Force, February 2009. Work in progress.

[CLMSV09b] T. Cheneau, M. Laurent-Maknavicius, S. Shen, and M. Vanderveen. Support for Multiple Signature Algorithms in Cryptographically Generated Addresses (CGAs). Internet-Draft draft-cheneau-cga-pk-agility-00, Internet Engineering Task Force, February 2009. Work in progress.

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, Internet Engineering Task Force, May 2008.

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

[GGCS02] V. Gupta, S. Gupta, S. Chang, and D. Stebila. Performance analysis of elliptic curve cryptography for SSL. In *WiSE '02 : Proceedings of the 1st ACM workshop on Wireless security*, pages 87–94, New York, NY, USA, 2002. ACM.

[HD06] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291, Internet Engineering Task Force, February 2006.

[HMV04] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to elliptic curve cryptography*. Springer, 2004.

[JPA04] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775, Internet Engineering Task Force, June 2004.

[Kem07] J. Kempf. Secure IPv6 Address Proxying using Multi-Key Cryptographically Generated Addresses (MCGAs) : draft-kempf-cgaext-ringsig-ndproxy-00, August 2007.

[MOVR97] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone, and R. L. Rivest. Handbook of applied cryptography, 1997.

[Nat08a] National Institute of Standards and Technology. *Draft FIPS 186-3 : Digital Signature Standard (DSS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, November 2008.

[Nat08b] National Institute of Standards and Technology. *FIPS 180-3 : Secure Hash Standard (SHS)*. National Institute for Standards and Technology, Gaithersburg, MD, USA, October 2008.

[NKN04] P. Nikander, J. Kempf, and E. Nordmark. IPv6 Neighbor Discovery (ND) Trust Models and Threats. RFC 3756 (Informational), May 2004.

[NNSS07] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), September 2007.

[TNJ07] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862, Internet Engineering Task Force, September 2007.