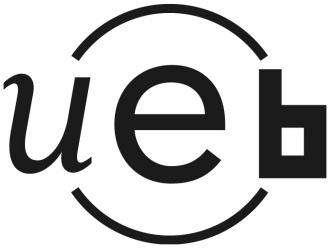
N° d'ordre : 4071 **ANNÉE 2009**



THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

François LESUEUR

préparée à l'unité de recherche EA 4039 (SSIR)
Sécurité des Systèmes d'Information et Réseaux
SUPÉLEC

Autorité de certification distribuée pour des réseaux pair-à-pair structurés : modèle, mise en œuvre et exemples d'applications

Thèse soutenue à Cesson-Sévigné le 27 novembre 2009

devant le jury composé de :

Anne-Marie KERMARREC
Directrice de Recherche, INRIA Rennes / *Présidente*

Ernst BIERSACK
Professeur, Eurécom / *Rapporteur*

Sébastien TIXEUIL
Professeur, UPMC Paris Universitas / *Rapporteur*

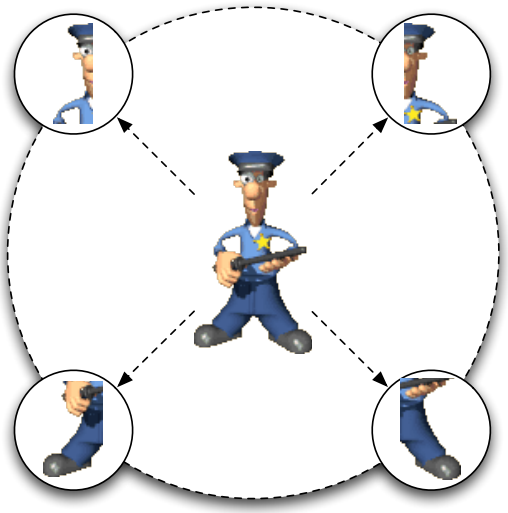
Isabelle CHRISMENT
Professeur, Université Nancy 1 / *Examinatrice*

Emmanuelle ANCEAUME
Chargée de Recherche, CNRS / *Examinatrice*

Hervé DEBAR
Professeur, Télécom SudParis / *Invité*

Olivier HEEN
Chercheur, THOMSON-INRIA / *Invité*

Ludovic MÉ
Professeur, SUPÉLEC / *Directeur de thèse*



DistACP

Distributed Authority for Certification in Peer-to-Peer

Remerciements

Je tiens tout d'abord à remercier l'ensemble des membres de mon jury de thèse. Je remercie Anne-Marie Kermarrec pour avoir accepté de présider ce jury, ce qui fut un honneur ainsi qu'un plaisir. Ernst Biersack et Sébastien Tixeuil ont rapporté ma thèse, je les remercie pour avoir accepté cette tâche malgré leur emploi du temps déjà chargé ainsi que pour le contenu de leurs rapports. Isabelle Chrisment et Emmanuelle Anceaume ont accepté le rôle d'examinatrice et j'ai pu apprécier leurs remarques et questions. Enfin, Hervé Debar et Olivier Heen, que j'ai eu plaisir à côtoyer durant mes années de thèse, m'ont fait le plaisir d'accepter l'invitation à participer à ce jury. Un grand merci à tous, pour votre implication, vos questions pertinentes, vos remarques précieuses et vos commentaires agréables.

Ludo, tu as bien sûr fait partie de mon jury mais tu restes avant tout mon directeur de thèse (ce qui me fait une belle transition !). Un très grand merci pour ces trois années durant lesquelles j'ai réellement apprécié de travailler avec toi. Tu as toujours été d'excellent conseil, à la fois du côté technique et dans la gestion personnelle et professionnelle de cette thèse, ce qui a été au moins aussi important. Valérie, tu as co-encadré cette thèse et je te remercie de ton implication dans ce travail. Tu as su apporter de la rigueur à la fois dans cette thèse et dans ma façon générale de travailler. Merci à tous les deux pour m'avoir offert une thèse pleine d'excellents souvenirs et à laquelle je repenserai toujours avec plaisir !

Durant ces trois années, j'ai rencontré un grand nombre de personnes fort sympathiques qui ont toute une part dans le dénouement de cette histoire. Je pense tout d'abord aux personnels permanents de Supélec participant à toutes les réunions café (Éric, Fred, Christophe, Ben, Laurent, Bernard, Véronique, Roland, Vincent, Renée...). Je pense bien sûr aussi au groupe de thésards transgénérationnel, amis, de SSIR et d'ailleurs (Neerd, Milin, Nicolun, Nicolotre, Domtom, Whylmiark, Mehdi, Dali, Tosh...), qui ont toujours été là dans les moments d'angoisse. Vive la chatroom et vive la lignée des bananiers !

Depuis bien plus longtemps que le début de cette thèse, je remercie également ma famille et mes amis. Ils ont été là durant tout le chemin qui m'a amené à faire une thèse ou m'y ont même incité (Ronan, c'est ton tour maintenant !) ; ils ont également été là dans les moments difficiles de cette thèse (merci Papa qui a tout relu !).

Et enfin, bien sûr, Mélinda. Impossible de résumer en quelques mots tout ce que je voudrais dire... Merci pour m'avoir supporté pendant toute cette période, dans les deux sens du terme. Je sais que ça n'a pas été facile pour toi tous les jours, mais tu m'as toujours soutenu et encouragé quand j'en ai eu besoin. Tu es exceptionnelle et j'ai vraiment de la chance de t'avoir...

Table des matières

Table des figures

Liste des tableaux

Introduction

Les systèmes pair-à-pair permettent de concevoir des réseaux de très grande taille à forte disponibilité et à faible coût. En reposant sur un grand nombre de pairs non fiables (de simples PC d'utilisateurs), les réseaux pair-à-pair atteignent une haute disponibilité grâce à des mécanismes de réplication.

Au delà du partage de fichiers, ces systèmes permettent de mettre en œuvre des systèmes de fichiers distribués (PAST [DR01]), des applications de type *publish/subscribe* (Meghdoot [GSAA04]), de la multidiffusion au niveau applicatif (SplitStream [CDK+03]), de la voix sur IP (Skype [1]) ou encore du *streaming* audio ou vidéo (Spotify [2], Joost [3]).

Au contraire des clients dans les systèmes client-serveur, les pairs d'un réseau pair-à-pair jouent un rôle actif dans le fonctionnement du réseau et fournissent leur bande passante, leur puissance de calcul et leur capacité de stockage.

Le réseau pair-à-pair doit prendre en compte la possibilité que certains de ces pairs soient malveillants et ne se conforment pas au comportement attendu. En outre, afin de permettre le développement de nouvelles applications, les réseaux pair-à-pair doivent pouvoir apporter des mécanismes de sécurité équivalents à ceux proposés par les systèmes centralisés ; ces mécanismes doivent être appliqués malgré la présence de pairs malveillants dans le réseau. Les objectifs de sécurité sont notamment :

- dans un système de fichiers distribué, les utilisateurs doivent pouvoir accorder des droits en lecture ou écriture sur leurs fichiers à certains autres utilisateurs uniquement ;
- dans un système de fichiers distribué ou de partage de fichiers, les utilisateurs doivent être sûrs que le fichier téléchargé est bien valide ;
- dans un système de voix sur IP, l'utilisateur doit être mis en relation avec le bon interlocuteur.

L'obtention de ces propriétés dans un réseau pair-à-pair pose de nouveaux problèmes par rapport à la sécurité des systèmes actuels. En effet, celle-ci repose le plus souvent sur des autorités ponctuelles qui autorisent ou non les opérations demandées : ainsi, pour les trois exemples proposés, une autorité centralisée est actuellement utilisée. Or, une autorité ponctuelle est contraire au paradigme pair-à-pair, dans lequel aucun pair ne doit avoir un rôle critique pour le système

1. `http://www.skype.com`
2. `http://www.spotify.com`
3. `http://www.joost.com`

entier. Une telle autorité ponctuelle n'est pas souhaitable car elle crée :

- un point de faiblesse, une attaque sur cette autorité permettant de rendre certaines fonctionnalités du réseau indisponibles ;
- un point de confiance, tous les pairs du réseau devant nécessairement accorder leur confiance à cette autorité.

Aujourd'hui, si le principe des réseaux pair-à-pair est de repousser la charge sur les pairs, certains mécanismes essentiels de sécurité sont toujours assurés de manière centralisée : dans ce mémoire, nous nous intéressons à la distribution de tels mécanismes de sécurité en refusant tout point central.

Nous nous attachons au cas applicatif des *Tables de Hachage Distribuées* (DHT), qui reposent sur des réseaux pair-à-pair structurés et sont utilisées à la base de nombreuses applications pair-à-pair.

La sécurité étant souvent perçue comme un coût par l'utilisateur, les mécanismes de sécurité doivent, pour être utilisés, être aussi peu intrusifs pour l'utilisateur que possible. Nous considérons par exemple que l'utilisateur ne doit pas avoir à :

- vérifier ou comparer des condensats SHA-1 manuellement [4] ;
- obtenir des clés de chiffrement par un moyen tiers, non intégré à l'application pair-à-pair.

La contribution principale de ce mémoire est une autorité de certification distribuée. Cette autorité permet la signature distribuée de certificats et la distribution sécurisée de clés cryptographiques. Au contraire des autorités de certification centralisées actuellement utilisées, y compris dans des réseaux pair-à-pair, l'autorité que nous proposons est entièrement distribuée dans le réseau pair-à-pair et ce sont les pairs eux-mêmes qui prennent les décisions, par l'accord d'un pourcentage donné d'entre eux.

Dans ce chapitre d'introduction, nous abordons les problèmes de sécurité traités durant cette thèse, nous décrivons l'organisation du mémoire et nous présentons les publications de nos contributions.

Problèmes de sécurité traités

Nous cherchons à appliquer aux DHT les propriété de sécurité usuelles qui sont la disponibilité, l'intégrité et la confidentialité. Ces trois propriétés sont définies de la façon suivante [SCS90] :

- **disponibilité** : les ressources d'un système ouvert sont accessibles et utilisables à la demande d'une entité autorisée ;
- **intégrité** : l'information n'a été ni altérée ni détruite de manière non autorisée ;
- **confidentialité** : l'information n'est ni rendue accessible ni divulguée à des individus, entités ou processus non autorisés.

Dans le cas des DHT, nous appliquons ces propriétés aux données stockées et nous présentons dans le chapitre 1 les problèmes posés et solutions actuelles. Nous nous intéressons particulièrement à l'attaque sybile ainsi qu'à l'intégrité et à la confidentialité des données stockées.

4. Combien de personnes vérifient les condensats des fichiers téléchargés sur internet, lorsque ces condensats sont pourtant indiqués sur la page web ?

Attaque sybile

Dans un réseau pair-à-pair structuré, chaque utilisateur est représenté par un pair et les pairs s'organisent pour router les messages et stocker les données. Dans l'attaque sybile, un attaquant physique crée un grand nombre de pairs et peut ainsi tromper le routage des messages et le stockage des données :

- pour tromper le routage, chaque message reçu par un des pairs de l'attaquant est mal redirigé. Comme l'attaquant possède beaucoup de pairs (et éventuellement certains pairs spécifiques), l'attaquant peut attaquer un grand nombre de routages, voire tous les routages provenant d'un pair donné ou à destination d'une ressource visée ;
- pour tromper le stockage des données, chaque donnée stockée par un des pairs de l'attaquant est modifiée. Si l'attaquant possède tous les pairs répliquant la donnée visée, alors toutes les copies de cette donnée sont altérées par l'attaquant et la donnée originale n'est plus accessible.

Nous détaillons dans l'état de l'art les protections existantes contre l'attaque sybile (approches centralisées de type autorités de certification, tests de ressources informatiques, tests de ressources sociales) et expliquons pourquoi nous pensons que ces protections ne sont pas encore suffisantes.

Intégrité et confidentialité des données stockées

L'intégrité et la confidentialité des données peuvent être obtenues par des mécanismes cryptographiques (signature et chiffrement) : le problème est alors la distribution sécurisée des clés de chiffrement. Une infrastructure à clé publique telle qu'une autorité de certification peut être utilisée pour cela ; cette infrastructure doit cependant être distribuée pour convenir aux réseaux pair-à-pair.

Nous détaillons dans l'état de l'art les mécanismes de certification distribuée existants et expliquons pourquoi ils ne nous semblent pas adaptés aux grands réseaux pair-à-pair.

Organisation du mémoire

Dans le chapitre 1, nous présentons un état de l'art de la sécurité dans les réseaux pair-à-pair. Nous présentons les principaux types de réseaux pair-à-pair – avec centre, non structurés et structurés – et nous choisissons de nous intéresser à la sécurité des réseaux pair-à-pair structurés. Nous détaillons les deux problèmes de sécurité traités dans ce mémoire – l'attaque sybile ainsi que l'intégrité et la confidentialité des données – et les solutions actuelles. Enfin, nous présentons la cryptographie à seuil, permettant de distribuer des opérations cryptographiques, ainsi que ses utilisations dans les réseaux de pairs.

Nous proposons dans ce mémoire une autorité de certification distribuée. Les capacités des utilisateurs (droit d'accès au réseau, possession d'un nom) sont matérialisées par des certificats signés par une clé secrète de réseau ; cette clé secrète est fragmentée dans le réseau et seule la coopération d'un pourcentage fixé des membres permet de signer un certificat. Les certificats sont ensuite vérifiés avec la clé publique du réseau, connue de tous les pairs. La suite du mémoire est séparée en deux parties, à savoir la construction de l'autorité de certification distribuée et son application aux deux problèmes présentés.

Première partie : Construction de l'autorité de certification distribuée

Dans la première partie de ce mémoire, nous présentons le fonctionnement de l'autorité de certification distribuée.

Dans le chapitre 2, nous décrivons le fonctionnement général de cette autorité de certification distribuée. Cette autorité est caractérisée par un couple de clés RSA publique/secrète :

- la clé publique est connue de tous les pairs ;
- la clé secrète (ou privée) est fragmentée dans le réseau, de telle sorte qu'aucun pair ne connaisse toute la clé privée et que la signature d'un certificat requière la coopération d'un pourcentage fixé des pairs.

Ainsi, un certificat n'est obtenu que si la requête est jugée légitime par un pourcentage fixé des membres ; ce certificat est ensuite vérifiable par chaque pair, la clé publique de l'autorité étant connue de tous.

Dans le chapitre 3, nous détaillons l'algorithme de signature distribuée. Cet algorithme déploie un arbre recouvrant sur l'ensemble des pairs devant coopérer ; chaque pair fournit une signature partielle et la structure arborescente permet de répartir le coût de la combinaison de toutes les signatures partielles. Nous présentons également une version de cet algorithme résistante à la présence d'attaquants dans le réseau.

Dans le chapitre 4, nous présentons la maintenance du partage de la clé. Chaque fragment de la clé est connu par un *groupe de fragmentation* contenant quelques dizaines de pairs et la maintenance de la fragmentation consiste à diviser ou fusionner ces groupes lorsque la taille du réseau change. Les opérations présentées sont réalisées sans consensus byzantin et sans synchronisation, afin d'être facilement utilisables dans un contexte pair-à-pair.

Dans le chapitre 5, nous décrivons et évaluons l'implémentation réalisée de l'autorité de certification distribuée, puis nous présentons les résultats obtenus par simulations et calculs théoriques.

Deuxième partie : Applications de l'autorité de certification distribuée

L'autorité de certification déployée, les pairs sont capables de signer des certificats par la coopération d'un pourcentage fixe d'entre eux. Nous présentons dans la deuxième partie deux mécanismes permettant aux pairs de décider de leur coopération à une requête de certification : ces deux mécanismes permettent deux applications de l'autorité de certification distribuée.

Dans le chapitre 6, nous présentons un couplage entre SybilGuard – une protection sociale contre l'attaque sybile – et l'autorité de certification distribuée afin de limiter l'attaque sybile. Nous notons, en effet, que SybilGuard a l'intérêt de limiter le nombre de pairs sybils en testant des ressources sociales, ce qui nous paraît plus équitable que les autres types de tests contre l'attaque sybile, mais ne contraint pas les identifiants à être choisis aléatoirement : SybilGuard ne limite pas les attaques sybiles ciblant un pair ou une ressource particulière. Le couplage avec l'autorité de certification distribuée permet de contraindre l'aléa des identifiants et d'accepter encore moins de pairs sybils.

Dans le chapitre 7, nous proposons un service de nommage sécurisé. Chaque utilisateur peut enregistrer un nom intelligible et obtient, si ce nom est dispo-

nible, un certificat liant sa clé publique à ce nom ; ce certificat est signé par l'autorité de certification distribuée. Chaque utilisateur possédant un certificat lui attribuant un nom intelligible, les autres utilisateurs peuvent obtenir ce certificat à partir de la connaissance de ce nom. Une fois le certificat obtenu, les mécanismes de chiffrement et de signature cryptographiques permettent d'assurer l'intégrité et la confidentialité des données. Ce service de nommage sécurisé permet donc, par exemple, d'attribuer des droits en lecture ou en écriture à certains utilisateurs nommés, comme dans un système de fichiers traditionnel.

Publications de nos contributions

Nos différentes contributions ont donné lieu aux publications suivantes :

- **A Distributed Certification System for Structured P2P Networks**, publié à AIMS 2008 [LMV08b] à partir d'une version initiale en français publiée à SAR-SSI 2007 [LMV07a], présente le modèle de l'autorité de certification distribuée ainsi que la signature distribuée. Cet article présente les travaux décrits ici dans les chapitres 2 et 3 ;
- **Experimenting with Distributed Generation of RSA Keys**, publié à STM 2009 [CL09], présente des expérimentations de génération distribuée de clés RSA avec plusieurs dizaines de pairs. Cet article présente les travaux décrits ici dans la section 2.4 du chapitre 2 ;
- **An Efficient Distributed PKI for Structured P2P Networks**, publié à IEEE P2P 2009 [LMV09], présente les opérations de maintenance sans consensus permettant de faire évoluer le partage de la clé lors de l'évolution du réseau. Cet article présente les travaux décrits ici dans le chapitre 4 ;
- **A Sybil-Resistant Admission Control Coupling SybilGuard with Distributed Certification**, publié à IEEE COPS 2008 [LMV08c], présente le couplage de l'autorité de certification distribuée avec SybilGuard, afin de limiter l'attaque sybile. Cet article présente les travaux décrits ici dans le chapitre 6.

Les articles suivants traitent de travaux non présentés ici :

- **A Sybilproof Distributed Identity Management for P2P Networks**, publié à IEEE ISCC 2008 [LMV08d] à partir d'une version initiale en français publiée à SAR-SSI 2007 [LMV07b], présente une première protection contre l'attaque sybile. Dans ce mémoire, nous ne décrivons pas cette première proposition car une seconde approche, présentée dans le chapitre 6, nous paraît plus intéressante ;
- **Detecting and Excluding Misbehaving Nodes in a P2P Network**, présenté à I^2CS 2008 [LMV08a] et à paraître dans la revue Studia Informatica des éditions Hermann, présente un mécanisme de détection et d'exclusion des pairs malveillants (par révocation de leur certificat). La détection des pairs malveillants était principalement utilisée pour exclure les pairs ne participant pas de manière honnête aux processus de signature distribuée ; face à ce problème, nous avons finalement retenu dans ce mémoire une solution de sélection progressive des pairs honnêtes par chaque pair, solution présentée dans le chapitre 3.

Chapitre 1

État de l'art

« *C'est au pied du mur qu'on voit le mieux le mur.* »
Jean-Marie Bigard

Dans ce chapitre, nous présentons le contexte général dans lequel s'inscrit cette thèse. Nous décrivons les réseaux pair-à-pair, les deux aspects de leur sécurité que nous traitons ainsi que la distribution de clés cryptographiques dans ces réseaux pair-à-pair.

Dans la section 1.1, nous introduisons les différents types de réseaux pair-à-pair, à savoir les réseaux pair-à-pair avec centre, les réseaux non structurés et les réseaux structurés. Pour les réseaux structurés, nous présentons également les DHT. Dans cette thèse, nous nous intéressons à la sécurité des réseaux structurés et plus particulièrement à la sécurité des DHT.

Dans la section 1.2, nous présentons les deux aspects de la sécurité des DHT traités dans cette thèse ainsi que les solutions proposées précédemment et leurs limitations. Le premier aspect est l'attaque sybile, dans laquelle un attaquant se fait passer pour un grand nombre de pairs ou pour certains pairs spécifiques. Les réseaux sociaux nous semblent une défense intéressante mais n'empêchent pas de choisir un identifiant spécifique, ce qui est par contre empêché par une autorité de certification. Le second aspect est l'intégrité et la confidentialité des données stockées dans la DHT. La cryptographie permet d'assurer ces deux propriétés mais pose le problème de la distribution des clés de chiffrement.

Dans la section 1.3, nous présentons les infrastructures à clé publique, qui permettent de distribuer des clés de chiffrement. Nous décrivons les réseaux de confiance et les autorités de certification. Les réseaux de confiance ne nous semblent pas apporter la cohérence nécessaire à un réseau structuré et nous leur préférons les autorités de certification. La distribution des autorités de certification permet de les utiliser dans un réseau pair-à-pair pour :

- empêcher l'attaque sybile, en complément des réseaux sociaux ;
- distribuer des clés de chiffrement permettant de signer et chiffrer les données stockées dans la DHT.

Nous montrons que les autorités de certification distribuées précédemment proposées ne sont pas adaptées à de grands réseaux pair-à-pair et la contribution principale de ce mémoire est ainsi une autorité de certification distribuée

adaptée à ces grands réseaux pair-à-pair. Cette nouvelle autorité de certification distribuée est ensuite utilisée pour contrôler l'accès au réseau, en réponse au problème de l'attaque sybile, et pour distribuer des clés de chiffrement.

1.1 Réseaux pair-à-pair

Dans cette section, nous présentons les différents types de réseaux pair-à-pair. Nous ne présentons pas les nombreux réseaux pair-à-pair de manière exhaustive mais nous essayons plutôt de distinguer les familles classiques de réseaux pair-à-pair, à savoir les réseaux avec centre, les réseaux non structurés et les réseaux structurés. Dans chacun des cas, nous décrivons le principe de ce type de réseau et nous présentons certains exemples.

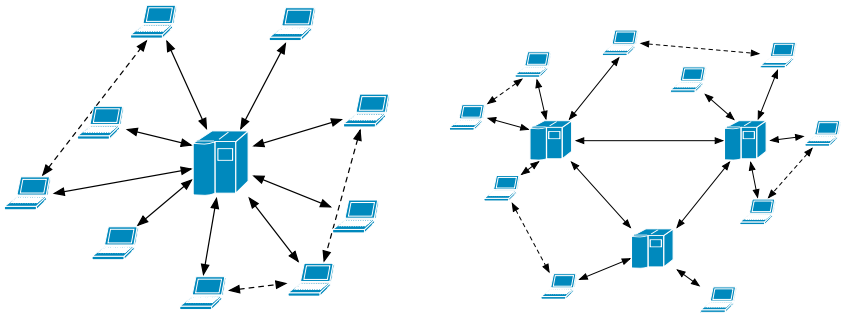
Il est intéressant de distinguer deux phases dans l'évolution des réseaux pair-à-pair. Les premiers réseaux et ceux qui sont encore aujourd'hui les plus connus du grand public sont avec centre ou non structurés. Ils ont été créés par les logiciels qui les utilisent et ont été initiés par des sociétés ou des petits groupes de personnes. Ces réseaux ne sont donc pas tous décrits de manière scientifique ; les éventuels articles scientifiques sont généralement écrits *a posteriori*. Au contraire, les réseaux structurés qui permettent de meilleures performances pour bon nombre d'applications sont des propositions académiques et ont émergé plus récemment vers des implémentations grand public, avec par exemple le réseau Kademlia utilisé par eDonkey (Overnet), eMule (Kad) et BitTorrent (DHT BitTorrent).

Dans la sous-section 1.1.1, nous présentons les réseaux pair-à-pair avec centre. Dans ces réseaux, un serveur central est responsable du maintien de la liste des pairs connectés et des ressources disponibles sur chaque pair ; ce serveur est utilisé comme un annuaire. Nous présentons les exemples de Napster, aujourd'hui fermé, dans lequel un seul serveur existait et était administré par les concepteurs de ce réseau, et de eDonkey/eMule (première version), dans lequel le logiciel serveur est public et un grand nombre de serveurs coexistent et communiquent.

Dans la sous-section 1.1.2, nous introduisons les réseaux non structurés. Ces réseaux sont entièrement décentralisés et propagent les requêtes principalement par inondation. Nous présentons les exemples de Gnutella et de FastTrack (Kazaa/Skype [1]). Dans Gnutella, tous les pairs sont égaux et les requêtes sont propagées par inondation à travers tout le réseau, ce qui implique pour chaque pair une charge linéaire en fonction du nombre de pairs. FastTrack contourne ce problème en promouvant certains pairs en *super-pairs* : chaque pair se connecte à un super-pair et seuls les supers-pairs communiquent par inondation.

Dans la sous-section 1.1.3, nous décrivons les réseaux structurés. Ces réseaux sont également entièrement décentralisés et se caractérisent par un routage très efficace. En utilisant la structure du réseau, chaque pair génère une table de routage et peut router un message quelconque en un faible nombre de sauts, de l'ordre du logarithme de la taille du réseau. Nous présentons les exemples de CAN, Chord et Kademlia qui se différencient par la structure utilisée.

1. Kazaa et Skype partagent la même architecture de communication pair-à-pair



(a) Topologie du réseau Napster. (b) Topologie du réseau eDonkey.

FIGURE 1.1 – Réseaux pair-à-pair avec centre.

1.1.1 Réseaux pair-à-pair avec centre

Nous décrivons d'abord les réseaux pair-à-pair avec centre puis nous présentons les exemples de Napster et eDonkey/eMule.

1.1.1.1 Description

Les premiers réseaux qualifiés de pair-à-pair ne sont pas entièrement décentralisés. Un serveur est utilisé comme annuaire des ressources disponibles ; chaque pair publie les ressources qu'il possède sur ce serveur et interroge ce serveur pour découvrir les pairs partageant les ressources qu'il recherche. Les transferts de données sont en revanche réalisés directement entre les pairs, sans faire intervenir le serveur, ce qui justifie leur qualification de pair-à-pair.

La présence d'un serveur est la limite de ce type de réseau, car le serveur est un point de faiblesse. Ce serveur doit être :

- *puissant* car, bien que n'intervenant pas dans les transferts de fichiers, le serveur est nécessaire lors de chaque insertion, de chaque recherche, ainsi qu'au fur et à mesure du téléchargement pour actualiser les sources. Son dimensionnement est donc critique pour le fonctionnement du système tout entier ;
- *disponible* car, étant nécessaire au fonctionnement du réseau, la coupure du serveur rend le réseau tout entier inopérant.

1.1.1.2 Exemples

Nous présentons dans cette catégorie Napster, illustré figure 1.1(a), et la version originale centralisée de eDonkey/eMule, illustrée figure 1.1(b).

Napster

Napster [2], qui est considéré comme le premier réseau pair-à-pair, a été créé en 1999 par Shawn Fanning. Bien que ne reposant pas sur une architecture entièrement distribuée, le transfert de fichiers qui constitue l'essentiel de la charge

2. `http://web.archive.org/web/*/http://www.napster.com`

est effectué directement entre les utilisateurs. Un serveur est utilisé comme annuaire des ressources disponibles, permettant une recherche et une localisation faciles. Chaque pair s'inscrit sur ce serveur et y publie la liste des fichiers qu'il partage. Lors d'une recherche, le serveur est interrogé et retourne la liste des pairs satisfaisant la requête.

Napster reposait sur un unique parc de serveurs et a été fermé par la simple déconnexion de ce parc. La présence d'un centre est une faiblesse pour la disponibilité d'un réseau pair-à-pair.

eDonkey / eMule

La première version d'eDonkey [HBMS04] a été distribuée en 2000 par Jed McCaleb. Contrairement à Napster, le réseau eDonkey est basé sur l'agrégation d'un ensemble de serveurs. Le logiciel permettant de mettre en place un serveur est disponible publiquement et un grand nombre de serveurs sont donc proposés. Chaque utilisateur se connecte sur un seul serveur qui stocke la liste de ses fichiers partagés : les serveurs ne jouent que le rôle d'annuaire et ne partagent aucun fichier. Lors d'une requête, un utilisateur interroge son serveur qui, à son tour, interroge les autres serveurs. À la différence des systèmes à super-pairs (tels que Kazaa, voir 1.1.2.2), les serveurs exécutent un code différent des clients, demandent d'importantes ressources et nécessitent une administration avancée : la relation entre les utilisateurs et les serveurs est donc bien une relation client-serveur et non pair-à-pair.

La multiplication des serveurs ne résout toutefois pas tous les problèmes que pose Napster. Pour accueillir un grand nombre d'utilisateurs, plusieurs centaines de serveurs sont nécessaires, dont les plus populaires doivent être puissants. En 2004, l'association Razorback ouvrait un nouveau serveur accueillant 730.000 pairs, basé sur un bi-processeur AMD Opteron 248 avec 16GO de mémoire vive et connecté à internet à 50Mbps [3] : à l'époque, il s'agit clairement d'une machine extrêmement puissante et reliée à internet par un lien rapide. Le problème de disponibilité s'illustre ici, comme dans Napster, par la fermeture successive des différents serveurs pour des raisons légales et financières.

Progressivement, les utilisateurs migrent vers les versions non centralisées d'eDonkey/eMule utilisant le réseau structuré Kademlia, présenté dans la sous-section 1.1.3.2.

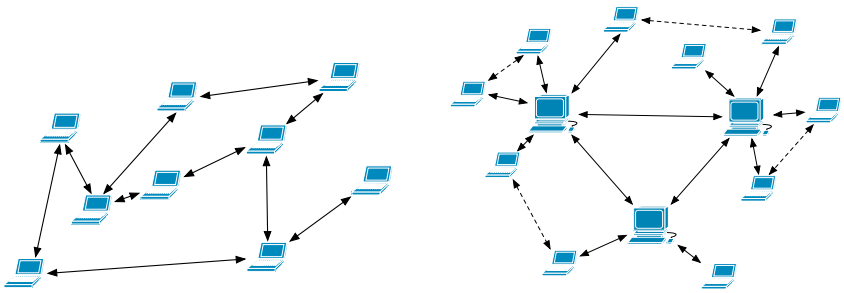
1.1.2 Réseaux pair-à-pair non structurés

Nous décrivons d'abord les réseaux pair-à-pair non structurés puis nous présentons les exemples de Gnutella 0.4 et FastTrack.

1.1.2.1 Description

Les réseaux pair-à-pair non structurés sont entièrement distribués et reposent sur la génération de graphes aléatoires entre les pairs. Chaque nouveau membre doit connaître un pair appartenant déjà au réseau et utilise ce pair pour s'insérer. Les requêtes sont ensuite propagées par *inondation* (demande à *tous* les voisins qui demandent à *tous* leurs voisins, etc.) ou par *marche aléatoire* (demande à *un* voisin qui demande à *un* voisin, etc.).

3. `http://www.razorback2.com/fr/hardware`



(a) Topologie du réseau Gnutella. (b) Topologie du réseau FastTrack.

FIGURE 1.2 – Réseaux pair-à-pair non structurés.

Les paramètres les plus importants des réseaux non structurés sont :
- les *degrés* entrant et sortant de chaque pair. En effet, ils conditionnent directement la robustesse aux défaillances et la connectivité du réseau ;
- le *nombre de sauts* d'un requête (TTL). Plus une requête est propagée, plus elle a de chances de trouver la ressource recherchée, mais plus la charge du réseau augmente ;
- le *choix du ou des voisins* utilisés pour propager une requête. Plus le nombre de voisins retenus est petit, plus le choix de ces voisins est important pour que la requête atteigne la ressource recherchée.

1.1.2.2 Exemples

Dans un premier temps, nous présentons Gnutella 0.4, illustré figure 1.2(a), dans lequel tous les pairs sont égaux et où les requêtes sont transmises par inondation. Ensuite, nous décrivons FastTrack, illustré figure 1.2(b), dans lequel les pairs sont soit des pairs ordinaires, soit des *supers-pairs* et où seuls les supers-pairs transmettent les requêtes par inondation.

Gnutella 0.4

Gnutella 0.4 [Cli00], proposé en 2000 par Justin Frankel et Tom Pepper, est le premier réseau pair-à-pair non structuré. Il se base dans sa version originale sur une inondation totale (transmission à tous les voisins). Chaque requête possède un identifiant unique et est retransmise à tous les voisins pendant TTL sauts, TTL valant au maximum sept [4], ce qui doit assurer une inondation totale *avec une grande probabilité*. Les réponses remontent ensuite en sens inverse, chaque pair sachant de quel pair il avait reçu le message portant cet identifiant.

Ce réseau est entièrement décentralisé, aucun pair n'ayant officiellement un rôle plus important que les autres. Il n'existe aucun point central, aucun point de faiblesse et, par conséquent, aucune autorité ponctuelle. Les requêtes peuvent être arbitrairement évoluées (portant sur le nom du fichier mais également ses méta-données) puisque interprétées par chaque pair du réseau.

4. `http://rfc-gnutella.sourceforge.net/developer/stable/index.html`

Cependant, cette approche pose le problème du passage à l'échelle, la charge de chaque pair augmentant linéairement avec le nombre de pairs du réseau. La version 0.6 intègre un système de super-pairs tel que présent dans FastTrack.

FastTrack

FastTrack, introduit en mars 2001 par Niklas Zennström, Janus Friis et Jaan Tallinn et utilisé notamment par Kazaa [LKR04] et Skype [BS06], est un système pair-à-pair non structuré qui divise les pairs en deux catégories : les pairs ordinaires et les super-pairs.

Chaque pair ordinaire est connecté à un super-pair sur lequel il publie ses fichiers partagés. Lorsqu'un pair ordinaire souhaite rechercher un fichier, il transmet la requête à son super-pair qui la retransmet à son tour par inondation à l'ensemble des super-pairs ; chaque pair ordinaire ayant publié ses partages sur un super-pair, la recherche sur les super-pairs permet de retourner toutes les réponses avec une charge réseau bien inférieure à celle de Gnutella. Le transfert du fichier est ensuite réalisé directement entre les pairs concernés.

Le choix des super-pairs reste cependant problématique puisque, pour un réseau fonctionnant de manière optimale, ces super-pairs doivent être puissants et bien connectés. De plus, Kazaa se limite à deux catégories de pairs quand les différences de capacités entre les pairs peuvent, en fait, atteindre plusieurs ordres de magnitude. Gia [CRB+03], non présenté ici, propose une évolution continue entre pairs ordinaires et super-pairs : chaque pair accepte d'autant plus de connexions d'autres pairs (en devenant ainsi un point de passage important du réseau) qu'il a une grande capacité.

Dans le cas de Skype, un serveur d'authentification est ajouté au centre du réseau. Ce serveur, opéré de manière privée, est responsable de la création des comptes ainsi que de l'authentification.

1.1.3 Réseaux pair-à-pair structurés

Nous décrivons d'abord les réseaux pair-à-pair structurés puis nous présentons les exemples de CAN, Chord et Kademlia.

1.1.3.1 Description

Les réseaux pair-à-pair structurés permettent la connexion d'un très grand nombre de pairs, tout en maintenant une charge faible sur chaque pair et des requêtes rapides. Les pairs et les ressources sont identifiés dans un espace logique commun, appelé *overlay*, sans lien avec l'espace physique. Chaque ressource est affectée de manière déterministe à un pair dans cet espace, comme illustré figure 1.3. La structure permet aux pairs de générer dynamiquement des tables de routage compactes, afin de localiser rapidement une ressource quelconque. Le routage s'effectue par bonds successifs et le nombre de bonds est au maximum en $O(log(N))$, N étant le nombre de pairs du réseau.

Les réseaux structurés sont souvent utilisés à travers l'abstraction des tables de hachage distribuées ou *Distributed Hash Tables*. Une DHT est une structure de données distribuée qui permet de stocker des valeurs persistantes dans le réseau pair-à-pair. Chaque valeur est en effet stockée par plusieurs pairs (les *réplicats*) : ainsi, au départ d'un pair, les valeurs qu'il stockait sont toujours

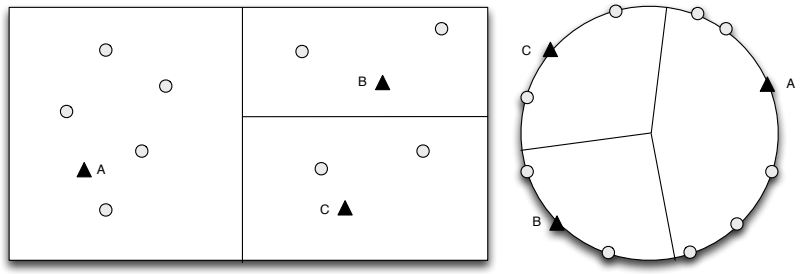


FIGURE 1.3 – Représentation symbolique d'un espace logique partagé entre les pairs et les ressources, où chaque pair est responsable d'une partie de l'espace. Les ▲ représentent les emplacements des pairs dans l'espace logique et les ∘ les emplacements des ressources. À gauche, un espace à 2 dimensions et, à droite, son équivalent à une dimension.

proposées par les autres réplicats. La DHT fournit deux fonctions de haut niveau pour insérer et obtenir des données :

- *insert*$(hash, data)$ insère la donnée *data* à l'emplacement *hash* ;
- *lookup*$(hash)$ obtient la donnée précédemment stockée à l'emplacement *hash*.

Les requêtes dans une DHT sont limitées aux ressources dont l'identifiant est connu (pas de requêtes évoluées comme dans les réseaux non structurés). Il est néanmoins possible de fournir un système de recherche au dessus de la DHT, par exemple *pSearch* de Tang *et al.* [TXM03].

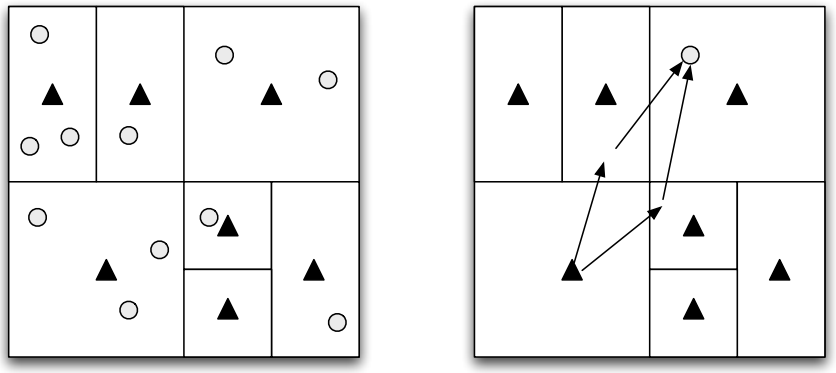
1.1.3.2 Exemples

Nous choisissons ici de présenter CAN, Chord et Kademlia. CAN utilise un espace multidimensionnel alors que Chord et Kademlia utilisent un espace unidimensionnel (un anneau).

CAN

CAN, proposé par Ratnasamy *et al.* dans [RFH+01], propose une structure multidimensionnelle. Pour faciliter la compréhension, nous nous limitons à deux dimensions dans les explications et les illustrations. L'espace logique est séparé en rectangles de tailles variables contenant chacun un pair et ce pair est responsable de toutes les ressources situées dans ce rectangle. Chaque pair est identifié par les coordonnées (x, x', y, y') de l'espace qu'il gère et chaque ressource est repérée par ses coordonnées (x, y), comme illustré figure 1.4(a). CAN rend des services de routage, de réplication et de répartition de charge.

Le routage s'effectue en parcourant l'espace logique vers la ressource, jusqu'à atteindre le bon rectangle. Chaque pair connaît tous ses voisins immédiats dans l'espace logique ainsi que la zone que chacun d'entre eux couvre. Lorsqu'un pair veut router un message, il envoie ce message vers un pair qui le rapproche de sa destination, au sens de la distance dans l'espace logique. Plusieurs pairs peuvent convenir et plusieurs routages sont donc possibles : le pair optimise le routage en envoyant, par exemple, son message vers un pair ayant une latence faible.



(a) Structure de l'espace virtuel de CAN, en 2 dimensions.

(b) Exemples de routages possibles.

FIGURE 1.4 – Réseau structuré CAN.

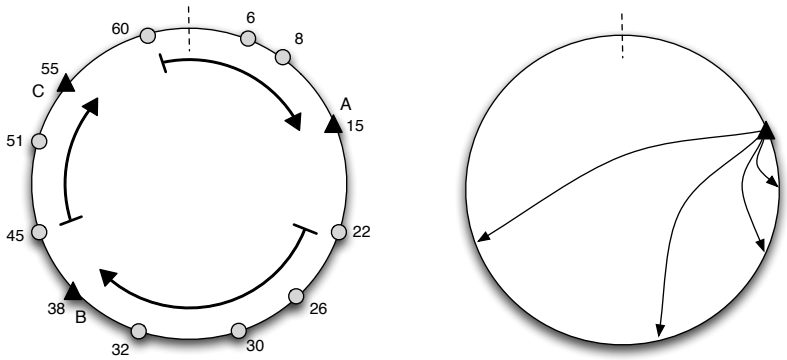
Ces multiples possibilités permettent également à CAN d'être naturellement résistant aux défaillances. Le routage est illustré sur la figure 1.4(b).

La référence [RFH+01] décrit comment utiliser CAN pour fournir une DHT. Une DHT doit assurer la disponibilité des valeurs insérées, même quand des pairs quittent le système, et proposer des mécanismes de cache pour les données fortement demandées :

- pour conserver une donnée même quand le pair responsable quitte le réseau, la redondance des données peut être assurée de deux manières :
  - en insérant plusieurs pairs dans chaque zone, chacun répliquant toutes les données de la zone. Cela permet également, lors du routage, de choisir le pair ayant la latence la plus faible pour transmettre le message à la zone ;
  - en maintenant plusieurs *réalités* (espaces logiques), chaque pair occupant une zone différente dans chaque réalité et chaque ressource ayant un identifiant commun à toutes les réalités. Les ressources sont ainsi répliquées sur autant de pairs qu'il existe de réalités. Un pair choisit de router un message dans la réalité où il est le plus proche de la destination ;
- pour gérer les données fortement demandées, en plus des mécanismes de réplication présentés, CAN propose deux mécanismes de cache :
  - chaque pair maintient un cache des données récemment recherchées ;
  - un pair responsable d'une ressource très demandée réplique cette ressource sur tous ses voisins. Étant donné le mécanisme de routage, chaque requête provient forcément d'un de ces voisins et cette méthode divise automatiquement la charge par le nombre de voisins.

Chord

Chord, proposé par Stoica *et al.* dans [SMK+01], repose sur une structure en anneau représentant 2^n valeurs. Usuellement, n vaut 160, l'identifiant d'un pair est un condensat SHA-1 réalisé à partir de son adresse IP et l'identifiant



(a) Assignation des ressources aux pairs dans Chord. Ici, les ressources 60, 6 et 8 sont assignées au pair A.

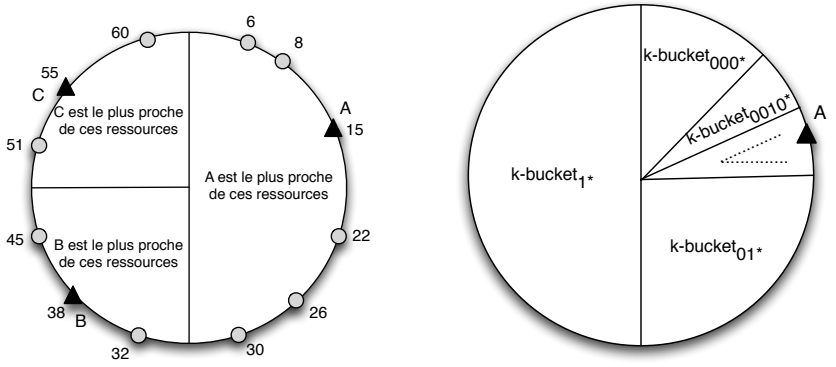
(b) Ensemble des raccourcis contenus dans la *finger table* du pair A.

FIGURE 1.5 – Réseau structuré Chord.

d'une ressource est le condensat SHA-1 de la donnée stockée. Chaque ressource d'identifiant *id* est assignée au pair possédant le premier identifiant supérieur à *id* (figure 1.5(a)). Chord fournit uniquement le routage d'un message vers le pair responsable de la ressource demandée, la DHT étant fournie par la couche supérieure.

Pour permettre le routage des messages, les pairs forment une liste simplement chaînée, chacun connaissant son successeur. Les messages peuvent ainsi être routés le long de l'anneau, tant que l'identifiant du pair courant est inférieur à l'identifiant recherché. Dès qu'un pair reçoit une requête pour une ressource d'identifiant inférieur au sien, il sait qu'il est responsable de cette ressource : en effet, le pair précédent a nécessairement un identifiant inférieur à la ressource demandée. Ce routage simple a cependant une complexité en $O(N)$, N étant la taille du réseau. Pour obtenir un routage plus efficace, chaque pair maintient également une *finger table* (figure 1.5(b)) contenant des raccourcis dans le routage. Cette table de 160 entrées contient chaque successeur de $peerId + 2^i$, $peerId$ étant l'identifiant du pair courant et i étant compris entre 0 et 159. Visuellement, il s'agit des successeurs dans le sens des aiguilles d'une montre en faisant des sauts de taille double à chaque fois. Avec cette table des raccourcis, la complexité du routage est en $O(log(N))$ et les tables de routage ont également une taille en $O(log(N))$.

DHash [DKK+01] est la DHT proposée au-dessus de Chord. Chaque ressource est répliquée sur plusieurs pairs suivant le pair responsable afin de garantir sa disponibilité. Quand un pair recherche une ressource, il obtient le pair précédant le responsable en utilisant le routage Chord puis demande à ce pair la liste de ses successeurs, qui sont les réplicats utilisés par la ressource : il choisit de télécharger depuis un des réplicats trouvés, par exemple celui ayant la latence reportée la plus faible. La donnée téléchargée est enfin stockée en cache sur l'ensemble des pairs utilisés lors du routage de cette requête, de futures requêtes ayant en effet de fortes probabilités de passer par ces mêmes pairs.



(a) Assignation des ressources aux pairs dans Kademlia. Ici, c'est le pair C qui est le plus proche des ressources 51 et 60.

(b) Table de routage du pair A. Chaque k-bucket contient k pairs situés dans l'autre moitié de l'*overlay*, l'autre quart, etc.

FIGURE 1.6 – Réseau structuré Kademlia.

Kademlia

Kademlia, proposé par Maymounkov et Mazières dans [MM02], repose également sur une structure unidimensionnelle. Kademlia est un *overlay* proche de Pastry [RD01] (de Rowstron et Druschel) mais n'utilise qu'une seule table de routage là où Pastry maintient une table de routage pour les identifiants éloignés ainsi qu'une table des voisins. Nous choisissons ici de présenter Kademlia car ses implémentations (Overnet, Kad, DHT BitTorrent) sont celles déployées à la plus grande échelle, tous réseaux structurés confondus. Tout comme dans Chord, les pairs et les ressources sont identifiés sur 160 bits. Chaque ressource est affectée au pair le plus proche au sens de la distance XOR (figure 1.6(a)) et répliquée sur les autres pairs les plus proches. Les tables de routage sont arborescentes et Kademlia fournit simultanément le routage pair-à-pair et une DHT.

La table de routage de chaque pair est composée de k-buckets (jusque 160). Chacun de ces k-buckets contient k pairs (usuellement, $k = 20$) situés à une distance XOR comprise entre 2^i et 2^{i+1} de lui même, avec $0 \leq i < 160$, ce qui est illustré sur la figure 1.6(b). L'ensemble des k-buckets d'un pair peut contenir n'importe quel autre pair du réseau et chaque pair peut ainsi compléter ses k-buckets et les mettre à jour à partir des requêtes qu'il reçoit, sans échanger de messages supplémentaires.

La fonction de routage permet de localiser les k pairs les plus proches d'un identifiant id donné, au sens de la distance XOR. Pour cela, le pair transmet une recherche de id aux α pairs les plus proches de id dans sa table de routage (usuellement, $\alpha = 3$). Chacun de ces pairs répond avec les k pairs les plus proches de id qu'il connaît. Le pair initiateur répète ce processus jusqu'à obtenir un ensemble stable de k pairs : ces k pairs ne connaissent pas de pairs plus proches et sont donc les plus proches de l'identifiant recherché. La parallélisation des requêtes ainsi que le choix d'α pairs parmi k permettent de réduire la latence et de tolérer la présence d'attaquants durant le routage.

Pour publier et retrouver des ressources, chaque ressource est dupliquée sur les k pairs les plus proches de l'identifiant de cette ressource :

- pour insérer une donnée, un pair localise les k pairs les plus proches de l'identifiant souhaité auxquels il transmet la donnée à stocker. Dans le cas de KAD, l'implémentation de Kademlia utilisée par eMule, l'insertion est légèrement différente, puisqu'une ressource est insérée sur 10 pairs qui ont les 8 premiers bits en commun avec l'identifiant de la ressource ;
- pour obtenir une ressource, le processus est identique, à l'exception qu'un pair intermédiaire possédant la ressource dans son cache renvoie directement cette réponse (auquel cas le pair initiateur arrête le processus de localisation). Le pair initiateur transmet cette réponse au pair selon lui le plus proche qui ne connaît pas cette ressource : ce pair stocke cette ressource dans son cache, ce qui permet d'augmenter la réplication des données fortement demandées.

La proposition développée dans ce mémoire est présentée dans le cadre du réseau Kademlia, bien qu'elle puisse être utilisée avec d'autres réseaux structurés, moyennant quelques modifications.

Résumé et positionnement

Nous avons présenté dans cette section les trois grandes familles de réseaux pair-à-pair, dont les principaux avantages et inconvénients sont rappelés tableau 1.1 :

- les réseaux avec centre sont une première approche du pair-à-pair mais ne sont pas satisfaisants. En effet, la présence du centre impose un point de confiance et de faiblesse qui va à l'encontre des principes du pair-à-pair ;
- les réseaux non structurés sont, eux, entièrement décentralisés et reposent sur des communications par inondation, totale ou partielle. Ce mode de communication induit une charge croissante sur les pairs, lorsque le réseau grandit ;
- les réseaux structurés sont également entièrement décentralisés et permettent un routage des messages efficace. La structure permet la génération de tables de routage et les pairs localisent les ressources en un faible nombre de sauts. Les DHT, utilisées sur ces réseaux structurés, permettent de stocker des données persistantes dans le réseau, grâce à des mécanismes de réplication.

Dans cette section, nous nous sommes volontairement limité à ces trois grandes familles de réseaux pair-à-pair. D'autres approches sont cependant à la croisée de différents types de réseaux, parmi lesquelles :

- dans [CSWH00], Clarke *et al.* proposent Freenet, un réseau anonyme non structuré dans lequel des identifiants sont attribués aux ressources. Bien que non structuré, le réseau s'organise spontanément pour s'approcher du routage dans un réseau structuré ;
- dans [CCR05], Castro *et al.* proposent un réseau structuré distinguant pairs ordinaires et super-pairs. Chaque pair ordinaire se connecte à un super-pair, comme dans Kazaa, mais les super-pairs sont ensuite reliés par un réseau structuré dans lequel ils redirigent les requêtes des pairs ordinaires. Le coût de maintenance de la structure est ainsi réduit ;
- dans [LSW06], Locher *et al.* proposent eQuus, un réseau structuré en hypercube. Dans eQuus, les pairs sont organisés en cliques et ce sont les

Réseaux pair-à-pair avec centre

Avantages	Inconvénients
+ Recherches évoluées + Charge minimale sur les pairs	– Disponibilité du serveur – Coût du serveur – Confiance dans l'administrateur du serveur

Réseaux pair-à-pair non structurés

Avantages	Inconvénients
+ Recherches évoluées + Décentralisation	– Charge élevée sur les pairs (augmentation linéaire)

Réseaux pair-à-pair structurés

Avantages	Inconvénients
+ Charge faible sur les pairs + Décentralisation + Stockage des données dans les DHT	– Recherches basiques – Structure à maintenir

TABLE 1.1 – Comparaison des différents types de réseaux pair-à-pair

cliques qui sont les sommets de l'hypercube. En quelque sorte, chaque super-pair de [CCR05] est simulé par une clique ;
– dans [GHH+06], Guerraoui *et al.* proposent GosSkip, un réseau structuré qui permet des recherches dans un intervalle au lieu des recherches exactes des DHT.

Dans ce mémoire, nous avons choisi de nous intéresser à la sécurité des réseaux structurés, bien que la plupart des problèmes que nous évoquons soient également présents dans les autres types de réseaux pair-à-pair. Nous illustrons plus précisément nos propos dans le cas des réseaux à une dimension (Chord, Pastry, Kademlia, etc.) mais les problèmes et solutions proposées sont généralement transposables aux réseaux à plusieurs dimensions.

1.2 Sécurité des réseaux pair-à-pair structurés

Après avoir présenté les différents types de réseaux pair-à-pair, nous étudions dans cette section les problèmes de sécurité posés dans les réseaux structurés. Plus précisément, nous nous attachons dans ce mémoire à garantir des propriétés de sécurité dans le cadre d'une DHT. En effet, les DHT sont utilisées en support d'un grand nombre d'applications (systèmes de fichiers, *publish/subscribe*, messagerie instantanée) et les propriétés assurées par les DHT profitent à toutes ces applications.

Les propriétés de sécurité sont la disponibilité, l'intégrité et la confidentialité. Dans le cadre d'une DHT, les propriétés de disponibilité, d'intégrité et de

confidentialité s'appliquent aux valeurs stockées. Dans ce mémoire, nous nous intéressons à deux aspects de la sécurité des DHT, aspects que nous décrivons dans cette section :

- la lutte contre l'attaque sybile ;
- le contrôle de l'intégrité et de la confidentialité des ressources de la DHT.

Dans la sous-section 1.2.1, nous présentons l'attaque sybile ainsi que les protections existantes. Un attaquant crée des identifiants supplémentaires et se fait ainsi passer pour un grand nombre de pairs ou pour certains pairs spécifiques. Cette attaque permet de censurer ou modifier une ressource, espionner les requêtes pour une ressource ou encore filtrer l'accès d'un pair honnête : il s'agit d'une attaque contre la disponibilité et l'intégrité. Nous décrivons précisément cette attaque, nous présentons les protections actuelles, centralisées ou distribuées, et nous analysons leurs limites.

Dans la sous-section 1.2.2, nous présentons le contrôle de l'intégrité et de la confidentialité des ressources en l'absence d'attaque sybile. Dans ce cas, l'intégrité et la confidentialité doivent être assurées en présence d'un pourcentage raisonnable (le pourcentage d'attaquants physiques) et uniformément distribué (les identifiants des pairs attaquants sont aléatoires) de pairs attaquants. Nous décrivons les différentes méthodes utilisées pour contrôler l'intégrité et la confidentialité des ressources dans une DHT, nous analysons leurs limites et nous expliquons l'intérêt de pouvoir distribuer des clés cryptographiques de manière sécurisée.

Nous ne traitons pas dans cette section les aspects de sécurité suivants qui sont tout aussi importants mais hors du contexte de notre étude :

- la sécurité du routage vise à garantir que les requêtes parviennent bien aux pairs recherchés en l'absence d'attaque sybile. Ce point, étudié d'abord par Sit et Morris dans [SM02] et pour lequel Castro *et al.* ont proposé de premières protections dans [CDG+02], concerne à la fois le routage des requêtes et la maintenance des tables de routage :
  - le routage des requêtes peut être vérifié par routage redondant [CDG+02, HB06, FSY05] ou par signature des pairs responsables [BRK07] ;
  - la corruption des tables de routage (attaque éclipse où un pair honnête devient isolé du reste du réseau) peut être empêchée en contraignant les entrées de la table de routage [CDG+02], en limitant les degrés des pairs [SNDW06] ou en créant de l'agitation artificiellement [AS04, AS06, Sch05, CKS+06].
- l'équité vise à empêcher des pairs de profiter du système sans y contribuer, la puissance et la robustesse des réseaux pair-à-pair reposant sur la participation conjointe des pairs. Ce problème, abordé pour la première fois dans le cadre de Gnutella [AH00], peut être partiellement résolu par des systèmes donnant-donnant tel que dans BitTorrent [Coh03] ou de comptabilité de la participation de chacun [HS05] ;
- l'anonymat empêche d'identifier le producteur et le consommateur d'un contenu, ce qui protège la vie privée et permet la communication d'informations dans certaines zones limitant la liberté d'expression. Freenet [CSWH00] est le réseau anonyme le plus connu.

1.2.1 Attaque sybile

Afin d'atteindre une haute disponibilité à partir de pairs à faible disponibilité, les réseaux pair-à-pair reposent sur les deux hypothèses suivantes :

- une donnée est répliquée sur plusieurs pairs et les fautes (malveillantes ou non) des réplicats sont indépendantes ;
- chaque pair connaît plusieurs autres pairs (entrées de sa table de routage) et les fautes de ces pairs sont également indépendantes.

Si chaque pair choisit un unique identifiant aléatoire, alors ces deux hypothèses sont justifiées car les réplicats d'une même ressource et les entrées de la table de routage d'un pair :

- sont physiquement situés à des endroits différents et donc la coupure du lien réseau d'un réplicat n'impacte pas les autres réplicats ;
- ont une probabilité d'être malhonnêtes égale à la probabilité que chaque pair du réseau soit malhonnête.

L'attaque sybile, définie par Douceur dans [Dou02], affaiblit ces deux hypothèses. Dans cette attaque, un attaquant choisit certains identifiants spécifiques afin de contrôler tous les réplicats d'une ressource (attaque à l'intégrité et à la disponibilité de cette ressource) ou tous les pairs connus par une victime (attaque à la disponibilité pour cette victime) [5].

1.2.1.1 Description de l'attaque sybile

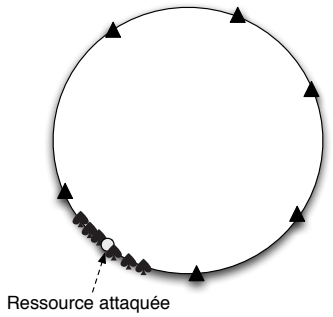
L'attaquant tente d'obtenir certains identifiants spécifiques en fonction de son but, pour :

- corrompre une ressource (figure 1.7(a)). L'attaquant crée des pairs ayant des identifiants sur lesquels la ressource visée est répliquée. Dans le cas de Chord, par exemple, l'attaquant choisit les pairs successeurs de la ressource visée et héberge ainsi toutes les répliques de la ressource : il peut la modifier ou la rendre indisponible. Dans le cas de Kademlia, l'attaquant choisit des identifiants proches de la ressource visée. Les mécanismes de réplication sont basés sur l'hypothèse que les fautes des réplicats sont indépendantes ; dans le cas d'une attaque sybile, cette hypothèse n'est plus vérifiée et la réplication est corrompue ;
- isoler un pair honnête (figure 1.7(b)). L'attaquant crée des pairs ayant les identifiants que la victime va chercher pour remplir sa table de routage. Dans le cas de Chord toujours, si la victime a pour identifiant $peerId$, alors l'attaquant choisit les identifiants $peerId + 2^i$, i représentant le nombre d'entrées dans les *finger tables*. L'attaquant intercepte ainsi toutes les requêtes de la victime et peut les modifier ou les abandonner. Les tables de routage tolèrent des défaillances ponctuelles et indépendantes ; dans le cas d'une attaque sybile, ces défaillances ne sont plus décorrélées.

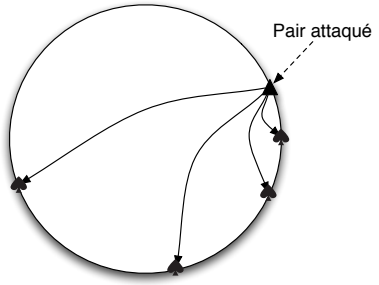
Pour réaliser cette attaque, l'attaquant n'a besoin que de quelques identifiants bien choisis. Selon les cas, il peut soit générer directement ces identifiants soit générer un grand nombre d'identifiants puis en choisir quelques-uns parmi eux. Une protection contre l'attaque sybile doit donc à la fois :

- limiter le nombre d'identifiants qu'un attaquant peut générer ;
- contraindre ces identifiants à être réellement aléatoires.

5. Une attaque sybile peut également être utilisée pour se protéger de botnets pair-à-pair ; dans [DFN+08], Davis *et al.* utilisent une attaque sybile contre le botnet STORM.



(a) Attaque d'une ressource dans Kademlia. L'attaquant place des pairs sybils autour de la ressource visée et possède tous les réplicats de la ressource : l'attaquant peut modifier cette ressource ou la censurer.



(b) Attaque d'un pair dans Chord. L'attaquant place des pairs sybils aux emplacements où la victime va chercher les entrées de sa table de routage : l'attaquant peut filtrer les accès de la victime.

FIGURE 1.7 – Attaques sybiles sur une ressource et un pair. Les ▲ sont les pairs honnêtes et les ♠ sont les attaquants.

Steiner *et al.* ont étudié plus précisément cette attaque dans le cas du réseau déployé Kad dans [SENB07a]. Les auteurs ont montré qu'il était possible avec peu de moyens de masquer des ressources ou encore d'espionner les requêtes vers des ressources visées. Dans [CF05], Cheng et Friedman montrent qu'une approche de réputation symétrique entre les pairs ne peut pas contrer l'attaque sybile. Intuitivement, avec un système de réputation symétrique, un attaquant peut dupliquer le réseau de réputation puis augmenter arbitrairement sa propre réputation : une dissymétrie doit nécessairement être introduite.

1.2.1.2 Protections contre l'attaque sybile

Les protections contre l'attaque sybile sont fondées sur le test de ressource et peuvent être divisées en trois catégories :

- les autorités centralisées, telles que des autorités de certification, testent des ressources administratives (cartes d'identité, ressources financières) ;
- les tests de ressources informatiques vérifient des ressources de calcul ou de stockage ;
- les tests de ressources sociales utilisent les relations entre les utilisateurs.

Autorités centralisées

Dans l'approche par autorité centralisée, chaque pair doit obtenir un droit d'accès personnel en présentant une ressource administrative supposée unique (adresse IP, carte d'identité). L'hypothèse est qu'un attaquant ne peut pas se faire passer pour plusieurs personnes auprès d'une autorité ponctuelle.

Dans Chord, les identifiants sont des condensats SHA-1 créés à partir de l'adresse IP du pair et d'un identifiant de pair virtuel. Plusieurs identifiants doivent être autorisés pour chaque adresse IP afin de permettre à plusieurs pairs partageant une même adresse IP publique (partage de connexion NAT) de

rejoindre le réseau mais un pair ayant une unique adresse IP ne doit pouvoir générer qu'un nombre limité d'identifiants. Un attaquant peut cependant acheter un ensemble d'adresses IP et générer ainsi un très grand nombre d'identifiants. Dans [DH06], Dinger et Hartenstein limitent le nombre de pairs à quelques-uns par adresse IP, grâce à un mécanisme d'enregistrement distribué. Un attaquant ayant accès à un ensemble d'adresses peut toujours générer beaucoup d'identifiants.

Dans [DR01], Druschel *et al.* proposent d'utiliser des cartes à puce délivrées par une autorité de certification. En échange d'une preuve de l'identité et d'une certaine somme d'argent, cette autorité délivre un unique certificat à chaque personne. Dans [RLS02], Rodrigues *et al.* proposent une certification équivalente reposant sur une fédération d'autorités. Cependant, le déploiement de ces autorités est complexe et onéreux, là où les réseaux pair-à-pair permettent justement un déploiement aisé et à faible coût ; la présence de cette autorité va de plus à l'encontre des principes du pair-à-pair en créant un point de confiance et de faiblesse. Au delà des réseaux pair-à-pair, nous pouvons d'ailleurs noter que ce principe d'autorité de certification, tout en étant relativement ancien, ne s'est principalement imposé que pour l'authentification de grandes institutions telles que des banques ou des sites de vente en ligne, à travers le protocole `https` ; les autres tentatives visant l'authentification de l'utilisateur final, par exemple pour sécuriser les échanges par courrier électronique, se sont pour la plupart soldées par des échecs [6]. Les difficultés rencontrées par les autorités de certification sont développées par Davis dans [Dav96] ou par Ellison et Schneier dans [ES00].

L'approche par *Trusted Computing* de Balfe *et al.* [BLP05] est similaire, la carte à puce étant remplacée par un module TPM intégré dans l'ordinateur et l'autorité certifiant des machines au lieu de personnes : une unique personne peut obtenir autant de TPM qu'elle peut acheter d'ordinateurs. L'autorité de certification peut également utiliser des captcha [7] afin de limiter la vitesse d'obtention des identifiants ; un attaquant peut, cependant, publier à son tour ces captcha sur son propre site pour les faire résoudre par des humains, éventuellement attirés par un spam. Utilisée seule, cette solution est donc insuffisante.

Tests de ressources informatiques

Dans l'approche par test de ressources informatiques, chaque pair doit effectuer des calculs pour rejoindre le réseau. L'hypothèse est qu'un attaquant a des ressources de calcul limitées et ne peut donc maintenir qu'un nombre limité de pairs.

Dans [CDG+02], Castro *et al.* proposent l'utilisation d'un puzzle calculatoire. L'identifiant du pair dans le réseau est le condensat de la clé publique du membre et les membres du réseau ne doivent choisir que des clés dont le condensat se termine par au moins p bits égaux à 0. L'objectif est de limiter

6. En France, l'administration fiscale a toutefois su déployer une autorité certifiant les personnes, dans le cadre de la télédéclaration des revenus. Cette certification a été rendue possible par la connaissance préalable d'un ensemble de données connues des deux parties, dont certaines non publiques (état-civil, numéro fiscal, référence de l'avis), ce qui implique un travail en amont de la certification. Cependant, malgré ces connaissances préalables, l'administration fiscale est finalement revenue en arrière et propose aujourd'hui également une authentification classique par mot de passe, sans certificat.

7. Un captcha est un test de reconnaissance de symboles brouillés, simple pour un humain mais difficile à résoudre par une machine.

le nombre d'identités calculables : en effet, le coût de calcul nécessaire pour trouver une telle paire de clés est en $O(2^p)$. Le problème de cette approche est qu'un attaquant peut pré-calculer un grand nombre de clés et parvenir ainsi à contrôler un grand nombre de pairs.

Pour remédier au pré-calcul statique des identifiants, Borisov propose dans [Bor06] un schéma de puzzle calculatoire dans lequel les *challenges* sont basés sur une source d'aléa distribuée et sont rafraîchis à intervalle régulier. Un attaquant ne peut donc pas pré-calculer les résultats mais un attaquant disposant de suffisamment de puissance de calcul peut toujours calculer plusieurs *challenges* en parallèle et ainsi maintenir plusieurs pairs.

Dans [REMP07], Rowaihy *et al.* proposent un système arborescent basé sur des puzzles calculatoires. La racine de l'arbre est une tierce partie de confiance et les premiers niveaux de cet arbre sont également considérés comme honnêtes. Le reste de l'arbre est constitué d'autres membres du réseau ayant une connexion stable. Pour rejoindre le réseau, un pair doit résoudre successivement les puzzles fournis par une branche entière de l'arbre, ce qui lui donne le droit d'obtenir un certificat de durée limitée par le pair racine. Ce pair racine est ainsi un point de confiance et de faiblesse, ce qui va à l'encontre des bases des réseaux pair-à-pair. Une fois ce point de confiance établi, les apports de l'arborescence de membres ne sont pas clairement établis. Dans la version longue [8], les auteurs notent que l'arborescence permet de distribuer la charge de vérification des puzzles : en effet, la racine ne vérifie pas de puzzles mais chacun de ses fils vérifie un puzzle pour chaque nouvel arrivant. Ce premier niveau de l'arbre devrait donc être suffisant. Les auteurs font de plus l'hypothèse que le réseau est d'abord rempli de pairs honnêtes avant l'arrivée des attaquants, ce qui n'est pas forcément le cas en réalité où quelques attaquants peuvent faire partie des pairs initiaux. Cet article présente des résultats expérimentaux : dans un réseau à 8000 pairs, un seul attaquant ayant une puissance de calcul seulement 8 fois supérieure à la puissance attendue peut obtenir 8% des identifiants.

De manière générale, les tests de ressources informatiques supposent que les utilisateurs ont des puissances de calcul relativement équivalentes, comme Borisov le note dans [Bor06]. Il faut en effet à la fois empêcher des attaquants de maintenir plusieurs pairs, tout en permettant aux utilisateurs honnêtes de facilement maintenir un pair. Dans la pratique, un attaquant peut acquérir quelques machines puissantes pour tromper ces systèmes. Anderson et Fedak notent, de plus, dans [AF06] que les ressources de calcul sont très hétérogènes entre les utilisateurs, avec plusieurs ordres de magnitude de différence. Un attaquant peut enfin louer un *botnet* pour quelques centaines de dollars et obtenir la puissance de calcul de centaines voire de milliers de machines infectées. Nous notons également que les mécanismes présentés ne contraignent pas fortement l'aléa des identifiants utilisés, à l'exception de [REMP07] où cet aléa dépend de l'intervention ponctuelle de la racine de l'arbre : ces mécanismes doivent donc être couplés à de l'agitation artificielle [AS04, AS06, Sch05, CKS+06] pour protéger des attaques ciblées, ce qui induit un surcoût pour la réplication et la maintenance des tables de routage.

8. Non publiée, disponible en ligne à l'adresse `http://www.cse.psu.edu/~rowaihy/publications/Sybil_long.pdf`

Tests de ressources sociales

Dans l'approche par test de ressources sociales, chaque pair doit posséder une reconnaissance sociale antérieure pour accéder au réseau. L'hypothèse est qu'un attaquant possède un nombre limité de relations sociales et en tous les cas comparable au nombre de relations d'un utilisateur honnête. La création d'une telle identité n'implique aucun coût matériel et semble juste. La création d'identités à coût social a déjà été utilisée dans des systèmes autres que le pair-à-pair. Par exemple, GMail proposait à son lancement un espace de stockage bien supérieur à ses concurrents et Google avait alors mis en place un système d'invitations qui empêchait un attaquant d'ouvrir un nombre illimité de comptes GMail.

Danezis *et al.* proposent dans [DLLKA05] un routage résistant à l'attaque sybile basé sur les liens sociaux. À partir du premier membre du réseau, chaque nouveau membre se connecte au réseau en utilisant un pair qu'il connaît : un arbre implicite des membres est ainsi créé, dans lequel les membres se connaissant appartiennent à un même sous-arbre. Lors du routage d'une requête, un pair diversifie les composantes sociales utilisées en choisissant pour chaque saut un pair éloigné dans l'arbre des membres : ce pair n'a *a priori* pas de lien social avec le pair précédent et un attaquant ne peut pas piéger une requête dans le sous-arbre de ses pair sybils [9].

Dans [YKGF06a], Yu *et al.* proposent SybilGuard qui est basé sur les graphes sociaux et empêche un attaquant inséré de créer un nombre illimité d'identités. Chaque utilisateur crée des connexions avec ses amis et un attaquant est limité par le nombre de connexions qu'il peut créer vers la partie honnête des utilisateurs : les composantes sybiles du graphe social sont peu connectées au reste du graphe honnête et SybilGuard utilise cette limite, comme illustré figure 1.8. Dans la partie honnête du graphe SybilGuard, qui correspond à un graphe social, deux marches aléatoires partant de deux utilisateurs quelconques se croisent rapidement, avec une grande probabilité. Ces marches aléatoires sont, en outre, créées pour converger, c'est-à-dire que deux marches venant du même utilisateur repartent vers le même utilisateur : le nombre de marches aléatoires, et donc le nombre de pairs sybils, qu'un attaquant peut faire entrer dans la partie honnête du graphe est limité. SybilGuard est *relativement simple* à déployer : *simple* car chaque pair doit juste créer des connexions avec ses amis, *relativement* car l'ajout de connexion nécessite un échange de clé cryptographique. Cet échange de clé cryptographique est effectué par un moyen tiers, par exemple par courrier électronique, ce qui rajoute une étape et dépend de la sécurité du moyen tiers utilisé. Dans [YGKX08], Yu *et al.* proposent SybilLimit, une amélioration qui réduit le nombre de pairs sybils acceptés. SybilGuard et SybilLimit n'assurent pas la cohérence des pairs acceptés par chaque pair honnête, c'est-à-dire que chaque pair honnête a une vue locale des pairs qu'il considère comme non sybils : un même pair peut être accepté comme honnête par un pair mais détecté comme sybil par un autre, ce qui peut interférer avec les mécanismes de réplication ou de maintenance des tables de routage dans les réseaux structurés. De plus, SybilGuard et SybilLimit limitent le nombre de pairs sybils mais ne contraignent pas

9. Ce système présente des similitudes avec la proposition de Marti *et al.* [MGGM04] permettant d'utiliser des liens sociaux dans le routage ; ce dernier a cependant été proposé pour améliorer la disponibilité du routage en dehors d'une attaque sybile et ne protège pas de cette attaque.

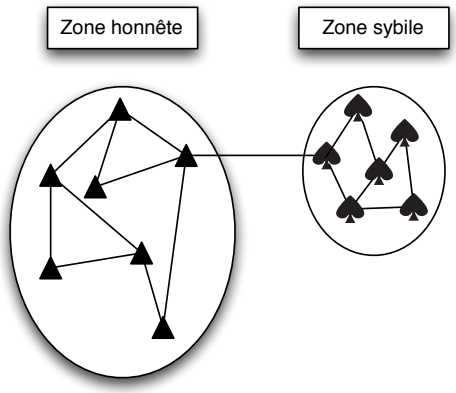


FIGURE 1.8 – Attaque sybile dans un graphe social. Les ▲ sont les pairs honnêtes, les ♠ sont les pairs sybils et chaque arc représente une relation sociale entre deux utilisateurs. La partie sybile du graphe, contenant les pairs sybils d'un unique attaquant physique, est peu connectée à la partie honnête.

les identifiants de pairs à être choisis aléatoirement, ne protégeant ainsi que d'un aspect de l'attaque sybile. Nous détaillons plus précisément SybilGuard dans le chapitre 6 où nous proposons une protection contre l'attaque sybile combinant SybilGuard à l'autorité de certification distribuée proposée dans ce mémoire.

Dans [DM09], Danezis et Mittal proposent SybilInfer qui, tout comme SybilGuard et SybilLimit, utilise la faible connectivité des pairs sybils au reste du réseau honnête. SybilInfer suppose la connaissance du graphe social en intégralité, ce qui est possible pour des réseaux sociaux tels que Facebook [10], et calcule les emplacements de ces frontières de connectivité. En contrepartie de cette connaissance globale, qui peut se révéler problématique dans un réseau pair-à-pair (les auteurs proposent cependant une piste de travail dans cette direction), SybilInfer accepte moins de pairs sybils que SybilGuard et SybilLimit.

1.2.1.3 Résumé et positionnement

Dans une attaque sybile, un attaquant physique crée un ensemble d'identifiants spécifiques ou un grand nombre d'identifiants parmi lesquelles il en choisit certains. Nous avons présenté dans cette sous-section les trois approches contre l'attaque sybile, qui sont résumées tableau 1.2 :

- les autorités centralisées, par exemple les autorités de certification, permettent à la fois de limiter le nombre d'identifiants d'une personne et de contraindre l'aléa de ces identifiants. Cette approche impose cependant un point central de confiance et de faiblesse, ce qui va à l'encontre des principes des réseaux pair-à-pair. De plus, l'opération d'une autorité de certification et la vérification des données des utilisateurs est complexe et coûteuse ;
- le test de ressources informatiques, en imposant à chaque pair de prouver la possession de ressources de calcul, permet de limiter le nombre de pairs

10. `http://www.facebook.com`

Protection	Avantages	Inconvénients
Autorité centralisée	+ Contrôle fort possible des identités et des identifiants + Équitable, si le coût n'est pas payé par les utilisateurs	– Coût, éventuellement payé par les utilisateurs – Centralisation
Test de ressources informatiques	+ Déploiement simple + Contrôle distribué	– Peu équitable – Possibilité pour un attaquant d'acheter de la puissance de calcul – Pas de contrôle des identifiants
Test de ressources sociales	+ Équitable + Contrôle distribué	– Insertion plus difficile – Dépend de la structure du graphe / de l'application – Pas de contrôle des identifiants

TABLE 1.2 – Comparaison des approches contre l'attaque sybile.

qu'un utilisateur peut créer et maintenir sur une même machine mais ne contraint pas les identifiants de pairs à être choisis aléatoirement. La calibration de la difficulté du test est complexe car un pair honnête disposant d'une puissance modeste doit pouvoir passer le test : un attaquant possédant des ressources de calcul supérieures à la normale peut ainsi maintenir un nombre important de pairs ;

– le test de ressources sociales, en utilisant le graphe social des utilisateurs du système, permet de limiter le nombre de pairs qu'un même utilisateur peut créer mais ne contraint pas les identifiants de pairs à être choisis aléatoirement. Cette ressource a l'avantage d'être plus équitable entre les pairs que la puissance de calcul. Dans la plupart des protections présentée, dont SybilGuard que nous réutilisons dans ce mémoire, la vue des différents pairs n'est pas nécessairement cohérente, chaque pair jugeant localement l'honnêteté des autres pairs : cette incohérence d'appartenance au réseau peut interférer avec la réplication et le routage dans les réseaux structurés.

Toutes les propositions présentées sont donc perfectibles et l'importance de nouvelles protections contre cette attaque efficaces et adaptées aux réseaux pair-à-pair est notamment illustrée par Levine *et al.* dans [LSM06]. Dans cet article, les auteurs inventorient les solutions retenues dans un ensemble d'articles proposant des applications pair-à-pair et exhibent les limites des solutions purement distribuées, auxquelles sont finalement souvent préférées des solutions centralisées :

- la majorité des articles (27 articles) proposent l'approche par certification centralisée, qui rompt les bases du pair-à-pair ;
- au deuxième rang (17 articles), les articles font l'hypothèse de l'absence de cette attaque sans recommander de solution ;
- viennent ensuite les tests de ressources (financières, matérielles ou sociales) des participants, éventuellement de manière récurrente.

Les tests de ressources sociales nous paraissent intéressants par la répartition plus uniforme de cette ressource. Les approches présentées jusqu'ici n'adressent cependant pas les deux aspects de l'attaque sybile puisqu'elles ne contraignent pas l'aléa des identifiants utilisés. De son côté, l'approche centralisée par autorité de certification pose certes le problème du contrôle des identités mais contraint l'aléa des identifiants et assure la cohérence de l'appartenance au réseau. Dans le chapitre 6 de ce mémoire, nous présentons un mécanisme couplant SybilGuard [11] à une autorité de certification distribuée que nous proposons dans le chapitre 2. Le couplage de ces deux protections permet de limiter le nombre d'identifiants et de contraindre l'aléa de ces identifiants, adressant ainsi les deux aspects de l'attaque sybile, tout en assurant la cohérence de l'appartenance au réseau. Nous montrons que notre mécanisme autorise encore moins de pairs sybils que SybilGuard seul. Nous proposons également dans le chapitre 7 de remplacer l'échange manuel de clés cryptographiques entre amis par un système automatique et sécurisé ne nécessitant de connaître que le nom de l'utilisateur à ajouter, comme pour ajouter un ami dans un système de messagerie instantanée ; ce problème de distribution des clés est également présent dans le contrôle de la confidentialité et de l'intégrité des données, ce que nous présentons dans la sous-section suivante.

1.2.2 Intégrité et confidentialité des ressources

L'intégrité et la confidentialité des ressources peuvent être obtenues par des mécanismes cryptographiques, symétriques (une seule clé) ou asymétriques (un couple de clés publique/secrète) :

- l'intégrité est assurée par la signature de la donnée. En cryptographie symétrique, la même clé permet de signer puis de vérifier la signature [12]. En cryptographie asymétrique, la signature est réalisée avec la clé privée et la connaissance de la clé publique associée permet de vérifier cette signature ;
- la confidentialité est assurée par le chiffrement de la donnée. En cryptographie symétrique, la même clé permet de chiffrer puis de déchiffrer. En cryptographie asymétrique, le chiffrement est réalisé avec la clé publique et seule la connaissance de la clé privée associée permet de déchiffrer la donnée initiale.

La difficulté réside alors dans la gestion et la distribution des clés. En effet, l'intégrité et la confidentialité ne sont assurées dans ce cas que si chaque partie possède la même clé (cryptographie symétrique) ou la clé publique de l'autre partie (cryptographie asymétrique). Avant l'obtention des clés, un attaquant

11. SybilGuard est alors utilisé par chaque pair pour évaluer chaque autre pair candidat à rejoindre le réseau : SybilGuard pourrait être remplacé par tout mécanisme équivalent tel que SybilLimit ou une version distribuée de SybilInfer.

12. La signature n'est donc pas imputable à une personne unique mais à *l'une* des personnes connaissant la clé.

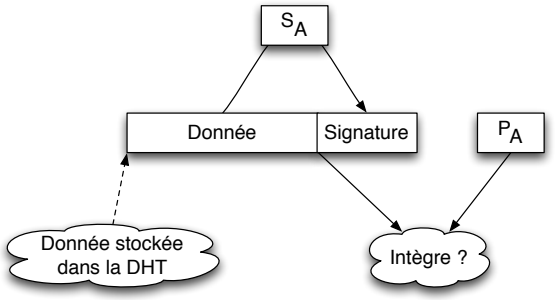


FIGURE 1.9 – Contrôle d'intégrité d'une donnée. La donnée, insérée par le pair A, est signée avec la clé secrète S_A de A. L'obtention de la donnée et de la signature permet, moyennant la connaissance de la clé publique P_A de A, de vérifier l'intégrité de la donnée.

peut se faire passer pour l'autre partie et fournir une clé qu'il connaît : il peut ensuite intercepter les communications et éventuellement les retransmettre à l'autre partie, sans qu'aucune des deux parties ne puisse remarquer l'attaque (attaque *man-in-the-middle*).

Nous expliquons dans cette sous-section comment assurer l'intégrité et la confidentialité des données dans un réseau pair-à-pair en utilisant notamment des mécanismes cryptographiques. Nous étudions les mécanismes de distribution de clé dans la section suivante.

1.2.2.1 Intégrité des ressources

Moyennant un mécanisme de distribution des clés, l'intégrité des données dans une DHT peut être assurée par signature cryptographique asymétrique. Une donnée étant signée par la clé privée d'un pair, ce pair peut ensuite mettre à jour cette donnée et les utilisateurs connaissant sa clé publique peuvent vérifier l'intégrité de la donnée à chaque instant, comme illustré figure 1.9. Le contrôle d'intégrité des données peut-être utilisé dans les contextes suivants :

- dans un système de fichiers distribué, chaque pair doit s'assurer que le fichier obtenu est bien le fichier qui a été enregistré par l'utilisateur légitime. Chaque pair signe le contenu d'un répertoire avec sa clé privée et ce pair peut ensuite mettre à jour le contenu de ce répertoire ; les utilisateurs connaissant sa clé publique peuvent vérifier l'intégrité du répertoire à chaque instant ;
- dans un système de messagerie instantanée, chaque pair doit s'assurer qu'il communique bien avec la personne souhaitée. Brian *et al.* proposent par exemple dans SOSIMPLE [BLJ05] un système SIP de voix sur IP utilisant une DHT. Chaque utilisateur enregistre son adresse IP dans la DHT à l'emplacement correspondant au condensat de son identifiant et une requête dans la DHT permet ensuite d'entrer en communication avec un autre utilisateur. Il faut donc empêcher un attaquant de se faire passer pour une tierce personne enregistrée. Si l'enregistrement dans la DHT est signé, l'appelant peut vérifier l'intégrité de l'adresse IP associée à la personne appelée.

L'intégrité des fichiers peut également être garantie par des chemins auto-signés, ce qui a été initialement proposé par Mazières *et al.* dans SFS [MKKW00]. Dans le cadre de la recherche d'un fichier dans une DHT, le fichier contenant la donnée *data* est stocké à l'emplacement $h(data)$: le demandeur recherchant la ressource $h(data)$ peut vérifier que le condensat de la valeur obtenue vaut bien $h(data)$. Dans le cas d'un système de fichiers, chaque pair A est muni d'un couple de clés publique/secrète noté (P_A, S_A) et publie ses données dans un dossier racine d'identifiant $h(P_A)$: le chemin d'accès aux fichiers de A contient $h(P_A)$, ce qui permet de vérifier une clé P_A obtenue de manière non sécurisée. La difficulté est dans ce cas reportée sur l'obtention sécurisée du condensat racine : il n'est en effet pas possible d'obtenir un fichier directement à partir d'un nom intelligible. L'utilisateur doit soit connaître un condensat, ce qui est peu pratique (un condensat SHA-1 est représenté par 160 bits, soit 40 chiffres hexadécimaux), soit utiliser une recherche non sécurisée. Ces chemins auto-certifiés sont par exemple utilisés dans CFS [DKK+01].

L'intégrité des ressources peut enfin être vérifiée par redondance. Si un pair insère une donnée de manière redondante (c'est-à-dire qu'il envoie lui même la donnée aux différents réplicats), alors un utilisateur peut télécharger cette donnée (ou au moins son condensat) à partir de plusieurs réplicats : tant que la majorité des réplicats est honnête, la majorité des réponses obtenues contient la donnée intègre. Cette méthode a l'inconvénient d'augmenter la bande passante nécessaire ainsi que la latence. De plus, elle est incompatible avec les mécanismes de cache. En effet, dans le cas d'une donnée présente dans le cache d'un pair, ce pair peut directement renvoyer la donnée au lieu de continuer le routage, ce qui est nécessaire, d'une part, pour assurer la disponibilité des ressources populaires sans surcharger les pairs qui en sont responsables et, d'autre part, pour améliorer les performances. Baumgart utilise cette méthode dans [Bau08] pour garantir l'intégrité et l'unicité des enregistrements SIP. Chaque requête est transmise de manière redondante par des routes disjointes et la ressource est demandée à chaque pair réplicat : la réponse majoritaire est considérée comme la réponse valide.

Dans l'approche par signature, une fois le coût de gestion des clés payé, une donnée signée peut être obtenue par un routage rapide sans redondance. Cette propriété est par exemple importante dans un système de résolution de noms, où le possesseur d'un nom peut passer du temps à s'enregistrer mais la personne recherchant ensuite ce nom souhaite une réponse instantanée.

1.2.2.2 Confidentialité des ressources

De manière symétrique, la confidentialité des données dans une DHT peut-être assurée par chiffrement cryptographique. Une donnée qu'un pair A est autorisé à lire est chiffrée avec la clé publique de A ; A peut ensuite déchiffrer cette donnée avec sa clé privée. La confidentialité des données peut être utilisée dans les contextes suivants :

- dans le système de fichiers distribué FARSITE de Adya *et al* [ABC+02], les utilisateurs chiffrent les fichiers stockés avec une clé symétrique K. Pour accorder le droit de lecture à un autre utilisateur A, la clé de chiffrement K est chiffrée avec la clé publique de A. A peut alors déchiffrer la clé K avec sa clé privée puis déchiffrer le fichier original, comme illustré figure 1.10 ;
- dans [HJ03], Harrington et Jensen choisissent de chiffrer chaque fichier puis

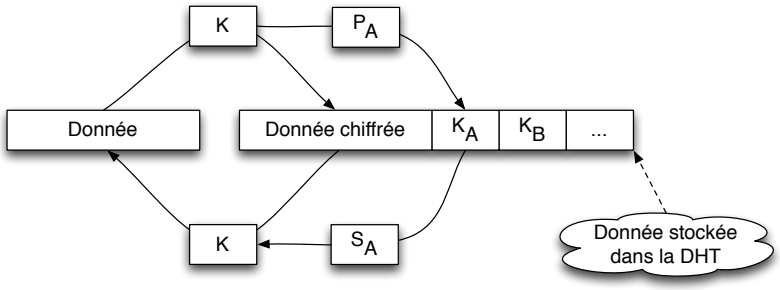


FIGURE 1.10 – Confidentialité d'une donnée dans FARSITE. La donnée, insérée par le pair A, est chiffrée avec une clé symétrique K. A ajoute à la donnée chiffrée la clé K elle-même chiffrée avec les clés publiques des pairs A, B, etc. autorisés à lire la donnée. Le pair A peut déchiffrer K avec sa clé secrète S_A puis déchiffrer la donnée avec la clé K.

de transmettre directement à chaque utilisateur autorisé la clé permettant de déchiffrer le fichier.

Dans les deux cas, la difficulté est de distribuer la clé de chiffrement utilisée. Dans le cas de FARSITE, le propriétaire du fichier doit obtenir de manière sécurisée la clé publique de l'utilisateur autorisé à accéder à ce fichier ; dans le cas de Harrington et Jensen, la personne autorisée doit pouvoir obtenir la clé du fichier à partir de son propriétaire.

1.2.2.3 Résumé et positionnement

Nous avons présenté dans cette sous-section les solutions pour garantir l'intégrité et la confidentialité des ressources, qui sont résumées tableau 1.3 :

- l'intégrité des données peut être assurée des trois façons suivantes :
  - la signature des données permet le nommage des ressources tout en offrant une garantie cryptographique. Le problème est cependant reporté sur la distribution des clés qui doit être sécurisée et, dans notre cas, reposer sur une architecture distribuée adaptée aux réseaux pair-à-pair ;
  - l'utilisation de chemins auto-signés permet le contrôle de l'intégrité de fichiers mais nécessite préalablement l'échange sécurisé d'un condensat SHA-1, ce qui est peu pratique (un condensat SHA-1 est représenté par 40 chiffres hexadécimaux) ;
  - la redondance des requêtes permet, en interrogeant plusieurs réplicats, d'augmenter la probabilité d'obtenir une réponse intègre. Cette méthode augmente cependant la latence des requêtes et ne protège pas un pair victime d'un routage malveillant, par exemple dans le cadre d'une attaque éclipse ;
- la confidentialité des données peut être assurée par chiffrement cryptographique. Dans ce cas, la donnée chiffrée est publique mais seuls les pairs possédant la clé de déchiffrement peuvent obtenir la donnée originale. Le problème est alors reporté sur la distribution des clés, tout comme dans le cas de la signature cryptographique.

Intégrité

Protection	Avantages	Inconvénients
Signature	+ Garantie cryptographique + Ressources nommées + Obtention rapide	– Distribution des clés (sécurisée ?)
Chemins auto-signés	+ Garantie cryptographique + Obtention rapide	– Ressources nommées par un condensat
Redondance	+ Ressources nommées	– Augmentation de la latence ressentie – Pas de garanties fortes

Confidentialité

Protection	Avantages	Inconvénients
Chiffrement	+ Garantie cryptographique + Ressources nommées + Obtention rapide	– Distribution des clés (sécurisée ?)

TABLE 1.3 – Solutions pour garantir l'intégrité et la confidentialité

La signature et le chiffrement permettent un système facilement utilisable grâce à des ressources nommées tout en permettant l'obtention rapide des données. La sécurité repose, dans ce cas, sur la distribution des clés cryptographiques. Dans le chapitre 7 de ce mémoire, nous présentons un système de distribution de clés cryptographiques. Une autorité de certification distribuée, gérée par les pairs eux-mêmes et présentée dans le chapitre 2, attribue des certificats nommés aux pairs. Cette distribution permet de mettre en place la signature et le chiffrement des données dans un contexte entièrement pair-à-pair.

1.3 Infrastructures à clé publique

Dans la section précédente, nous avons exprimé le besoin central de distribuer des clés cryptographiques de manière sécurisée afin de gérer les membres du réseau et d'assurer la confidentialité et l'intégrité des données stockées. La distribution des clés peut être réalisée suivant trois procédures :

- la distribution explicite, proposée notamment dans [HJ03, AEHF08, BPS05, BLJ05, RS07], consiste en une communication manuelle explicite de chaque clé nécessaire. Cette approche est insuffisante car elle pose des problèmes de passage à l'échelle et a un impact trop fort sur l'utilisateur. De plus, cette méthode ne permet pas l'obtention d'un système autonome et dépend de la sécurité du moyen de communication utilisé pour la transmission des clés ;
- le suivi des connexions, utilisé par exemple dans SSH [Ylö96], ZFone

[ZJC06] ou SOSIMPLE [BLJ05], permet de vérifier que des opérations successives ont bien été réalisées avec le même interlocuteur. La première obtention de la clé publique est réalisée de manière non sécurisée mais un condensat de cette clé est conservé. Lors des communications ultérieures, l'utilisateur peut vérifier que la clé publique n'a pas changé, ce qui garantit que, si la première clé était intègre, alors toutes les opérations ont été réalisées avec la bonne clé. Cette méthode ne permet, en revanche, pas de vérifier la clé publique lors de la première interaction ;
- une infrastructure à clé publique permet la distribution automatique des clés. Les deux possibilités sont les réseaux de confiance de type PGP [Zim95] et les autorités de certification.

Dans un réseau pair-à-pair où les interactions avec de nouveaux pairs sont nombreuses, la distribution des clés doit être automatique et transparente pour l'utilisateur. Nous éliminons donc la distribution manuelle via courrier électronique par exemple et le suivi des connexions qui ne permet par la vérification de la clé lors de la première connexion. La distribution automatique des clés peut être assurée par une infrastructure à clé publique. Nous détaillons dans la sous-section 1.3.1 les deux principales familles, à savoir les réseaux de confiance de type PGP et les autorités de certification ; nous expliquons pourquoi les autorités de certification nous paraissent plus adaptées à notre cadre des réseaux pair-à-pair structurés.

Cependant, les autorités de certification sont usuellement centralisées, ce qui n'est pas souhaitable en environnement pair-à-pair. Des mécanismes de certification distribuée ont été proposés précédemment, permettant de distribuer une autorité de certification parmi plusieurs entités. Nous présentons ces mécanismes dans la sous section 1.3.2.

1.3.1 Catégories d'infrastructures à clé publique

Nous présentons ici les deux catégories d'infrastructures à clé publique, à savoir les réseaux de confiance et les autorités de certification.

1.3.1.1 Réseaux de confiance

PGP [Zim95] utilise les réseaux de confiance pour sécuriser les échanges de courriers électroniques. Chaque personne accorde sa confiance dans l'identité d'autres personnes qu'il connaît directement ; ces personnes elles-mêmes accordent à leur tour leur confiance à d'autres. PGP repose ensuite sur la transitivité de la confiance pour trouver un *chemin de confiance* entre deux personnes : un *chemin de confiance* est une suite de liens de confiance formant un chemin dans le graphe. Afin de rester de confiance, un chemin a une longueur limitée et la transitivité de la confiance n'est accomplie que pour certains pairs de confiance supérieure. La propriété sous-jacente est que la population mondiale forme un graphe *petit-monde* [Mil67], dans lequel chaque couple de personnes est séparé par au plus six personnes (un lien étant un lien de confiance). La version déployée de PGP utilise des serveurs pour trouver les chemins de confiance, chemins de confiance qui sont ensuite vérifiés localement ; Capkun *et al.* dans [CBH03], Morselli *et al.* dans KeyChains [MBKM06] ainsi que Wölfl dans [Wöl05] proposent des équivalents entièrement distribués.

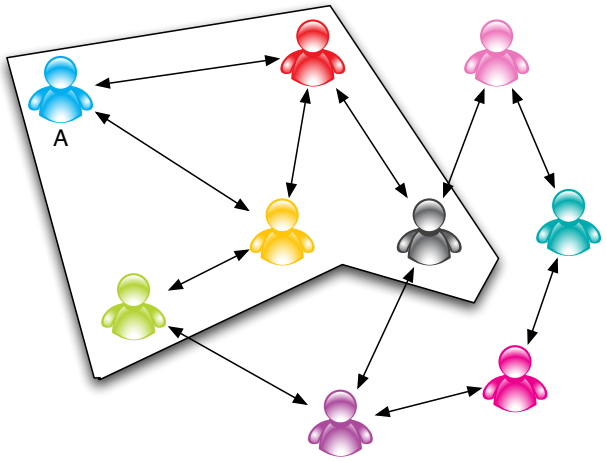


FIGURE 1.11 – Dans un réseau de confiance, chaque pair ne reconnaît pas le même ensemble de pairs. Ici, les chemins de confiance sont de longueur 2 et la partie du réseau reconnue par le pair A est encadrée.

Dans un réseau de confiance, chaque utilisateur a une vision locale du réseau de confiance et agit comme une mini autorité de certification, comme illustré figure 1.11. Il n'y a donc pas de cohérence globale, ce qui pose problème avec l'utilisation d'un réseau pair-à-pair structuré. En effet, dans ce cas, l'infrastructure à clé publique est utilisée pour contrôler l'appartenance des pairs au réseau (contrôle d'admission résistant à l'attaque sybile) ainsi que pour garantir l'intégrité et la confidentialité de ressources. Dans le cadre d'un réseau structuré, il est souhaitable que chaque pair considère les mêmes pairs comme faisant partie du réseau. Sans cette propriété, une ressource insérée sur un pair de confiance par son propriétaire peut être invisible à un autre utilisateur et les mécanismes de réplication ne sont pas adaptés à des pairs ayant une vue locale différente. De façon similaire, chaque pair doit reconnaître la même version d'une même donnée comme étant la version intègre afin de maintenir la cohérence du réseau, de la réplication et des caches. Dans le cas des réseaux de confiance, différents pairs peuvent reconnaître des ensembles de pairs de confiance différents et ainsi ne pas reconnaître la même version d'une donnée, ce qui brise la cohérence attendue. Enfin, un intérêt des réseaux structurés est leur rapidité, là où l'obtention d'une chaîne de confiance demande l'exécution de plusieurs requêtes successives, ce qui ne permet pas d'obtenir des latences basses.

Les réseaux de confiance ont une structure similaire aux réseaux non structurés et nous semblent plus adaptés à ces réseaux non structurés qu'aux réseaux structurés.

1.3.1.2 Autorités de certification

Les autorités de certification, telles que la Direction Générale des Impôts pour la télédéclaration des revenus, permettent de lier des capacités (permissions, possessions, etc.) à une clé publique au sein d'un certificat, certificat qui est signé avec la clé secrète de cette autorité. Le propriétaire du certificat peut

Infrastructure	Avantages	Inconvénients
Réseau de confiance	+ Déploiement simple + Contrôle distribué	– Pas de cohérence globale – Temps de calcul du chemin de confiance
Autorité de certification	+ Cohérence globale + Vérification rapide	– Centralisation – Déploiement complexe

TABLE 1.4 – Comparaison des infrastructures à clé publique.

ensuite prouver que ce certificat est le sien par la connaissance de la clé privée associée. Chaque utilisateur connaît la clé publique de l'autorité et peut vérifier puis accepter chaque certificat valide, ce qui garantit la cohérence globale. Les autorités de certification sont par exemple utilisées pour garantir des propriétés de sécurité dans les travaux suivants :

- dans [DR01], Druschel *et al.* utilisent une autorité de certification pour enrayer l'attaque sybile, solution qui est utilisée dans un grand nombre d'articles ;
- dans FARSITE [ABC+02], Adya *et al.* utilisent une autorité de certification pour nommer les utilisateurs et pouvoir ainsi déléguer des droits à d'autres utilisateurs nommés, comme dans un système de fichiers traditionnel ;
- dans [CMM02], Cox *et al.* proposent d'utiliser un réseau pair-à-pair pour servir des requêtes DNS qui sont donc de la forme `www.acme.org`. Pour garantir l'intégrité des réponses, cette approche est basée sur DNSSEC qui met en place une chaîne de certification depuis les autorités de nommage racine ;
- dans SOSIMPLE [BLJ05] de Brian *et al.* et P2P-IM [RS07] de Rao et Singhal, qui sont des systèmes de messagerie instantanée, les noms d'utilisateur peuvent être validés par une autorité de certification.

Cependant, ces autorités rompent le paradigme pair-à-pair en étant un point central de confiance et de faiblesse. Elles sont, de plus, complexes et coûteuses à administrer, comme nous l'avons expliqué dans la sous-section 1.2.1 : à de rares exceptions près dans des contextes spécifiques, les déploiements visant la certification de l'utilisateur final se sont soldés par des échecs. Les difficultés rencontrées par les autorités de certification sont développées par Davis dans [Dav96] ou par Ellison et Schneier dans [ES00].

1.3.1.3 Résumé et positionnement

Dans cette sous-section, nous avons présenté les réseaux de confiance et les autorités de certification, qui sont comparés tableau 1.4 :

- les réseaux de confiance offrent un déploiement simple et peuvent être utilisés de manière entièrement distribuée. Ils ne garantissent cependant pas la cohérence du réseau (chaque pair sélectionne localement les autres pairs qu'il accepte dans le réseau), ce qui peut interférer avec les mécanismes de réplication et de routage des réseaux pair-à-pair structurés.

Enfin, lorsqu'ils sont utilisés de manière entièrement distribuée, le calcul d'un chemin de confiance nécessite plusieurs requêtes et donc un temps non négligeable ;
- les autorités de certification assurent la cohérence de l'appartenance au réseau et permettent la vérification rapide des certificats générés. En revanche, elles imposent une centralisation, contraire au paradigme pair-à-pair, et leur déploiement est complexe et coûteux.

Dans ce mémoire, nous choisissons de distribuer une autorité de certification au sein du réseau pair-à-pair, ce qui permet de profiter de la cohérence de l'autorité de certification sans les inconvénients du contrôle centralisé. Le modèle et la mise en œuvre de cette autorité de certification distribuée sont présentés dans la première partie de ce mémoire ; deux exemples d'applications (protection contre l'attaque sybile et service de nommage sécurisé) sont proposés dans la seconde partie.

Le processus de certification se décompose en deux phases. L'autorité prend d'abord la décision d'accepter ou non la requête et ensuite, le cas échéant, cette autorité signe le certificat avec sa clé privée. Dans ce mémoire, nous nous attachons à distribuer ces deux phases dans un réseau pair-à-pair. Nous présentons ici les travaux antérieurs sur la signature distribuée, correspondant à la première partie de ce mémoire ; à notre connaissance, il n'existe pas d'étude des mécanismes de décision pouvant être associés à cette signature distribuée dans le cadre d'un réseau pair-à-pair.

1.3.2 Signature distribuée

Les algorithmes de signature distribuée utilisent de la *cryptographie à seuil*. La cryptographie à seuil permet de signer des certificats par la collaboration de plusieurs entités, tout en empêchant un petit groupe d'attaquants de signer de faux certificats. Nous présentons d'abord la cryptographie à seuil. Nous présentons ensuite son utilisation, d'abord à seuil fixe, puis à seuil variable.

1.3.2.1 Cryptographie à seuil

La cryptographie a seuil trouve ses fondements dans le secret partagé de Shamir [Sha79] et Blakley [Bla79]. Une valeur secrète, modélisée par un polynôme de degré t, est partagée entre n entités connaissant chacune un point de ce polynôme ; t entités peuvent alors collaborer pour retrouver le secret par interpolation de Lagrange. Desmedt [Des87] a proposé d'utiliser un principe similaire pour réaliser des signatures distribuées. Dans le cas d'une signature, une clé secrète est fragmentée en n fragments qui sont distribués à n entités différentes : t entités parmi les n doivent coopérer pour réaliser une signature, signature qui est réalisée sans révéler les secrets et sans reconstruire la clé secrète. Nous utilisons ici les travaux de Desmedt [Des97], Rabin [Rab98] et Shoup [Sho00] pour présenter la cryptographie à seuil et nous illustrons plus précisément nos propos dans les cas des chiffrements ElGamal/DSA et RSA.

Définition 1 (*Schéma de cryptographie à seuil* (t, n))

Un schéma de cryptographie à seuil (t, n) permet de signer un message en utilisant n'importe quels t fragments choisis parmi les n fragments générés initialement.

Les valeurs t et n sont, dans les propositions initiales, des constantes réglées à l'initialisation du système. Personne ne connaît la clé secrète complète, qui est ainsi mieux protégée que dans un système centralisé. La signature d'un message nécessite la collaboration de t fragments mais $t-1$ fragments n'apportent aucune information sur la clé secrète : un attaquant doit ainsi obtenir t fragments pour pouvoir calculer la clé secrète complète.

Pour obtenir la signature d'une donnée d, un utilisateur demande à t entités de signer d avec leur fragment ; la signature de d avec la clé secrète est ensuite une combination des t signatures partielles. Prenons l'exemple d'un chiffrement RSA t-parmi-t sans redondance tel que proposé dans [Boy89, Fra89]. La fonction f de signature RSA est homomorphique, c'est-à-dire que $f(x+y) = f(x) \times f(y)$. Si la somme des t fragments vaut l'exposant secret, alors la multiplication des t signatures partielles correspond à la signature avec la clé secrète. Seules les signatures partielles sont communiquées, les fragments de clé ne sont connus que par leur possesseur. La cryptographie à seuil peut ainsi être utilisée dans des systèmes de signature distribuée. Plusieurs nouvelles difficultés sont cependant posées par la distribution de la signature et nous détaillons les aspects qui nous préoccupent dans ce mémoire :

- l'**initialisation** du système est une phase critique. La distribution initiale des fragments requiert soit un tiers de confiance, soit un algorithme de génération distribuée. Dans le cas de l'utilisation d'un tiers de confiance, ce tiers connaît toute la clé privée et doit distribuer les fragments de manière sécurisée. Ce tiers est une faiblesse car la sécurité de tout le système dépend en fait de lui et chaque utilisateur doit donc lui faire confiance. La génération distribuée de la clé permet de s'affranchir de cette confiance ;
- la **vérification** des signatures partielles est nécessaire pour maintenir l'efficacité du schéma. Un attaquant peut en effet renvoyer une mauvaise valeur au lieu de la signature partielle demandée et ainsi corrompre le chiffrement final. La clé publique peut être utilisée pour vérifier la signature globale mais les signatures partielles ne peuvent pas être vérifiées sans mécanisme supplémentaire. Le cryptosystème doit donc permettre de vérifier la validité d'une signature partielle sans la connaissance du fragment de clé associé ;
- la **mise à jour** du seuil t et du nombre de participants n permet de s'accommoder d'un nombre de membres variable. Dans un réseau pair-à-pair, nous attendons d'un tel système qu'il s'adapte à la taille du réseau, qui peut varier de plusieurs ordres de magnitude, et que le nombre de pairs à convaincre pour obtenir un certificat reste une fraction non négligeable du réseau, quelle que soit sa taille. Le cryptosystème doit donc permettre de faire évoluer les valeurs t et n.

Plusieurs propositions ont été faites pour les deux principales familles d'algorithmes de signatures cryptographiques, à savoir ElGamal [ElG85] ou DSA d'un côté (dont la sécurité repose sur le problème du logarithme discret), RSA [RSA78] de l'autre (dont la sécurité repose sur le problème de la factorisation de grands nombres). Nous proposons ici un aperçu des propositions pour ces deux familles. Nous présentons brièvement le cryptosystème DSA et expliquons pourquoi il ne semble pas adapté à notre problématique. Nous décrivons ensuite plus précisément le cryptosystème RSA, que nous utilisons dans ce mémoire, et la distribution du schéma de signature RSA ainsi que ses limites.

ElGamal/DSA

DSA (*Digital Signature Algorithm*) est un algorithme asymétrique de signature proposé par le NIST en 1991 (RSA étant sous brevet à cette époque). Le système est tout d'abord défini par p, q et g qui peuvent être communs à tous les utilisateurs :

- p et q sont deux nombres premiers tels que $(p-1)$ est un multiple de q ;
- g est un nombre aléatoire dont l'ordre multiplicatif modulo p est q.

Les clés secrète et publique sont alors :

- la clé secrète est un nombre aléatoire x compris entre 0 et q ;
- la clé publique est (p, q, g, y) avec $y = g^x[p]$.

Définition 2 (*Signature DSA*)

La signature d'un message o est le couple :

$$(r = (g^k[p])[q], s = k^{-1}(h(o) + x \times r)[q])$$

avec k un nombre aléatoire compris entre 0 et q choisi par le signeur et conservé secret.

La signature est valide si $((g^{(h(o)\times s^{-1}[q])[q]} \times y^{(r\times s^{-1}[q])[q]})[p])[q] = r$.

L'algorithme de signature ElGamal est fonctionnellement proche, notamment l'utilisation d'une valeur aléatoire k secrète et différente pour chaque message.

Une version distribuée d'ElGamal a été proposée par Desmedt et Frankel [DF89]. Les nouvelles difficultés dues à la distribution de la signature ont été traitées de la manière suivante :

- **initialisation** : Pedersen [Ped91] et Gennaro *et al.* [GJKR99] ont proposé des algorithmes de génération distribuée de la clé ;
- **vérification** : Harn [Har94] a proposé une version distribuée du cryptosystème ElGamal permettant de vérifier les signatures partielles ainsi que Gennaro *et al.* [GJKR96b] pour le cryptosystème DSA ;
- **mise à jour** : Frankel *et al.* ont proposé dans [FGMY97a] un mécanisme permettant de faire varier le seuil t.

Cependant, tous ces cryptosystèmes sont interactifs et synchronisés : comme le note Shoup dans [Sho00], ils semblent condamnés à l'être. En effet, les schémas de signature ElGamal et DSA sont dits *probabilistes*, c'est-à-dire que l'entité réalisant la signature paramètre cette signature par une valeur aléatoire k : la signature est déterminée à la fois par le message à signer et par la valeur aléatoire k. Cette valeur doit être uniformément distribuée et ne doit pas être divulguée, au risque de compromettre la confidentialité de la clé secrète. Il n'est donc pas possible au demandeur de proposer une valeur aléatoire à utiliser et les entités collaborant à la signature doivent communiquer pour générer ensemble un nombre réellement aléatoire et non divulgué. Dans le cas d'applications à grande échelle comme dans un réseau pair-à-pair, ce type d'interaction entre les différentes entités collaborant à la signature pose *a priori* un problème de passage à l'échelle.

RSA

RSA est un algorithme asymétrique de chiffrement et de signature ; nous nous intéressons ici aux signatures. L'utilisateur génère les trois paramètres m, e et d tels que :

- $m = p \times q$ avec p et q premiers ;
- $e \times d \equiv 1\ [(p-1)(q-1)]$.

Les clés secrète et publique sont alors :

- la clé secrète (ou privée) est (e, m) ;
- la clé publique est (d, m).

Définition 3 (*Signature RSA*)

La signature d'un message o vaut :

$$s = h(o)^e[m]$$

où h est une fonction de padding *(hachage spécifique).*
La signature est valide si $h(o) = s^d[m]$.

Nous présentons plus précisément le standard RSA PKCS#1 v2.1 [RSA02] dans le chapitre 3, notamment les mécanismes de *padding* utilisés. Contrairement à DSA, RSA propose une signature déterministe qui ne dépend pas d'un élément aléatoire généré par l'entité réalisant la signature [13] : une signature RSA déterministe n'est déterminée que par le message à signer.

La première proposition d'un système à seuil pour RSA, avec $t \neq n$, a été exposée par Desmedt et Frankel [DF92] mais non prouvée ; ces mêmes auteurs ont ensuite proposé une évolution prouvée [FD92] dépendant d'interactions synchronisées. De Santis *et al.* [SDFY94] ont proposé une version prouvée, sans interactions, mais dans laquelle la taille des fragments croît linéairement avec n. Gennaro *et al.* [GJKR96a], Frankel *et al.* [FGMY97a, FGMY97b] et Rabin [Rab98] ont ensuite proposé des systèmes prouvés avec des fragments de petite taille.

Les nouvelles difficultés dues à la distribution de la signature ont été traitées de la manière suivante :

- **initialisation** : des algorithmes de génération de clé distribuée pour RSA ont d'abord été proposés par Boneh et Franklin [BF97] et Frankel *et al.* [FMY98]. La première proposition est plus rapide mais ne tolère que des attaquants passifs (honnêtes mais curieux) dans le groupe générant la clé. Algesheimer *et al.* [ACS02] ont également proposé un algorithme de génération distribuée où p et q sont des premiers forts [14] : cela permet d'exploiter certains protocoles de robustesse pour la cryptographie à seuil mais n'est pas nécessaire à la sécurité du cryptosystème RSA, comme l'ont démontré Rivest et Silverman [RS98]. La génération distribuée a été évaluée avec succès par Malkin *et al.* [MWB99] et Wright et Spalding [WS99] pour des petits groupes (au plus 5 participants). Nous utilisons

13. Une version probabiliste, non utilisée dans ce mémoire, existe également pour RSA [RSA02]. La PKCS #1 v2.1, dernière en date, recommande par précaution la migration progressive vers cette version probabiliste mais note qu'il n'existe aujourd'hui aucune attaque connue contre le schéma de signature RSA déterministe.

14. Un nombre premier p est fort si et seulement si $p = 2p' + 1$ avec p' premier.

dans ce mémoire le cryptosystème RSA et évaluons la génération distribuée dans des groupes plus grands dans le chapitre 2, section 2.4 ;

- **vérification** : Frankel *et al.* [FGY96] ont proposé de détecter les mauvaises signatures partielles. Pour chaque fragment, des valeurs *témoins* sont publiées ainsi que leur signature avec ce fragment (l'ensemble des témoins doit générer $\mathbb{Z}_m^*$). Le demandeur demande deux signatures, celui de la donnée à signer et une combinaison de la donnée à signer avec les témoins : le demandeur peut vérifier la cohérence des deux signatures mais le signeur ne peut pas tricher sur ce couple de réponses, ne connaissant pas la décomposition du deuxième élément. Gennaro *et al.* [GJKR96a] ont proposé un système proche quand le module m est le produit de deux nombres premiers forts où un seul témoin est nécessaire par fragment. Des variations ont été proposées par Frankel *et al.* [FGMY97a, FGMY97b] et Rabin [Rab98]. Dans tous les cas, la vérification repose sur la connaissance de valeurs associées à chaque fragment ;
- **mise à jour** : dans [FGMY97a], Frankel *et al.* proposent également un mécanisme pour faire varier le seuil t. Deux algorithmes sont proposés, le premier *Poly-to-Sum* permettant de passer d'un partage t-parmi-n à un partage additif t-parmi-t, le second *Sum-to-Poly* permettant de passer d'un partage t-parmi-t à un partage t'-parmi-n' : leur application successive permet donc de changer les valeurs t et n. Desmedt et Jajodia ont également proposé un système similaire dans [DJ97], précisé et étendu par Wong, Wang et Wing dans [WWW02]. Zhou *et al.* ont proposé APSS [ZSvR05], permettant également de faire évoluer les seuils, et ce pour la première fois dans un système supposé asynchrone ; les fragments de clé croissent cependant de manière exponentielle, ce qui limite cette approche à de faibles valeurs de t (moins de 3). Enfin, Schultz *et al.* [SLL08] (version étendue dans la thèse de Master [Sch07]) ont proposé une version plus efficace permettant d'utiliser un seuil t plus important, de l'ordre de 10. Ces opération ont toutes un coût exponentiel en fonction du nombre de fragments et requièrent des communications entre toutes les entités (broadcast), ce qui pose *a priori* un problème de passage à l'échelle pour de grands réseaux. Toutes les entités doivent également préalablement s'accorder sur les mêmes valeurs pour t et n.

Pour notre étude dans le cadre du pair-à-pair, nous avons choisi le chiffrement déterministe RSA afin de limiter les communications entre les participants.

1.3.2.2 Certification distribuée à seuil fixe

La cryptographie à seuil permet de distribuer le mécanisme de certification. Nous en présentons dans ce paragraphe quelques utilisations où le seuil t de pairs nécessaires pour signer un certificat est fixe tout au long de la vie du système.

Avec COCA (Cornell OnLine Certification Authority) [ZSvR02], Zhou *et al.* proposent une autorité de certification en ligne tolérante aux fautes car distribuée. Les auteurs partent du constat que les clés des autorités de certification sont le plus souvent opérées hors-ligne (isolées du réseau) pour mieux les protéger, ce qui complique la gestion et augmente les délais d'obtention d'un certificat. COCA distribue le processus de certification entre des entités contrôlées par l'autorité, t de ces entités devant coopérer pour signer un certificat. Comme $t-1$ fragments de la clé ne permettent pas de signer des certificats et n'apportent

aucune information sur la clé secrète, la clé de l'autorité est protégée tant que strictement moins de t de ses serveurs sont corrompus. De plus, la disponibilité du processus de certification est assurée tant que t serveurs de certification sont fonctionnels. La cryptographie à seuil permet un compromis intéressant entre la sécurité et l'utilisabilité [15].

Kong *et al.* proposent dans [KZL+01] un système de certification distribuée pour les réseaux ad hoc. Le seuil t est fixe et correspond au nombre de voisins immédiats attendus. Ces t membres peuvent soit créer un certificat pour un nouveau membre (ce qui lui donne un droit d'accès au réseau), soit renouveler un certificat. Luo *et al.* proposent avec URSA [LKZ+04] un système fonctionnellement proche mais utilisant un protocole de signature distribuée plus efficace. Jarecki *et al.* complètent URSA et résolvent une faiblesse dans [JSY04]. Cette approche à seuil fixe est assez naturelle pour les réseaux ad hoc : si le nombre de voisins de chaque membre reste fixe dans le temps, alors ce sont ces voisins qui vont router les messages de ce membre et ce sont donc eux qui gèrent son droit local d'accès au réseau.

Josephson *et al.* proposent avec CorSSO (Cornell Single Sign-On) [JSS04] un service d'authentification distribué pour des réseaux pair-à-pair. Les serveurs sont séparés entre serveurs d'application et serveurs d'authentification. Les serveurs d'authentification partagent des clés de chiffrement et les serveurs d'application peuvent exprimer des politiques de sécurité demandant l'authentification conjointe par plusieurs serveurs. Ce système permet de tolérer des serveurs d'authentification corrompus mais ne distribue pas le contrôle de la certification, ce rôle étant toujours réservé à un ensemble de serveurs spécifiques.

La certification distribuée à seuil fixe nous semble adaptée pour garantir un contrôle local, typiquement par les voisins d'un réseau ad hoc, mais ne nous paraît pas souhaitable pour distribuer le contrôle dans des réseaux pair-à-pair. En effet, dans le cas des réseaux pair-à-pair, les ressources sont globales et le contrôle de la certification doit donc également être réalisé globalement, par une fraction non négligeable et donc variable des pairs.

1.3.2.3 Certification distribuée à seuil variable

Dans [KMT03], Kim *et al.* proposent un contrôle d'admission des pairs utilisant de la cryptographie à seuil. L'admission d'un pair est matérialisée par l'obtention d'un certificat et le seuil t de pairs participant à cette certification peut être adapté en utilisant les algorithmes de Frankel *et al.* [FGMY97a]. Saxena *et al.* [STY03a, STY03b] proposent une étude plus approfondie dans le cas des réseaux pair-à-pair. Comme présenté dans la section 1.3.2.1, les algorithmes de changement de seuil de Frankel *et al.* nécessitent des communications en broadcast entre tous les membres participant, ce qui pose *a priori* un problème de passage à l'échelle dans un grand réseau pair-à-pair. De plus, leur utilisation nécessite de définir explicitement le seuil recherché et les travaux de Saxena *et al.* imposent ainsi un serveur responsable de compter les membres du réseau,

15. L'utilisabilité est définie par la norme ISO 9241-11 comme « le degré selon lequel un produit peut être utilisé, par des utilisateurs identifiés, pour atteindre des buts définis avec efficacité, efficience [16]et satisfaction, dans un contexte d'utilisation spécifié ».

16. L'efficience est définie par la norme ISO 9000 comme « le rapport entre le résultat obtenu et les ressources utilisées ».

centralisation qui n'est pas souhaitable dans un réseau pair-à-pair. Nous notons enfin que ces articles ne proposent pas de critères distribués sur lesquels chaque pair participant puisse s'appuyer pour décider de coopérer ou non à une opération de certification.

Avramidis *et al.* proposent également de gérer les membres par certification distribuée dans Chord-PKI [AKD07]. L'anneau de Chord est initialement décomposé en segments, chaque segment étant caractérisé par une paire de clés. Chaque clé publique est connue de tous et chaque clé privée est fragmentée entre les pairs dont les identifiants appartiennent au segment concerné : chaque segment est, en fait, une autorité autonome. Le nombre de segments étant défini lors de l'initialisation, les auteurs proposent d'utiliser les algorithmes de Desmedt et Jajodia [DJ97] pour augmenter les seuils quand le nombre de pairs dans le segment augmente. Cependant, ces algorithmes posent toujours la question du passage à l'échelle dans de grands groupes et la décomposition en un nombre fixe de segments repousse le problème de la croissance du réseau mais ne le résout pas. Les nouveaux pairs sont certifiés par leurs voisins appartenant au même segment après une période de probation, en utilisant un mécanisme de réputation.

Enfin, Bhattacharjee *et al.*, dans un travail qualifié de préliminaire [BRK07], proposent des recherches sécurisées. L'espace des identifiants est découpé en groupes d'un nombre de pairs constant et chaque groupe est dirigé par un comité composé *a priori* de 10 à 20 membres. Chaque comité possède l'unique clé secrète du réseau (c'est-à-dire que tous les comités connaissent la même clé secrète) et cette clé est fragmentée entre les membres d'un même comité : chaque comité peut générer des certificats signés par cette clé unique. Le comité d'un groupe est responsable de la surveillance des pairs rejoignant ou quittant ce groupe et émet un certificat contenant la liste des pairs présents dans ce groupe. Lors d'une opération de recherche sécurisée, ce certificat prouve les pairs existant autour de la ressource recherchée et permet donc de vérifier si l'opération a bien retourné la liste des voisins de l'identifiant recherché. Pour gérer la croissance du réseau et donc des groupes, les groupes peuvent se scinder, afin de limiter le nombre de pairs à surveiller par chaque comité. Deux comités sont alors formés à partir d'un, en utilisant [SLL08] pour redistribuer la clé secrète à ces deux comités. Cette opération de scission est présentée de manière informelle et les auteurs renvoient la suite à des travaux futurs ; à notre connaissance, cette suite n'a pas encore été publiée. Nous présentons ce système pour la certification distribuée qu'il propose mais nous notons qu'il ne s'agit pas réellement d'une infrastructure à clé publique. En effet, les signatures sont générées de manière locale pour prouver la topologie locale et il n'est pas question de distribution des clés, alors qu'une infrastructure à clé publique s'intéresse au système dans sa globalité et à la distribution des clés cryptographiques.

1.3.2.4 Résumé et positionnement

Dans cette sous-section, nous avons présenté la cryptographie à seuil ainsi que les mécanismes de certification distribuée utilisant cette cryptographie, d'abord à seuil fixe, puis à seuil variable.

La cryptographie à seuil permet de signer des messages par la coopération de t entités parmi n. Des deux algorithmes de signature cryptographique DSA et RSA, nous retenons RSA qui permet de réaliser des signatures sans commu-

nication en broadcast entre tous les pairs participant, broadcast qui n'est pas souhaitable dans un grand réseau pair-à-pair.

La certification distribuée à seuil fixe permet de signer un certificat par la coopération d'un nombre fixe d'entités. Dans le cadre d'un réseau pair-à-pair, la taille du réseau évolue de manière importante et, pour assurer un contrôle sûr, le nombre de pairs devant coopérer à une signature doit toujours représenter une fraction importante des pairs : le seuil de pairs devant coopérer doit varier en fonction de la taille du réseau.

Plusieurs systèmes de certification distribuée à seuil variable ont été proposés précédemment. Cependant, tous ces systèmes dépendent soit d'un serveur de comptage des pairs (centralisation contraire au paradigme pair-à-pair), soit de communications entre tous les pairs lors d'un changement de seuil (passage à l'échelle limité). Un système de certification distribuée à seuil variable pour des réseaux pair-à-pair doit être totalement distribué et limiter les communications.

Dans la première partie de ce mémoire, nous proposons un nouveau schéma de signature distribuée adapté à l'évolution des seuils dans un très grand réseau pair-à-pair structuré. Nous présentons la distribution des clés RSA, l'algorithme de signature distribuée et les protocoles de maintenance du seuil souhaité ; la maintenance du schéma ne dépend que d'interactions locales et est réalisée à coût faible et constant.

Résumé et objectifs de la thèse

Dans ce chapitre, nous avons présenté le contexte de notre étude.

Dans la section 1.1, nous avons présenté les trois grands types de réseaux pair-à-pair, à savoir les réseaux pair-à-pair avec centre, les réseaux pair-à-pair non structurés et les réseaux pair-à-pair structurés. Les réseaux structurés sont entièrement distribués et limitent la charge subie par chaque pair : nous avons choisi de nous intéresser à la sécurité de ces réseaux structurés, plus précisément dans le cadre des DHT.

Dans la section 1.2, nous avons présenté deux problèmes de sécurité dans les réseaux pair-à-pair structurés, l'attaque sybile, d'une part, ainsi que le contrôle de l'intégrité et de la confidentialité des données stockées, d'autre part :

- dans l'attaque sybile, un attaquant utilise un grand nombre d'identifiants ou certains identifiants spécifiques et obtient un rôle plus important que les autres utilisateurs : pour contrer cette attaque, le nombre d'identifiants de chaque utilisateur doit être limité et ces identifiants doivent être choisis aléatoirement. Nous avons présenté trois types de protection :
  - les autorités centralisées, complexes à administrer et contraires au paradigme pair-à-pair, permettent de limiter les deux aspects de l'attaque sybile ;
  - les tests de ressources informatiques, simples à déployer, posent le problème de la variété de puissance de calcul parmi les utilisateurs ;
  - les tests de ressources sociales permettent de limiter le nombre d'identifiants mais pas de contraindre ces identifiants à être choisis aléatoirement. Nous retenons le test de ressources sociales auquel nous rajoutons le contrôle des identifiants générés.
- le contrôle de l'intégrité des données peut être assuré par signature cryptographique, chemin auto-signé ou redondance. Le contrôle de la confiden-

tialité peut être assuré par chiffrement. Nous avons choisi la signature et le chiffrement, ce qui pose le problème de la distribution des clés.

Dans la section 1.3, nous avons présenté les infrastructures à clé publique permettant de distribuer des clés de chiffrement et avons retenu le système de l'autorité de certification pour les réseaux pair-à-pair structurés. Ces autorités sont usuellement centralisées mais peuvent être distribuées par l'utilisation de la cryptographie à seuil. Nous avons présenté les autorités de certification distribuées précédemment proposées et expliqué pourquoi aucune ne nous paraissait adaptée à la grande taille des réseaux pair-à-pair.

Dans ce mémoire, nous présentons une autorité de certification distribuée pour un réseau pair-à-pair structuré. L'autorité est représentée par un pourcentage fixe des membres (et donc un nombre de pairs variable) partageant une clé secrète et l'évolution du seuil est réalisée à coût faible et indépendant de la taille du réseau. Tous les pairs participent équitablement à cette autorité qui permet à la fois :

- de décider, de manière globale, de la légitimité d'une requête de certification ;
- de signer le certificat.

Les certificats sont ensuite stockés dans la DHT et sont disponibles pour tous les pairs.

Première partie

Construction de l'autorité de certification distribuée

Chapitre 2

Modèle de l'autorité de certification distribuée

« Democracy is the worst form of government,
except for all those other forms that have been tried from time to time. »
Winston Churchill

Dans ce mémoire, nous présentons une autorité de certification distribuée adaptée à des réseaux pair-à-pair structurés. Cette autorité repose sur la collaboration d'un nombre de pairs variable correspondant à un ratio fixe du nombre de pairs et utilise des protocoles passant à l'échelle, afin de s'adapter à des réseaux pair-à-pair de grande taille. Un certificat contient la clé publique de son propriétaire ainsi que les droits associés (droit d'accès, possession d'un nom, etc.) ; un certificat est signé avec la clé secrète de l'autorité de certification. Une autorité de certification est composée de deux parties :

- un mécanisme de décision, permettant de savoir si la requête de certification est légitime ;
- **un mécanisme de signature, permettant de matérialiser cryptographiquement l'accord de l'autorité de certification**.

Dans ce chapitre, nous nous intéressons à la distribution du mécanisme de signature. Nous présentons et évaluons le partage de la clé secrète et le modèle de la signature distribuée. La mise en œuvre effective de la signature distribuée est décrite dans le chapitre 3 et celle du partage de la clé secrète dans le chapitre 4 ; cette mise en œuvre est implémentée et évaluée dans le chapitre 5. Des mécanismes de décision sont enfin proposés dans la seconde partie de ce mémoire, composée des chapitres 6 (contre l'attaque sybile) et 7 (service de nommage).

Une certification distribuée relève de la coopération d'un ratio t des pairs connectés au réseau et chacun de ces pairs décide localement s'il participe ou non à cette instance de certification. Ces décisions peuvent, par exemple, être fondées sur des observations locales, des politiques de sécurité ou des preuves incluses dans la requête de certification. L'obtention d'un certificat requiert donc la coopération d'un ratio t fixé des membres du réseau qui signent conjointement le certificat. Ainsi, accepter un certificat comme valide est équivalent à croire qu'il n'existe pas assez d'attaquants en collusion pour représenter un ratio t

des pairs et qu'il n'a pas non plus été possible à l'attaquant de tromper un tel nombre de pairs honnêtes.

Définition 4 (*Signature à seuil de ratio t parmi n*)

Si n est la taille du réseau et t le ratio souhaité de membres devant coopérer à une signature, alors une signature à seuil de ratio t parmi n demande la coopération de $t \times n$ membres pour obtenir une signature valide.

Dans un réseau pair-à-pair :

- le ratio doit représenter une part non négligeable des pairs ;
- les opérations de signature et de maintenance du ratio doivent passer à l'échelle ;
- le schéma doit tolérer la présence d'attaquants internes en proportion inférieure au ratio t.

Dans la section 2.1, nous présentons le partage de la clé secrète du réseau. Afin de contraindre le ratio de pairs devant coopérer à une signature, la clé secrète de l'autorité est divisée en *fragments*. Le réseau pair-à-pair est décomposé en groupes et chaque groupe connaît un fragment de la clé secrète. Nous montrons que le ratio t de pairs devant coopérer est contraint en fixant la taille de ces groupes.

Dans la section 2.2, nous introduisons la signature distribuée. À partir du partage de la clé secrète de l'autorité, le processus de signature fait intervenir un membre de chaque groupe et n'est donc possible que si un membre de chaque groupe accepte de coopérer à cette certification. Dans le chapitre 3, nous proposons une mise en œuvre de la signature distribuée robuste aux attaquants et passant à l'échelle.

Dans la section 2.3, nous décrivons les opérations de maintenance du partage de la clé. Afin de contraindre de manière efficace le ratio de pairs devant coopérer, les opérations de maintenance permettent notamment de *diviser* ou *fusionner* des groupes quand la taille du réseau évolue, afin de garder des groupes de taille constante et donc un ratio t constant. Dans le chapitre 4, nous présentons une mise en œuvre de ces opérations à travers des algorithmes sans synchronisation et sans consensus byzantins.

Dans la section 2.4, nous étudions l'initialisation du système. En effet, même si une signature ne peut être obtenue que par la coopération d'un ratio fixé des membres, le partage de la clé est créé soit par un membre fondateur, qui connaît alors toute la clé secrète, soit par un consortium de membres fondateurs. L'initialisation est donc critique pour la sécurité du réseau et nous évaluons ces deux possibilités.

Dans la section 2.5, nous évaluons la sécurité du modèle de signature distribuée proposé. Nous étudions plus précisément comment un groupe d'attaquants peut obtenir la clé secrète du réseau ou détruire le partage de cette clé. Dans les deux cas, nous proposons une analyse probabiliste du nombre d'attaquants nécessaire à ces attaques. Dans le chapitre 5, nous renforçons ces calculs par des simulations et nous montrons que le système proposé est résistant à ces attaques jusqu'à environ 20% de pairs malveillants.

Dans la section 2.6, nous abordons les mécanismes de décision qui, combinés à cette signature, permettent de proposer une autorité de certification distribuée. Nous présentons une défense contre l'attaque sybile ainsi qu'un service de nommage sécurisé ; ces applications sont détaillées dans les chapitres 6 et 7.

Dans la suite de ce chapitre, nous faisons l'hypothèse que chaque pair est muni d'un certificat d'appartenance au réseau et qu'il n'y a pas d'attaque sybile, c'est-à-dire que chaque utilisateur ne possède qu'un pair et que son identifiant est choisi aléatoirement. Dans le chapitre 6, nous présentons un contrôle d'admission auto-entretenu résistant à l'attaque sybile. Ce contrôle d'admission rejette les pairs sybils et fournit aux pairs honnêtes un certificat d'appartenance au réseau en utilisant l'autorité de certification distribuée, opérée par les pairs déjà acceptés. Si le réseau est initialement constitué de pairs non sybils, alors les pairs qui opèrent l'autorité de certification distribuée sont non sybils et acceptent donc un nombre limité de pairs sybils.

2.1 Partage de la clé secrète

Dans cette section, nous présentons le partage de la clé secrète, partage qui garantit que le seuil dynamique effectif correspond bien à un ratio t fixé des n pairs connectés ($t \times n$ pairs doivent donc coopérer). Nous décrivons la décomposition de la clé secrète, la distribution des fragments de cette clé et l'affectation de ces fragments.

2.1.1 Décomposition de la clé secrète

La clé secrète est décomposée selon un schéma additif tel que proposé dans [Boy89, Fra89]. Le partage est basé sur la propriété d'homomorphie de la fonction de chiffrement RSA.

Propriété 1 (*Homomorphie de la fonction RSA*)

Si $S = (e, m)$ est la clé secrète RSA, nous pouvons choisir s entiers notés $e_1, \ldots, e_s$ tels que $e = \sum_{i=1}^{s} e_i$. Dans ce cas, la signature de toute donnée o vaut :

$$h(o)^e[m] = h(o)^{\sum_{i=1}^{s} e_i}[m] = \Big(\prod_{i=1}^{s} h(o)^{e_i}[m]\Big)[m]$$

En d'autres termes, la signature RSA de o avec S, qui est $h(o)^e[m]$, est égale au produit des signatures avec chaque entier e_i modulo m, m étant public car présent dans la clé publique $P = (d, m)$.

Dans le cas de la signature RSA, $h(o)$ est la sortie d'un *padding* préalable de la donnée o à signer et $h(o)$ est ensuite chiffré avec la clé secrète : ce chiffrement est la signature de la donnée initiale. Dans ce chapitre, le *padding* est représenté par une fonction $h()$ qui est détaillée dans la mise en œuvre de la signature distribuée (chapitre 3).

L'homomorphie de la fonction RSA permet un schéma de cryptographie à seuil additif dans lequel s *fragments* parmi s sont nécessaires.

Définition 5 (*Schéma additif s-parmi-s*)

Si $S = (e, m)$ est la clé secrète RSA, s entiers notés $e_1, \ldots, e_s$ peuvent être générés de la façon suivante :

- *$e_1, \ldots, e_{s-1}$ sont choisis aléatoirement ;*
- *$e_s = e - \sum_{i=1}^{s-1} e_i$*

La somme des entiers choisis vaut e et chacun d'eux est aléatoire et non devinable. Chaque entier e_i est appelé fragment de la clé secrète.

En utilisant cette décomposition de la clé secrète, la signature RSA de o avec S est égale au produit des signatures avec chaque fragment. Ce schéma permet donc de réaliser une signature par la coopération de s entités possédant chacune un fragment distinct.

Si s fragments sont distribués parmi $2s$ entités de sorte que chaque fragment soit connu par deux entités, alors la moitié des entités doit coopérer pour réaliser une signature. Nous utilisons ce type de réplication pour atteindre le ratio t souhaité : le réseau est divisé en groupes de pairs, chaque groupe connaissant un fragment de la clé.

2.1.2 Distribution des fragments

L'*overlay* est décomposé en autant de *groupes de fragmentation* qu'il y a de fragments de clé.

Définition 6 (*Groupe de fragmentation*)

Un groupe de fragmentation est un ensemble de pairs associés à un fragment de clé tel que :

- *un fragment distinct est affecté à chaque groupe et est répliqué sur tous les membres de ce groupe ;*
- *les groupes de fragmentation forment une* partition *de l'*overlay *(leur intersection est vide et leur union vaut l'*overlay*).*

Le groupe connaissant le fragment e_i est noté $\mathring{e}_i$ et est composé de g_i pairs.

Ainsi, puisqu'une signature est calculable en utilisant chaque fragment de la clé, l'obtention d'une signature valide requiert la coopération d'un membre de chaque groupe de fragmentation.

À partir de cette décomposition du réseau en groupes de fragmentation, nous pouvons obtenir les relations suivantes entre le ratio t de pairs devant coopérer, le nombre n de pairs, la taille g_i des groupes de fragmentation et le nombre s de fragments (un fragment par groupe) :

- $s = t \times n$: le nombre de fragments correspond au nombre de pairs devant coopérer ;
- $n = \sum_{i=1}^{s} g_i$: le nombre total de pairs correspond à la somme des tailles des groupes ;
- $t = \frac{s}{n} = \frac{s}{\sum_{i=1}^{s} g_i} = \frac{s}{s \times g} = \frac{1}{g}$, avec g la moyenne des g_i : le ratio t de pairs devant coopérer est l'inverse de la taille moyenne g des groupes.

Propriété 2

Si t est le ratio souhaité et g la taille moyenne des groupes, l'égalité $t = \frac{1}{g}$ implique qu'en fixant la taille des groupes et donc leur moyenne g, le ratio t de pairs devant coopérer est également fixé, quelle que soit la taille du réseau n.

Le ratio t de pairs devant coopérer est ainsi fixé en contraignant la taille des groupes de fragmentation. Lorsque le réseau grandit, la taille des groupes étant limitée, de nouveaux groupes sont créés et de nouveaux fragments leur sont attribués. Cette opération doit être réalisée sans centre et sans algorithme coûteux. Par exemple, il n'est pas envisageable de redécouper le réseau entier et de générer de nouveaux fragments pour tous les groupes.

Afin de créer un nouveau groupe et un nouveau fragment, un groupe existant est *divisé* en deux ainsi que son fragment. Comme la seule contrainte sur les fragments est que leur somme vaut e, un groupe connaissant un fragment e_i peut localement séparer son fragment en deux fragments e_j et e_k tels que $e_i = e_j + e_k$. Cette opération permet à un groupe unique de générer deux groupes. Dans ce cas, la taille des nouveaux groupes vaut $\frac{g_i}{2}$.

Afin de permettre ces divisions, la taille des groupes est fixée entre deux bornes g_{min} et $g_{max} = 2 \times g_{min}$. Chaque groupe de fragmentation peut donc décider de se *diviser* à partir d'observations uniquement locales (taille du groupe de fragmentation) et ainsi contraindre le ratio t de pairs devant coopérer. Aucune communication globale n'est nécessaire et la taille du réseau n'est pas calculée pour maintenir ce ratio t.

Propriété 3

Quelle que soit la taille du réseau n, si les groupes de fragmentation ont une taille comprise entre g_{min} et g_{max}, alors l'inégalité suivante est vérifiée et contraint le ratio t de pairs devant coopérer à une signature :

$$\frac{1}{g_{max}} \leq t \leq \frac{1}{g_{min}}$$

En effet, $t = \frac{1}{g}$ et $g_{min} < g < g_{max}$.

Sur la figure 2.1, qui nous sert d'exemple tout au long de ce mémoire, les groupes sont composés de $g_{min} = 3$ à $g_{max} = 6$ membres, ce qui contraint le ratio t entre $\frac{1}{3}$ et $\frac{1}{6}$.

Le choix des valeurs g_{min} et g_{max} reflète un compromis entre la sécurité d'une part et la charge et la disponibilité d'autre part. Plus ces valeurs sont petites, plus le nombre de pairs devant coopérer est important et plus les attaquants doivent être nombreux pour signer de faux certificats. En revanche, plus ces valeurs sont grandes et moins les pairs sont sollicités pour participer à des certifications. De plus, ces valeurs doivent être suffisamment élevées pour garantir la disponibilité de tous les fragments à chaque instant. Nous utilisons dans ce mémoire :

- $g_{min} = 20$ et $g_{max} = 2 \times g_{min} = 40$ pour les évaluations du système proposé. Même si ces valeurs permettent en théorie à $2,5\%$ d'attaquants de forger des certificats, l'étude présentée dans la section 2.5 et évaluée dans le chapitre 5 montre que le système résiste à environ 20% d'attaquants ;
- $g_{min} = 3$ et $g_{max} = 2 \times g_{min} = 6$ pour les illustrations du système (par

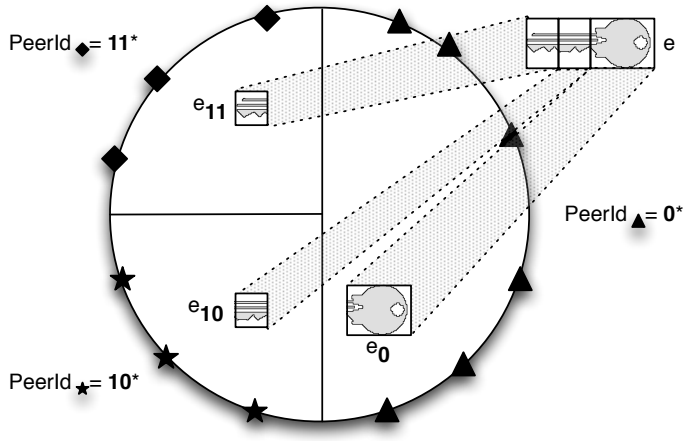


FIGURE 2.1 – Distribution de trois fragments e_0,e_{10} et e_{11} avec $e = e_0 + e_{10} + e_{11}$.

exemple les figures présentées).

Afin de permettre, d'une part, de pouvoir localiser les fragments nécessaires à l'obtention d'une signature et, d'autre part, à chaque pair de surveiller les autres membres de son groupe de fragmentation, chaque fragment est identifié de manière unique et est affecté à un ensemble déterminé de pairs.

2.1.3 Affectation des fragments

Chaque fragment e_i est identifié de manière unique par un identifiant binaire. Le réseau est initialement composé d'un seul groupe de fragmentation connaissant e dont l'identifiant est vide. Lorsque ce groupe initial se divise, deux fragments d'identifiants 0 et 1 sont créés : ces fragments sont notés e_0 et e_1. Les pairs appartenant à ce groupe initial de fragmentation se répartissent entre les deux groupes ainsi créés en fonction de leur identifiant de pair : les pairs dont l'identifiant commence par 0 vont dans $\mathring{e_0}$ et ceux dont l'identifiant commence par 1 vont dans $\mathring{e_1}$. Ces groupes se divisent ensuite en $\mathring{e_{00}}$, $\mathring{e_{01}}$, $\mathring{e_{10}}$ et $\mathring{e_{11}}$. À chaque instant, un fragment e_i n'est connu que par les pairs dont l'identifiant binaire commence par i.

Sur la figure 2.1, il y a trois fragments e_0, e_{10} et e_{11}. Les pairs dont l'identifiant commence par 0 connaissent e_0, ceux par 10 connaissent e_{10} et ceux par 11 connaissent e_{11}.

Définition 7 (*Partage de la clé*)

Nous définissons le partage de la clé *comme la répartition des fragments sur les différents pairs à un instant donné. Ce partage est noté KS et contient une liste d'éléments $(e_i, \mathring{e_i})$ où e_i est un fragment de clé et $\mathring{e_i}$ est le groupe des pairs connaissant e_i. Cette liste est munie des deux opérations suivantes :*

- $\ominus$ $(e_i, \mathring{e_i})$ *retire de KS le groupe $\mathring{e_i}$ connaissant e_i*
- $\oplus$ $(e_i, \mathring{e_i})$ *ajoute à KS le groupe $\mathring{e_i}$ connaissant e_i*

Le partage de la clé KS vérifie les propriétés suivantes :

- les groupes sont constitués de g_{min} à g_{max} membres
- l'ensemble des groupes forme une partition de l'espace de l'*overlay*
- la somme de tous les fragments vaut e

Les notations utilisées sont résumées figure 2.2.

- $(P, S) = ((d, m), (e, m))$: paire de clés du réseau
- n : nombre de pairs du réseau
- t : ratio des pairs devant coopérer pour obtenir une signature
- g_{min} et g_{max} : tailles minimum et maximum des groupes
- e_i : valeur numérique du fragment de clé e_i
- $\mathring{e}_i$: groupe de fragmentation connaissant e_i

FIGURE 2.2 – Résumé des notations utilisées

2.2 Obtention d'une signature

Pour obtenir une signature avec la clé secrète S, un membre A doit obtenir et multiplier les chiffrements partiels réalisés avec chacun des fragments. Dans le cas de la figure 2.1, ces fragments sont e_0, e_{10} et e_{11} et la signature nécessite donc la coopération de trois pairs dont les identifiants commencent respectivement par 0, 10 et 11.

Définition 8 (*Signature RSA distribuée*)

Si E représente l'ensemble des s fragments de clé distribués dans le réseau, la signature RSA distribuée d'une donnée o vaut :

$$\Big(\prod_{e_i \in E} h(o)^{e_i}[m] \Big)[m]$$

chaque $h(o)^{e_i}[m]$ étant calculé dans un groupe de fragmentation différent.

Une requête de certification *Req* est composée des deux paramètres *Content* et *Clues* :

- *Content* contient les données du certificat à signer ;
- *Clues* contient d'éventuelles données supplémentaires pour aider chaque pair à faire son choix de coopération.

Le pair demandeur a besoin de la coopération et de l'accord d'un pair de chaque groupe de fragmentation pour obtenir la signature de son certificat qui est $h(Content)^e[m]$.

Le pair demandeur A pourrait directement demander une signature partielle à un pair de chaque groupe. Dans ce cas, A doit réaliser un routage vers chaque groupe, communiquer sa requête dans chaque groupe, recevoir la signature partielle et enfin combiner toutes les signatures partielles en les multipliant. Cependant, le nombre de groupes augmente linéairement avec la taille du réseau et la charge sur A pourrait devenir prohibitive. Nous proposons dans le chapitre 3 un algorithme de signature distribuée arborescent permettant de maintenir une charge équilibrée dans le réseau tout en tolérant la présence d'attaquants.

Chaque pair décide de sa coopération en utilisant un mécanisme de décision, tel que présenté dans la section 2.6 et développé dans la seconde partie de

ce mémoire. Si le demandeur A ne parvient pas à trouver un pair de chaque groupe qui accepte de coopérer, alors A n'obtient pas de certificat. Ainsi, chaque certificat obtenu prouve qu'un pair de chaque groupe de fragmentation, et donc un ratio t des pairs, a jugé la requête légitime.

2.3 Évolution du réseau et maintenance du partage

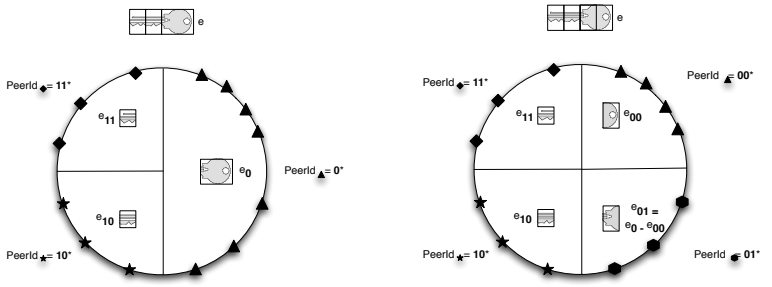
Lorsque la taille du réseau augmente ou diminue, le nombre de groupes de fragmentation doit varier, puisque la taille des groupes est fixée. Cinq opérations de maintenance – *division, rafraîchissement, fusion, connexion* et *déconnexion* – permettent de faire évoluer le partage de la clé. Les opérations présentées sont locales à un ou deux groupes de fragmentation et ne nécessitent donc aucune communication globale pour maintenir le seuil t.

Nous présentons ici la sémantique de ces opérations vis-à-vis du partage de la clé ; leur mise en œuvre sans consensus et sans synchronisation est proposée dans le chapitre 4.

2.3.1 Division d'un groupe

Quand de nouveaux pairs rejoignent le réseau, le nombre de groupes de fragmentation doit augmenter. Afin de permettre le passage à l'échelle, la création d'un groupe ne modifie pas l'intégralité du partage de la clé mais un nouveau groupe est créé par la division d'un groupe existant.

Chaque pair maintient la liste des membres de son groupe. Quand un groupe détecte qu'il contient plus de g_{max} membres, il se *divise* en deux groupes et génère deux nouveaux fragments. Pour créer deux fragments e_{i0} et e_{i1} à partir de e_i, e_{i0} et e_{i1} doivent simplement être choisis tels que $e_{i0} + e_{i1} = e_i$ et e_i doit être effacé et non recalculable. Afin d'empêcher e_i d'être calculable, e_{i0} est choisi aléatoirement et e_{i1} vaut $e_i - e_{i0}$. Après cette opération, un fragment supplémentaire a été créé et la somme des fragments vaut toujours e. En fonction de leur identifiant, les pairs migrent vers e°_{i0} ou e°_{i1}, comme illustré figure 2.3.



(a) Avant la division, le groupe e°_0 est constitué de plus de $g_{max} = 6$ membres.

(b) Après la division, deux groupes e°_{00} et e°_{01} sont créés à partir du groupe e°_0.

FIGURE 2.3 – Division du groupe e°_0.

Définition 9 (*Division*)

Après une division du groupe $\mathring{e_i}$, le partage de la clé KS vaut

$$KS \ominus (e_i, \mathring{e_i}) \oplus (e_{i0}, \mathring{e_{i0}}) \oplus (e_i - e_{i0}, \mathring{e_{i1}})$$

Le partage contient un groupe de plus et la somme des fragments reste égale à e.

2.3.2 Rafraîchissement de deux fragments

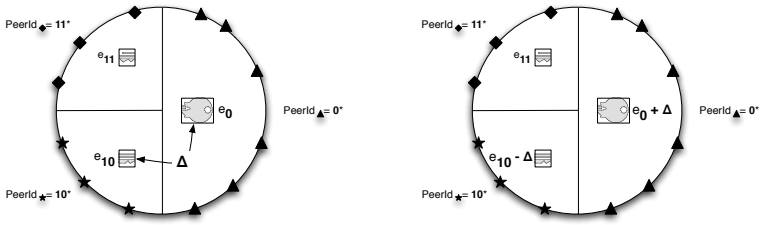
Après une division, les pairs qui connaissaient e_i connaissent à la fois e_{i0} et e_{i1}. En effet, e_{i0} est choisi aléatoirement et communiqué à tous les pairs du groupe, soit pour choisir ce fragment, soit pour calculer e_{i1}. De plus, un attaquant peut conserver e_i et posséder ainsi un fragment supérieur à ce qu'il devrait connaître. Une solution est de *rafraîchir* e_{i0} et e_{i1}, opération qui consiste à modifier ces fragments. Pour rafraîchir e_{i0}, un autre fragment e_x est choisi aléatoirement et une valeur aléatoire Δ est ajoutée à e_{i0} et soustraite de e_x. La somme des fragments reste e mais e_{i0} change et e_i n'est plus compatible avec le partage de la clé secrète. Le rafraîchissement est illustré figure 2.4.

Définition 10 (*Rafraîchissement*)

Après un rafraîchissement entre les groupes $\mathring{e_x}$ et $\mathring{e_y}$, le partage de la clé KS vaut

$$KS \ominus (e_x, \mathring{e_x}) \ominus (e_y, \mathring{e_y}) \oplus (e_x + \Delta, \mathring{e_x}) \oplus (e_y - \Delta, \mathring{e_y})$$

Deux fragments sont modifiés et la somme des fragments reste égale à e.



(a) Pour rafraîchir, $\mathring{e_0}$ et $\mathring{e_{10}}$ échangent une valeur aléatoire Δ.

(b) Après le rafraîchissement, les deux fragments e_0 et e_{10} ont changé et la somme des fragments vaut toujours e.

FIGURE 2.4 – Rafraîchissement entre les groupe $\mathring{e_0}$ et $\mathring{e_{10}}$.

2.3.3 Fusion de deux groupes

Quand des pairs quittent le réseau, si un groupe est constitué de moins de g_{min} membres, il *fusionne* avec son groupe frère afin de garantir la disponibilité du fragment possédé. Les deux fragments e_{i0} et e_{i1} sont ajoutés pour obtenir e_i et les deux groupes fusionnent leur liste des membres. Le nombre de fragments de clé dans le réseau est diminué de 1 et la somme de ces fragments vaut toujours

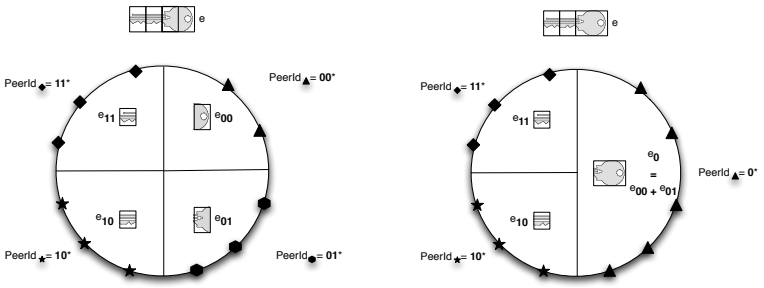
e. Cette opération permet de garantir la disponibilité de tous les fragments. La fusion est l'opération réciproque de la division et est illustrée figure 2.5.

Définition 11 (*Fusion*)

Après une fusion entre les groupes e°_{i0} et e°_{i1}, le partage de la clé KS vaut

$$KS \ominus (e_{i0}, e^{\circ}_{i0}) \ominus (e_{i1}, e^{\circ}_{i1}) \oplus (e_{i0} + e_{i1}, \mathring{e}_i)$$

Le partage contient un groupe de moins et la somme des fragments reste égale à e.



(a) Avant la fusion, le groupe e°_{00} est constitué de moins de $g_{min} = 3$ membres.

(b) Après la fusion, un groupe $\mathring{e}_0$ est créé à partir des deux groupes e°_{00} et e°_{01}.

FIGURE 2.5 – Fusion des groupe e°_{00} et e°_{01}.

2.3.4 Connexion d'un pair

Quand un pair se connecte au réseau, il rejoint le groupe de fragmentation contenant son identifiant de pair. Ce pair obtient le fragment du groupe ainsi que la liste des membres et signale son arrivée aux autres membres. L'arrivée de ce nouveau pair peut provoquer une opération de division si le groupe devient constitué de plus de g_{max} membres.

Définition 12 (*Connexion*)

Après la connexion d'un nouveau pair A qui rejoint le groupe $\mathring{e}_i$, le partage de la clé KS vaut

$$KS \ominus (e_i, \mathring{e}_i) \oplus (e_i, \mathring{e}_i \cup \{A\})$$

Les fragments ne changent pas et le groupe $\mathring{e}_i$ contient un membre de plus.

2.3.5 Déconnexion d'un pair

Quand un pair se déconnecte du réseau, les autres membres de son groupe constatent qu'il ne répond plus (grâce à un mécanisme de surveillance) et le retirent de la liste des membres. Le départ de ce pair peut provoquer une opération de fusion si le groupe devient constitué de moins de g_{min} membres.

Définition 13 (*Déconnexion*)

Après la déconnexion d'un pair A du groupe $\mathring{e}_i$, le partage de la clé KS vaut

$$KS \ominus (e_i, \mathring{e}_i) \oplus (e_i, \mathring{e}_i \setminus \{A\})$$

Les fragments ne changent pas et le groupe $\mathring{e}_i$ contient un membre de moins.

Ces cinq opérations permettent de maintenir le partage de la clé et leur mise en œuvre est détaillée dans le chapitre 4. Nous nous intéressons maintenant à la génération initiale de la clé.

2.4 Génération de la clé du réseau

Le réseau étant caractérisé par sa paire de clés RSA, la sécurité repose sur le partage de la clé privée entre les membres du réseau. Il est donc nécessaire d'étudier la génération initiale de cette clé privée et les garanties apportées. Cette clé peut soit être générée de manière centralisée puis partagée, soit directement générée de manière distribuée.

2.4.1 Une entité unique génère la clé du réseau

La clé de réseau peut tout d'abord être générée par une entité unique. Par exemple, une institution propose une application pair-à-pair et souhaite que le réseau propose des mécanismes de sécurité tout en restant entièrement autonome.

L'institution ayant généré la clé peut dans ce cas forger de faux certificats si elle conserve la clé secrète. Cependant, même si l'institution fondatrice est fermée, le réseau autonome peut toujours assurer sa fonction d'autorité de certification distribuée et l'application pair-à-pair n'est donc pas atteinte par la coupure d'un serveur central. De plus, si l'institution efface la clé privée après l'avoir partagée, alors corrompre les machines de l'institution ne permet ni de stopper le réseau pair-à-pair formé ni d'obtenir la clé secrète.

Cette génération centralisée garantit donc la disponibilité de l'autorité de certification distribuée mais son intégrité dépend du comportement de l'institution fondatrice. Pour l'institution fondatrice, cette approche a également l'intérêt de réduire la charge qu'elle doit supporter puisqu'elle n'intervient plus dans le processus de certification.

2.4.2 Un groupe génère la clé du réseau

La clé du réseau peut également être générée par un groupe d'individus en utilisant un algorithme de génération distribuée, tel que proposé dans [BF97, ACS02]. Un ensemble de membres fondateurs coopèrent durant cette génération et chacun obtient un fragment de la clé privée ainsi que la clé publique. Personne ne connaît alors la clé privée en entier.

Certains membres fondateurs peuvent être malveillants : le but de la certification distribuée est justement de tolérer ces attaquants internes. Comme

tous les fragments de la clé sont nécessaires pour signer des certificats, la clé secrète est distribuée de manière sécurisée aussitôt qu'il manque un fragment aux attaquants, condition qui est remplie si au moins un membre fondateur est honnête. Après cette distribution initiale, les fragments sont ceux utilisés par la certification distribuée et sont donc rafraîchis, ce qui rend les fragments initiaux obsolètes. Un unique membre fondateur honnête garantit l'intégrité de la certification distribuée.

Une fois initialisée, l'autorité de certification distribuée bénéficie de la robustesse des réseaux pair-à-pair et reste effective tant que le réseau sous-jacent fonctionne ; elle ne peut notamment pas être stoppée par une décision localisée. De plus, même si un attaquant corrompt tous les membres fondateurs après que le réseau a grandi, cet attaquant ne peut pas obtenir la clé privée : en effet, lors de la croissance du réseau, les fragments sont divisés et rafraîchis et les membres fondateurs honnêtes suppriment leur fragment initial.

Nous avons implémenté et évalué l'algorithme de génération distribuée de Boneh et Franklin [BF97]. Ce travail a été réalisé dans le cadre du stage de Master 1 de Thierry Congos, encadré par l'auteur de ce mémoire. En utilisant la plate-forme de test *PlanetLab*[1], nous avons généré des clés RSA de 1024 bits dans des groupes de 37 membres en quelques heures, ce qui montre la faisabilité de l'initialisation distribuée. Ce résultat a été publié à STM 2009 [CL09].

Nous étudions maintenant la composition du groupe fondateur qui peut soit être un consortium bien connu soit un ensemble de membres fondateurs anonymes.

2.4.2.1 Le groupe est un consortium bien connu

La certification distribuée peut être initialisée par un consortium bien connu de membres fondateurs. Ces membres fondateurs doivent avoir la réputation de vouloir protéger le réseau car les futurs membres doivent être persuadés que ces fondateurs ont agi dans l'intérêt du réseau. Les membres fondateurs peuvent être les promoteurs d'une nouvelle technologie pair-à-pair qui souhaitent apporter de la sécurité dans leur réseau ou des générateurs de contenu utilisant le réseau pair-à-pair pour transmettre ces contenus.

Le consortium devrait être composé de membres de différentes institutions séparées géographiquement afin de réduire les risques de coalition malveillante.

2.4.2.2 Le groupe est un ensemble anonyme de pairs initiaux

La certification distribuée peut également être initialisée par un groupe de membres initiaux anonymes. Comparativement au consortium clairement défini, ces membres anonymes peuvent être plus nombreux et il y a donc moins de confiance à placer en chacun d'eux. Comme un seul fondateur honnête suffit, il faut un grand nombre d'attaquants pour corrompre l'initialisation. De plus, ces pairs initiaux sont exécutés par des personnes ordinaires, tout comme les futurs pairs.

Dans ce cas, le réseau pair-à-pair est initialement protégé par un mécanisme adapté aux petits réseaux, par exemple du *friend-to-friend* (F2F). En F2F, un nouveau membre doit faire enregistrer sa clé publique explicitement par tous les

1. `http://planet-lab.eu`

Génération centralisée :

Avantages	Inconvénients
+ La génération de la clé est simple + L'autorité a une disponibilité élevée + La charge sur le fondateur est réduite	– Le fondateur peut générer de faux certificats – La confiance repose totalement sur le fondateur

Génération distribuée :

Avantages	Inconvénients
+ Personne ne connaît la clé secrète + L'autorité est intègre avec une forte probabilité + L'autorité a une disponibilité élevée	– La génération de la clé est complexe – La définition du groupe fondateur doit être consensuelle

TABLE 2.1 – Résumé des générations centralisée et distribuée.

membres actuels du réseau. Cela permet de créer un petit réseau (une cinquantaine de membres) dans lequel les membres sont ajoutés soit par connaissance antérieure, soit par recommandation directe. Ce mécanisme ne passe cependant pas à l'échelle et les membres peuvent donc initialiser une autorité de certification distribuée dès que le réseau atteint quelques dizaines de membres.

L'analyse des générations centralisée et distribuée est résumée tableau 2.1.

2.5 Analyse de la sécurité du modèle

Dans cette section, nous discutons la sécurité du modèle *après initialisation* (la clé secrète est déjà partagée) et en présence d'attaquants. Les attaquants peuvent soit tenter d'obtenir la clé secrète du réseau, soit tenter de rendre indisponible le système de signature distribuée.

2.5.1 Obtention de la clé secrète du réseau

Pour obtenir la clé secrète du réseau, un attaquant doit obtenir tous les fragments de la clé. En dehors d'une attaque sur le logiciel utilisé, l'attaquant doit placer un pair dans chaque groupe pour obtenir tous les fragments. En l'absence d'attaque sybile (nous avons fait cette hypothèse dans ce chapitre et nous présentons ensuite une protection contre cette attaque), nous étudions combien de pairs attaquants doivent s'unir pour être placés au moins une fois dans chaque groupe.

Soit k le ratio d'attaquants. Si un groupe $\mathring{e_i}$ est composé de g_i membres, alors la probabilité qu'il n'y ait aucun attaquant dans ce groupe est $(1-k)^{g_i}$. La probabilité qu'il y ait au moins un attaquant est donc $1-(1-k)^{g_i}$.

Considérons maintenant un réseau composé de s groupes de fragmentation composés respectivement de $g_1, \ldots, g_s$ membres (la taille du réseau est donc

$\sum_{i=1}^{s} g_i$). Alors la probabilité qu'il y ait un attaquant dans chaque groupe vaut :

$$\boxed{P_{revealed} = \prod_{i=1}^{s} 1 - (1-k)^{g_i}}$$

Étant donné que $1 - (1-k)^{g_i}$ est inférieur à 1, cette probabilité décroît quand le nombre de fragments s augmente, c'est-à-dire quand le réseau grandit. De plus, pour une taille de réseau donnée, réduire la taille des groupes g_i (et donc augmenter le nombre de groupes s) diminue cette probabilité. Cette probabilité est évaluée théoriquement et expérimentalement dans le chapitre 5. Avec des groupes constitués de $g_{min} = 20$ à $g_{max} = 40$ membres, il faut au moins 20% d'attaquants pour mener à bien cette attaque.

2.5.2 Destruction du partage de la clé

Pour détruire le partage de la clé secrète du réseau, un groupe d'attaquants doit détruire un fragment de cette clé.

Pour cela, les attaquants peuvent tout d'abord attaquer tous les représentants d'un groupe et les rendre indisponibles (attaque de déni de service). Pour mener à bien cette attaque, tous les pairs du groupe visé doivent être attaqués simultanément, sinon les pairs restants ont le temps de fusionner avec le groupe voisin. Nous jugeons cette attaque contre tous les membres d'un groupe difficile à mettre en place et nous ne la traitons pas directement dans ce mémoire. Nous proposons tout de même dans les travaux futurs d'augmenter la taille des groupes avec la taille du réseau ; dans ce cas, plus le réseau est grand (et donc plus l'intérêt est grand pour des attaquants de réaliser une telle attaque de grande envergure), plus il faut attaquer de pairs, ce qui rend cette attaque encore plus difficile.

Les attaquants peuvent également posséder tous les pairs appartenant à un groupe de fragmentation quelconque et ainsi détruire le fragment de clé associé à ce groupe. Nous calculons ici la probabilité pour un groupe d'attaquants de contrôler un groupe de fragmentation entier.

Soit k le ratio d'attaquants. Si un groupe donné est composé de g_i membres, alors la probabilité qu'il n'y ait que des attaquants dans ce groupe vaut k^{g_i}. La probabilité qu'il y ait au moins un pair honnête vaut donc $1 - k^{g_i}$.

Considérons maintenant un réseau composé de s groupes de fragmentation composés respectivement de $g_1, \ldots, g_s$ membres. La probabilité qu'il y ait au moins un pair honnête dans chaque groupe vaut $\prod_{i=1}^{s} 1 - k^{g_i}$. La probabilité qu'un groupe ne soit composé que d'attaquants est donc la probabilité contraire :

$$\boxed{P_{broken} = 1 - \prod_{i=1}^{s} 1 - k^{g_i}}$$

Étant donné que $1 - k^{g_i}$ est inférieur à 1, cette probabilité augmente quand le réseau grandit. De plus, pour un réseau de taille donnée, la diminution de la taille des groupes g_i (et donc l'augmentation du nombre de groupes s) augmente la probabilité de cette attaque. Cette probabilité est évaluée théoriquement et expérimentalement dans le chapitre 5. Avec des groupes constitués de $g_{min} = 20$

à $g_{max} = 40$ membres, il faut au moins 50% d'attaquants pour mener à bien cette attaque.

Les deux probabilités présentées dans cette section – obtenir la clé secrète du réseau ou détruire le partage de la clé – impliquent donc un compromis sur la taille des groupes.

2.6 Applications de l'autorité de certification distribuée

Dans cette section, nous introduisons les deux application présentées dans ce mémoire pour l'autorité de certification distribuée. Ces applications, détaillées dans la seconde partie de ce mémoire, ont pour but d'enrayer l'attaque sybile et de proposer un système de nommage sécurisé.

2.6.1 Protection contre l'attaque sybile par contrôle d'admission

Dans l'attaque sybile [Dou02], un attaquant crée un grand nombre d'identités et peut en extraire un sous-ensemble spécifique afin de contrôler complètement une ressource ou d'isoler un membre honnête. Une solution doit à la fois empêcher un attaquant de générer un grand nombre d'identifiants et contraindre ces identifiants à être réellement aléatoires.

En utilisant des liens sociaux, SybilGuard [YKGF06a] permet à chaque pair d'évaluer localement si un autre pair est sybil ou non, ce qui permet de limiter le nombre d'identités sybiles. En revanche, l'identifiant est le condensat de la clé publique du pair : un attaquant peut donc générer un grand nombre de clés, choisir la clé publique qu'il souhaite, puis l'enregistrer dans SybilGuard.

Dans le chapitre 6, nous proposons d'associer SybilGuard à l'autorité de certification distribuée. Pour accéder au réseau, un membre doit obtenir un certificat contenant sa clé publique signé par la clé privée du réseau et doit donc convaincre un ratio t des pairs. Chacun de ces pairs teste le nouveau membre avec SybilGuard et ne coopère que s'il détecte que le nouveau membre n'est pas sybil ; si le nouveau membre est détecté comme sybil par un grand nombre de pairs, il n'obtient pas de certificat. Ensuite, l'identifiant du pair est le condensat du certificat signé obtenu, qui n'est pas prévisible avant son obtention. Un attaquant ne peut ni générer un grand nombre de pairs sybils, ni choisir leurs identifiants. Le réseau s'auto-protège ainsi des attaques sybiles.

2.6.2 Nommage sécurisé des ressources

Dans [BLJ05], Bryan *et al.* proposent d'utiliser une DHT comme un annuaire pour de la voix sur IP. Chaque utilisateur insère une entrée dans la DHT liant son nom tel que « Jean Dupont » à son adresse IP : cette entrée est insérée sous l'identifiant h(« Jean Dupont »). Quand un utilisateur appelle Jean Dupont, il obtient son enregistrement dans la DHT et le contacte à l'adresse IP indiquée. Il n'existe cependant aucune garantie que la bonne adresse IP est utilisée. En effet, un attaquant peut insérer un enregistrement identique ou intercepter la requête. La solution généralement utilisée est soit d'utiliser des routes redondantes, coûteuses et augmentant la latence, soit d'appeler Jean Dupont par le condensat

de sa clé publique, ce qui pose le problème de l'obtention de ce condensat (un condensat SHA-1 a une taille de 160 bits, soit 40 chiffres hexadécimaux).

Dans le chapitre 7, nous proposons de fournir à chaque utilisateur un certificat liant son nom intelligible à sa clé publique. Chaque pair impliqué dans la certification vérifie que le nom demandé est libre, ce qui implique l'utilisation de routes différentes par les différents pair. Les mécanismes de cache de la DHT permettent de gérer ce pic de trafic. Un membre recherchant ensuite Jean Dupont obtient son certificat et a la garantie de contacter la personne enregistrée avec ce nom.

Résumé

Dans ce chapitre, nous avons présenté le modèle de l'autorité de certification distribuée proposée dans ce mémoire.

Dans la section 2.1, nous avons introduit le partage de la clé. Le réseau est divisé en groupes de taille constante et la clé secrète du réseau est partagée entre ces groupes, chaque membre d'un groupe donné connaissant le fragment du groupe. Lorsque le réseau grandit, le nombre de groupes augmente et le nombre de fragments de la clé secrète augmente également : cela permet de maintenir un ratio fixe de nœuds devant coopérer à une certification distribuée. Cette contrainte est assurée localement dans chaque groupe, sans communication globale, ce qui permet le passage à l'échelle.

Dans la section 2.2, nous avons expliqué l'obtention d'une signature distribuée. Une requête de certification est transmise à un pair de chaque groupe de fragmentation qui signe avec son fragment. Les signatures partielles sont ensuite combinées pour obtenir la signature du certificat. La mise en œuvre de la signature distribuée est détaillée dans le chapitre 3.

Dans la section 2.3, nous avons présenté les opérations de maintenance de la fragmentation. Ces opérations, locales à un ou deux groupes de fragmentation, permettent de diviser un groupe en deux, de rafraîchir deux fragments, de fusionner deux groupes, de connecter un pair et de déconnecter un pair. La mise en œuvre de ces opérations sans consensus est proposée dans le chapitre 4.

Dans la section 2.4, nous avons étudié les différentes possibilités pour générer la clé du réseau. Si la clé est générée de manière centralisée, il reste un point de faiblesse dans le système. Nous avons donc proposé de générer la clé de manière distribuée entre les membres d'un consortium clairement établi ou entre les membres fondateurs. Expérimentalement, nous avons pu vérifier que la génération distribuée était possible avec environ 40 pairs.

Dans la section 2.5, nous avons évalué la sécurité de notre modèle de manière probabiliste. Nous avons étudié la probabilité que des attaquants obtiennent la clé secrète complète en appartenant à chaque groupe de fragmentation et que des attaquants rendent l'autorité de certification distribuée indisponible en possédant tous les pairs d'un groupe. L'évaluation de ces probabilités ainsi que les résultats expérimentaux associés sont présentées chapitre 5. Cette évaluation montre qu'avec les tailles de groupe proposées ($g_{min} = 20$ et $g_{max} = 40$), il faut au moins 20% d'attaquants pour mener à bien l'une de ces attaques.

Enfin, dans la section 2.6, nous avons introduit les deux applications de l'autorité de certification distribuée que nous développons dans la seconde partie de ce mémoire, à savoir une protection contre l'attaque sybile (chapitre 6) et un

système de nommage certifié (chapitre 7).

Ce travail a été publié en français à SAR-SSI 2007 [LMV07a] puis en anglais à AIMS 2008 [LMV08b].

Après avoir présenté le modèle de l'autorité de certification distribuée, nous présentons dans les chapitre suivants de cette première partie la mise en œuvre de la signature distribuée, la maintenance des groupes de fragmentation ainsi que l'implémentation et l'évaluation de l'autorité de certification distribuée. La seconde partie de ce mémoire est ensuite consacrée aux deux applications proposées.

Chapitre 3

Signature distribuée

« *Trust, but Verify.* »
Ronald Reagan

Dans ce chapitre, nous présentons le processus de signature distribuée. La signature distribuée est obtenue en signant avec chaque fragment de clé et en multipliant les résultats : la signature complète n'est donc obtenue que si un pair de chaque groupe de fragmentation accepte de coopérer. L'algorithme utilisé déploie un arbre binaire recouvrant sur les groupes de fragmentation, ce qui permet de répartir la charge induite par chaque signature. L'arbre peut être redondant pour tolérer des attaquants, au prix d'un coût non négligeable ; ce coût est diminué par la sélection progressive de pairs honnêtes.

Dans la section 3.1, nous présentons en détail l'algorithme de signature distribuée et nous l'illustrons avec le partage de clé proposé dans le chapitre précédent.

Dans la section 3.2, nous nous intéressons à l'impact des pairs malveillants qui peuvent corrompre une signature distribuée. Nous proposons de détecter ces attaques par des requêtes redondantes et nous présentons un mécanisme pour progressivement sélectionner les pairs honnêtes.

Dans la section 3.3, nous évaluons la probabilité de succès de l'algorithme de signature en fonction du nombre de pairs, de la taille des groupes de fragmentation, du nombre d'attaquants et de la redondance utilisée.

Enfin, dans la section 3.4, nous décrivons le format des signatures et des fragments de clé.

3.1 Algorithme de signature distribuée

Étant donné le partage de la clé proposé précédemment et rappelé figure 3.1, nous expliquons l'algorithme de signature distribuée associé. Un pair A souhaite obtenir le certificat correspondant à la requête $Req = (Content, Clues)$ où :

- $Content$ contient les données du certificat à signer ;
- $Clues$ est composé d'éventuelles données supplémentaires pour aider chaque pair à prendre sa décision de coopération.

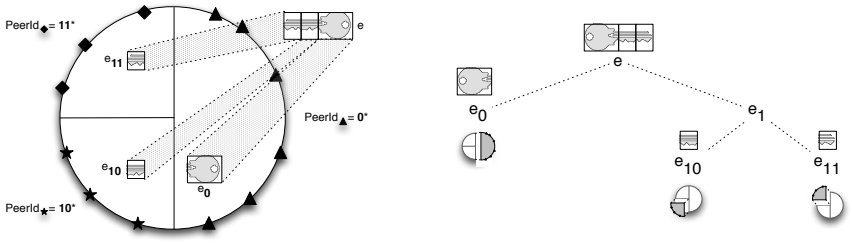


FIGURE 3.1 – À gauche, la décomposition de l'*overlay* en groupes de fragmentation ; à droite, l'arbre correspondant à cette décomposition.

A a besoin de la coopération et de l'accord d'un pair de chaque groupe de fragmentation pour obtenir la signature de son certificat, signature qui vaut $h(Content)^e[m]$. Afin de passer à l'échelle, nous proposons un algorithme de signature arborescent permettant de maintenir une charge équilibrée dans le réseau. Cette signature est réalisée en deux phases par un algorithme récursif distribué ; cet algorithme utilise le fait que la fragmentation de la clé génère un arbre binaire entier (chaque nœud de l'arbre a zéro ou deux fils).

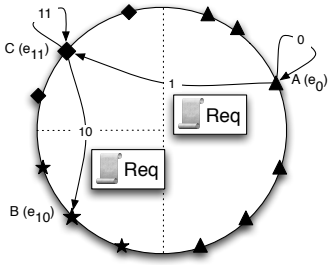
Dans cet arbre, les nœuds correspondent aux fragments de la clé : chaque nœud a zéro fils s'il représente un groupe existant et deux fils s'il représente un groupe s'étant déjà divisé. Sur la figure 3.1, le réseau est décomposé en trois groupes $\mathring{e_0}$, $\mathring{e_{10}}$ et $\mathring{e_{11}}$. Les nœuds de l'arbre e_0 et e_1 ont été créés par la division de e, puis e_{10} et e_{11} ont été créés par division de e_1. Dans cette configuration, seuls e_0, e_{10} et e_{11} existent dans le réseau ; e et e_1 ne sont connus de personne mais sont définis implicitement par $e = e_0 + e_1$ et $e_1 = e_{10} + e_{11}$. L'opération de signature consiste à réaliser un parcours de cet arbre.

La signature distribuée d'une requête d'un pair A est illustrée figure 3.2. La première étape de la signature distribuée est de déployer un arbre binaire recouvrant sur les groupes de fragmentation (figure 3.2(a)) :

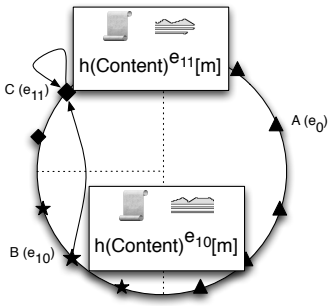
1. A communique sa requête *Req* à deux pairs quelconques dont les identifiants commencent respectivement par 0 et 1. A est automatiquement éligible pour 0 et C est choisi pour 1. C peut soit être trouvé par routage pair-à-pair soit être directement présent dans la table de routage de A, ce qui est le cas avec Kademlia ;
2. A possède le fragment e_0 et arrête la récursivité à ce niveau ;
3. C ne possède en revanche pas e_1 (que personne ne connaît) et il retransmet la requête à deux pairs B et C dont les identifiants commencent respectivement par 10 et 11 ;
4. B et C possèdent chacun le fragment demandé et arrêtent la récursivité.

À ce moment-là, la requête est transmise à exactement un pair de chaque groupe de fragmentation. La seconde étape est de créer la signature complète :

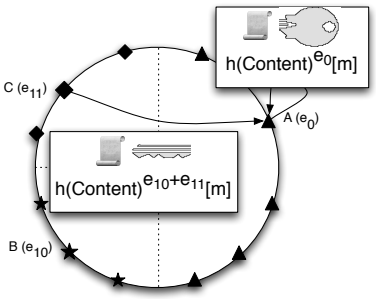
1. B et C réalisent leur signature partielle avec e_{10} et e_{11} et envoient les résultats $h(Content)^{e_{10}}[m]$ et $h(Content)^{e_{11}}[m]$ à C (figure 3.2(b)) ;
2. C multiplie ces deux signatures partielles et obtient la signature partielle avec e_1 qui vaut $h(Content)^{e_1}[m] = h(Content)^{e_{10}+e_{11}}[m]$. C envoie cette signature à A ;



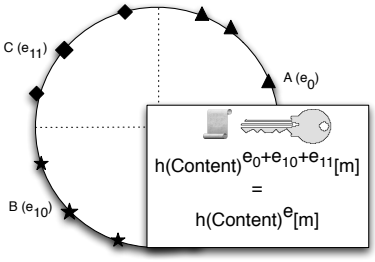
(a) A transmet la requête *Req* à un pair de chaque groupe de fragmentation



(b) B et C réalisent leur signature partielle avec e_{10} et e_{11}



(c) C multiplie les deux signatures partielles et A réalise sa signature partielle



(d) A obtient la signature complète de *Content* avec $S = (e, m)$

FIGURE 3.2 – Signature distribuée de *Req* avec $S = (e, m)$. Les fragments ne transitent pas, seules les signatures partielles sont envoyées.

3. A signe avec e_0 pour obtenir $h(Content)^{e_0}[m]$ (figure 3.2(c)) ;

4. A multiplie enfin les deux signatures partielles avec e_0 et e_1 et obtient $h(Content)^{e}[m]$.

A obtient $h(Content)^{e}[m]$ qui est la signature de $Content$ avec la clé secrète $S = (e, m)$ (figure 3.2(d)).

Chaque pair peut également décider de ne pas coopérer, par désaccord avec la requête de certification, auquel cas il répond négativement à la sollicitation lors de la première étape ; le pair demandeur peut alors essayer de trouver un autre pair du même groupe pour le remplacer.

L'algorithme utilisé par chaque pair pour obtenir ce comportement, à l'exception du remplacement d'un pair refusant de coopérer, est présenté de manière simplifiée algorithme 1. La fonction *Cooperate* est une fonction tierce, décrite dans la seconde partie de ce mémoire, permettant de prendre la décision de coopérer ou non.

Algorithme 1 Algorithme de signature distribuée

$Cert(Req = (Content, Clues), i)$:
  si $Cooperate(Req)$ **alors**
    si e_i est le fragment possédé **alors**
      Retourner $h(Content)^{e_i}[m]$
    sinon
      $sig_0 = Cert(Req, i0)$
      $sig_1 = Cert(Req, i1)$
      Retourner $(sig_0 \times sig_1)[m]$
    fin si
  sinon
    Retourner *deny*
  fin si

3.2 Tolérance aux pairs malveillants

L'algorithme de signature distribuée présenté jusqu'ici ne prend pas en considération la présence de pairs malveillants. Un attaquant peut en effet renvoyer un mauvais chiffrement (une fausse exponentiation ou une fausse multiplication) et le demandeur ne peut pas tester les signatures partielles, les clés publiques correspondant à chaque fragment n'étant pas connues. Le demandeur ne peut que vérifier la signature complète avec la clé publique du réseau, sans pouvoir discriminer les différentes signature partielles.

Dans l'état de l'art, à la section 1.3.2.1, nous avons présenté les mécanismes de *témoin* précédemment proposés contre cette attaque. Une ou plusieurs valeurs témoins sont publiées pour chaque fragment, sous la forme d'un message en clair et d'un message chiffré par ce fragment ; ces témoins permettent ensuite de vérifier un chiffrement partiel. Cependant, dans notre cas, la publication de ces témoins pose elle-même un problème : en effet, comment prouver que le témoin obtenu est bien le témoin valide ? De plus, les groupes se divisent, fusionnent et rafraîchissent leurs fragments : cette dynamicité complique encore l'obtention de témoins à jour. Nous proposons donc plutôt une approche basée sur des requêtes redondantes et sur la sélection progressive des pairs honnêtes.

3.2.1 Principe de la redondance

Afin de détecter les mauvaises réponses, le demandeur transmet la requête de signature à plusieurs pairs au lieu d'un seul. En faisant l'hypothèse que la majorité des pairs sont honnêtes, le demandeur choisit la réponse majoritaire qui est probablement la bonne. Le principal inconvénient de cette méthode est le surcoût engendré, puisque plusieurs pairs réalisent finalement le même chiffrement en parallèle.

Pour limiter ce surcoût, nous proposons une sélection progressive des pairs honnêtes par chaque pair. Nous notons tout d'abord que chaque pair délègue des signatures partielles pour un faible nombre d'identifiants de fragments. Si nous prenons l'exemple d'un pair A connaissant le fragment e_{010}, ce pair ne peut être interrogé que pour les 4 signatures avec les fragments e, e_0, e_{01} et e_{010}, où e dénote l'appel initial pour le fragment racine correspondant à la clé secrète en entier. Dans chacun de ces cas, le pair A est éligible pour l'un des deux appels

récursifs et ne transmet que l'autre appel, qui peut être adressé respectivement à l'un des 3 fragments e_1, e_{00} et e_{011}, à comparer avec les $2^4 - 1 = 15$ signatures partielles possibles. Si la longueur des identifiants de fragments vaut l, ce qui correspond à un arbre de fragmentation de hauteur l avec chacune des feuilles regroupant au moins g_{min} pairs, alors chaque pair ne peut déléguer des signatures partielles que pour l identifiants de fragments parmi les $2^{l+1} - 1$ identifiants (groupes de fragmentation) présents dans le réseau. Ces l identifiants de fragments correspondent aux k-buckets de Kademlia.

Pour chacune de ces signatures partielles possibles, chaque pair peut sélectionner un ensemble de pairs répondant les mêmes valeurs, considérées donc comme les valeurs honnêtes. Ensuite, les signatures peuvent être demandées sans redondance à l'un de ces pairs sélectionnés.

3.2.2 Mise en œuvre

Chaque pair évalue lui-même les candidats et fait ses choix localement. Lors de la première requête pour une signature avec un fragment donné, le pair A interroge $nbAsksMax$ pairs et choisit la réponse majoritaire. Les pairs ayant renvoyé cette réponse majoritaire sont choisis pour les futures requêtes sur ce même fragment. Ensuite, lors d'une nouvelle requête pour le même fragment, A choisit un unique pair parmi les pairs précédemment sélectionnés.

Lorsqu'un de ces pairs honnêtes se déconnecte, A recherche un nouveau pair honnête pour maintenir cet ensemble à une taille supérieure à $nbAsksSet$. A remplace le pair déconnecté en interrogeant un nouveau pair candidat et en comparant sa réponse avec celle d'un pair déjà sélectionné. Même avec la volatilité du réseau, A connaît toujours un ensemble de pairs honnêtes pour chaque signature partielle qu'il peut demander.

L'ensemble des pairs honnêtes pour une signature partielle contient donc des pairs qui, à la fois :

- sont honnêtes ;
- fournissent des signatures partielles valides et savent donc obtenir ces signatures de pairs honnêtes également.

À terme, une requête de certification reste donc dans une partie honnête de l'*overlay*.

Un nombre d'attaquants important peut, toutefois, biaiser l'hypothèse selon laquelle la réponse majoritaire est la réponse valide. Dans ce cas, un pair honnête peut sélectionner un ensemble de pairs attaquants pour un identifiant donné et ensuite transmettre toutes ses requêtes vers ces attaquants. Un attaquant pourrait également répondre parfois de manière honnête afin d'être sélectionné puis répondre de manière malhonnête. Deux précautions permettent de limiter ces possibilités :

- chaque pair réinitialise périodiquement ses ensembles de pairs honnêtes avec de nouveaux pairs ;
- chaque pair détectant une réponse reçue erronée le signale au pair incriminé par un mécanisme de retour. Ainsi, si un pair honnête reçoit plusieurs erreurs pour ses chiffrements avec un fragment donné, il peut déduire que son ensemble de pairs choisis est mauvais et réinitialiser uniquement cet ensemble.

Dans le chapitre 5, nous évaluons cette solution et montrons que :
- la sélection des pairs parmi $nbAsksMax$ permet d'atteindre une robustesse presque équivalente à l'interrogation de $nbAsksMax$ pairs à chaque signature partielle, pour un coût bien moindre ;
- le mécanisme de retour, qui permet de réinitialiser les mauvais choix de pairs, permet d'obtenir une robustesse encore meilleure.

Dans la section suivante, nous étudions la probabilité de succès de l'algorithme de signature distribuée en présence d'attaquants, sans ce système de sélection des meilleurs pairs.

3.3 Probabilité de succès de l'algorithme de signature distribuée

Nous calculons ici la probabilité de succès de l'algorithme de signature distribuée dans le cas d'une demande de certification légitime. Chaque pair honnête accepte de coopérer ; chaque attaquant essaie de corrompre le processus en renvoyant une valeur erronée. Chaque requête est transmise à $nbAsks$ pairs, $nbAsks$ impair pour obtenir une majorité, et la valeur utilisée est la valeur retournée par plus de la moitié des pairs. Nous supposons en effet le pire cas où les attaquants agissent en collusion et retournent la même signature incorrecte. Soit $k \in [0, 1]$ le ratio d'attaquants dans le réseau.

Nous considérons l'arbre des appels récursifs pour obtenir la signature partielle avec le fragment d'identifiant i (la figure 3.1 illustre l'arbre des appels pour une signature avec la clé complète) :
- les nœuds internes de cet arbre correspondent aux fragments qui ne sont pas présents dans le réseau et qui provoquent deux appels récursifs avec des identifiants plus longs ;
- les feuilles de cet arbre correspondent aux fragments de la clé secrète dont l'identifiant commence par i, i étant la racine de l'arbre des appels récursifs.

Chaque nœud de cet arbre coïncide avec un ou plusieurs pairs qui doivent signer avec l'identifiant de ce nœud.

Nous définissons la probabilité de succès d'une signature partielle de manière récursive sur la hauteur de l'arbre représentant les appels récursifs. $P_{success}(h)$ est la probabilité qu'une signature partielle réussisse dans un arbre de hauteur h (cette probabilité inclut celle que la racine de l'arbre soit honnête) :
- si $h = 0$ (cas d'une feuille), la signature réussit si et seulement si ce pair est honnête :
$$P_{success}(0) = 1 - k$$
- si $h > 0$, la certification réussit si et seulement si la racine est honnête et les deux appels récursifs réussissent :
  - comme la racine est automatiquement éligible pour un des appels récursifs, la probabilité que ce pair soit honnête et que le premier appel réussisse vaut $P_{success}(h-1)$;
  - le second appel est transmis à $nbAsks$ pairs et réussit si la majorité des requêtes réussissent. Une requête réussit avec une probabilité $P_{success}(h-1)$ et échoue avec une probabilité $1 - P_{success}(h-1)$: j requêtes parmi $nbAsks$ réussissent donc avec la probabilité suivante $C^{j}_{nbAsks} P_{success}(h-1)^{j}(1 - P_{success}(h-1))^{nbAsks-j}$

Le second appel réussit si plus de la moitié des requêtes réussissent, soit une probabilité $\sum_{j=\frac{nbAsks+1}{2}}^{nbAsks} C^j_{nbAsks} P_{success}(h-1)^j(1-P_{success}(h-1))^{nbAsks-j}$.

La probabilité que les deux appels réussissent vaut donc :

$$P_{success}(h) = P_{success}(h-1) \times \sum_{j=\frac{nbAsks+1}{2}}^{nbAsks} C^j_{nbAsks} P_{success}(h-1)^j(1-P_{success}(h-1))^{nbAsks-j}$$

Pour obtenir la signature complète, la racine de l'arbre est appelée avec l'identifiant de fragment vide et les feuilles représentent donc les s fragments, avec $s = t \times n$ où t est le ratio de pairs devant coopérer et n la taille du réseau. Comme cet arbre est binaire et supposé équilibré (les pairs ont des identifiants aléatoires et les groupes de fragmentation se remplissent donc à la même vitesse), sa hauteur vaut $h = log_2(s)$. La probabilité de succès d'une signature distribuée vaut donc :

$$\boxed{P_{success}(log_2(t \times n))}$$

Nous pouvons conclure que plus le réseau est grand, plus la certification est difficile. Nous présentons les résultats associés à cette probabilité dans le chapitre 5 et montrons que la redondance ainsi que la sélection des pairs permettent la signature distribuée dans de grands réseaux.

3.4 Format des signatures et des fragments

Dans cette section, nous présentons le format des signatures RSA utilisées ainsi que les implications sur le format des fragments de clé.

3.4.1 Format des signatures

Nous utilisons ici l'algorithme de signature RSASSA-PKCS1-v1_5 décrit dans [RSA02]. Cet algorithme de signature RSA déterministe est celui proposé officiellement par les *RSA Laboratories*. Dans le schéma RSASSA-PKCS1-v1_5, l'exponentiation RSA est réalisée sur une version encodée de la donnée à signer (*padding*). Cet encodage peut être vu comme une fonction de hachage et nous notons $h(o)$ l'encodage de la donnée à signer o.

La signature RSA de o avec une clé $S = (e, m)$ est alors $h(o)^e[m]$. Selon les spécifications, $h(o)$ a la même longueur que le module m et est de la forme `0x00||0x01||PS||0x00||T`, où :

- PS est une chaîne d'octets extensible contenant des valeurs `0xff`, utilisée pour rallonger $h(o)$ jusqu'à la longueur de m ;
- T est l'encodage DER de `DigestInfo`, contenant le type de condensat utilisé ainsi que le condensat de la donnée à signer.

3.4.2 Format des fragments de clé

Dans notre proposition, les fragments de clé sont des entiers relatifs. Quand certains fragments sont négatifs, la signature RSA d'une donnée o avec e vaut toujours $h(o)^e[m] = h(o)^{\sum e_i}[m] = (\prod h(o)^{e_i}[m])[m]$, mais il faut s'assurer que

tous les $h(o)^{e_i}[m]$ existent. Si e_i est négatif, $h(o)^{e_i}[m]$ existe si et seulement si $h(o)$ est inversible modulo m.

La valeur $h(o)$ est inversible modulo m si et seulement si $pgcd(h(o), m) = 1$. Lors de la génération de la clé RSA, le module m vaut $p \times q$ avec p et q premiers. D'où :

$$pgcd(h(o), m) = 1 \Leftrightarrow (pgcd(h(o), p) = 1 \wedge pgcd(h(o), q) = 1)$$

Nous savons donc que $h(o)$ est inversible modulo m si $h(o)$ n'est un multiple ni de p ni de q. Nous calculons ici la probabilité pour cet encodage $h(o)$ d'être un multiple de p ou q et nous montrons que la structure imposée sur $h(o)$ lors de la signature RSA implique qu'au plus 2 parmi les 2^{160} valeurs possibles de $h(o)$ ne sont pas inversibles.

Nous considérons les conditions suivantes :
- le module RSA m fait 1024 bits
- les condensats sont réalisés avec SHA-1, donc sur 160 bits

Nous obtenons :
- $h(o)$ fait la longueur de m soit 1024 bits
- T fait 280 bits (120 bits pour coder que SHA-1 est utilisé et 160 bits pour le condensat en lui même)
- PS est étendu à 720 bits
- $h(o)$ est de la forme `0x00`||`0x01`||`0xff`||...||`0xff`||`0x00`||`T`, avec `0xff` répété 90 fois (720 bits font 90 octets)

Comme le module RSA fait 1024 bits, la procédure de génération crée p et q de 512 bits, c'est-à-dire que le premier des 512 bits de p et q vaut 1. Nous montrons par l'absurde qu'il existe au plus un message encodé qui soit un multiple de p.

Soit $h(o)$ un message encodé. Par la forme de $h(o)$, nous savons que :

$$h(o) \geq \sum_{i=511}^{1008} 2^i, \text{ soit } h(o) \geq 2^{511}\frac{1-2^{498}}{1-2} \qquad \text{et} \qquad h(o) < 2^{1009}$$

De plus, nous avons :

$$p \geq 2^{511}$$

Nous pouvons en déduire :

$$h(o) + p \geq 2^{511}\frac{1-2^{498}}{1-2} + 2^{511}, \text{ soit } h(o) + p \geq 2^{1009}$$

Imaginons maintenant qu'il existe plusieurs multiples de p qui soient des messages encodés valides et nous choisissons $h(o)$ le plus petit de ces multiples. Dans ce cas, $h(o) + p$ est plus grand que tous les messages encodés possibles : il y a donc au plus un message encodé qui soit un multiple de p, tout comme pour q. En utilisant SHA-1 pour créer les condensats, il y a 2^{160} messages encodés possibles parmi lesquels 2 au plus ne sont pas inversibles, ce qui est négligeable. Le même raisonnement fournit des valeurs identiques pour des clés RSA de 2048 bits. ■

Cependant, un attaquant pourrait exploiter le système pour trouver un $h(o)$ non inversible, qui est donc un multiple de p, et ainsi déduire p (un attaquant connaissant p peut factoriser le module m et retrouver la clé secrète). Cette

attaque n'est pas liée au partage de la clé mais à la connaissance du module m qui est public : cette attaque est présente dans le système RSA standard. Dans notre système, un attaquant pourrait, toutefois, distribuer la recherche exhaustive de valeurs non inversibles en demandant des certificats aléatoires à des pairs honnêtes. Pour empêcher cela, un attaquant doit passer plus de temps à demander un certificat qu'à inverser lui-même la valeur $h(o)$: la requête de certification contient donc l'inverse de $h(o)$ et l'attaquant n'a ainsi plus d'intérêt à demander des certificats aléatoires.

Nous avons de plus constaté expérimentalement que l'inversion était une opération rapide, typiquement 50 fois plus rapide qu'une exponentiation. L'inversion d'une donnée à signer a donc un coût négligeable pour un pair honnête et la distribution de cette attaque, même sans la contre-mesure proposée, n'est donc pas forcément intéressante pour un attaquant (coût des communications).

Résumé

Dans ce chapitre, nous avons présenté l'algorithme de signature distribuée. Une signature ne peut être obtenue que par la coopération d'un membre de chaque groupe de fragmentation et l'algorithme utilisé permet de répartir la charge sur l'ensemble des pairs sollicités.

Dans la section 3.1, nous avons présenté en détail l'algorithme de signature distribuée. Cet algorithme déploie un arbre binaire recouvrant sur l'ensemble des groupes de fragmentation, en distribuant les calculs sur l'ensemble des pairs sollicités. La signature complète est obtenue en agrégeant les signatures partielles lors de la remontée de l'arbre.

Dans la section 3.2, nous avons analysé l'impact de pairs malveillants et avons proposé une protection. Des pairs malveillants peuvent en effet retourner une signature partielle erronée et nous avons proposé de détecter ces erreurs par des requêtes redondantes. Afin de limiter le surcoût lié à la redondance, chaque pair sélectionne un sous-ensemble de pairs réalisant des signatures partielles valides, en utilisant de la redondance, et utilise ensuite ces pairs honnêtes sans redondance.

Dans la section 3.3, nous avons calculé la probabilité de succès de l'algorithme de signature distribuée. Cette probabilité est évaluée et comparée aux résultats expérimentaux dans le chapitre 5, résultats qui montrent que la signature distribuée est réalisable dans de grands réseaux contenant des attaquants.

Enfin, dans la section 3.4, nous avons décrit le format des signatures et des fragments. Les signatures utilisent le format standard RSASSA-PKCS1-v1_5 décrit dans [RSA02]. À partir de ce format de signature, nous avons montré que les fragments de clé pouvaient être des entiers relatifs.

Ce travail a été publié en français à SAR-SSI 2007 [LMV07a] puis en anglais à AIMS 2008 [LMV08b].

Chapitre 4

Maintenance de la fragmentation de la clé

« *Divide et impera.* »
(Diviser pour régner.)
Proverbe latin

Comme expliqué dans le chapitre 2, la politique de sécurité définit les tailles minimale g_{min} et maximale g_{max} des groupes de fragmentation afin de maintenir le ratio t de pairs devant coopérer à une signature. Quand un groupe atteint la taille g_{max}, il se *divise* en deux groupes afin de contraindre le ratio t de nœuds devant coopérer à une signature ; ces deux groupes *rafraîchissent* ensuite leur fragment. Quand un groupe atteint la limite g_{min}, il *fusionne* avec le groupe adjacent pour garantir la disponibilité du fragment connu. Un pair qui se *connecte* au réseau doit obtenir le fragment du groupe qu'il rejoint afin de pouvoir participer aux signatures distribuées. Enfin, un pair qui se *déconnecte* doit être retiré de la liste des membres du groupe. Cinq opérations sont ainsi nécessaires pour maintenir le partage de la clé.

Ces cinq opérations pourraient être mises en œuvre par des consensus byzantin au sein des groupes, afin de décider par exemple de la division. Cependant, étant donnée la taille importante des groupes de fragmentation, réaliser des consensus byzantins dans ou entre ces groupes semble déjà un problème difficile ; dans un contexte pair-à-pair, ce problème est encore plus difficile car des membres rejoignent ou quittent le réseau à tout moment, ce qui complique la définition de l'ensemble de pairs participant au consensus. Leur implémentation dans ce contexte dynamique est de plus particulièrement complexe.

Dans les sections 4.1, 4.2, 4.3, 4.4 et 4.5, nous réalisons ces cinq opérations sans consensus byzantin et sans synchronisation entre les pairs, en échange d'une structure de données nommée *arbre de fragmentation*. Les arbres de fragmentation permettent de définir tous les fragments possibles dans le système en utilisant une représentation compacte, tout en autorisant aux opérations de rafraîchissement la possibilité de modifier ces arbres. Nous définissons les arbres de fragmentation au fur et à mesure de leur utilisation dans les sections pré-

sentant les opérations de maintenance [1]. Ces cinq opérations de maintenance ne nécessitent que des communications internes à un groupe ou entre deux groupes ; les groupes étant de taille constante, le coût des opérations de maintenance ne dépend pas de la taille du réseau.

Les opérations de maintenance du partage de la clé doivent vérifier les trois propriétés de cohérence, de confidentialité et d'intégrité définies ci-après.

Propriété 4 (*Cohérence de la fragmentation*)

Tous les pairs du groupe $\mathring{e}_i$ connaissent la même valeur du fragment e_i.

La cohérence de la fragmentation garantit que chaque pair honnête d'un même groupe de fragmentation peut être interrogé indifféremment lors d'une signature distribuée. Les opérations de maintenance doivent maintenir individuellement cette propriété et nous démontrons cette propriété pour chacune des opérations présentées.

Propriété 5 (*Confidentialité des fragments*)

La valeur courante de chaque fragment e_i n'est connue que par les membres du groupe $\mathring{e}_i$ et est nécessaire pour obtenir une signature distribuée valide.

Propriété 6 (*Intégrité du partage*)

La somme de tous les fragments e_i présents dans le réseau vaut e.

La confidentialité des fragments garantit qu'un membre de chaque groupe de fragmentation doit coopérer pour obtenir une signature valide avec e ; l'intégrité du partage garantit qu'une signature distribuée réalisée par la coopération de pairs honnêtes est valide. Ces deux propriétés sont, elles, assurées par le fonctionnement global du système et sont démontrées dans la section 4.6.

4.1 Opération de division

Quand un groupe $\mathring{e}_i$ est composé de plus de g_{max} membres, ce groupe doit se diviser en deux groupes connaissant respectivement e_{i0} et e_{i1}, tels que $e_i = e_{i0} + e_{i1}$. Cette division permet de contraindre le ratio t de pairs devant coopérer pour obtenir une signature. Une solution consiste à choisir e_{i0} aléatoirement, puis fixer $e_{i1} = e_i - e_{i0}$. Dans le système que nous proposons, les pairs prennent cette décision sans consensus et sans synchronisation : tous les pairs d'un même groupe doivent pouvoir diviser le groupe à un moment légèrement différent, tout en générant les mêmes fragments.

Pour résoudre ce problème, tous les fragments possibles sont prédéfinis dans l'arbre de fragmentation, lors de l'initialisation du système ; ensuite, pendant la vie du système, aucun pair ne connaît cet arbre en entier, mais la somme de toutes les connaissances définit implicitement cet arbre de fragmentation complet.

Nous expliquons d'abord comment l'arbre de fragmentation est créé afin de rester compact. Nous décrivons ensuite le protocole de division des groupes. Nous montrons que les arbres de fragmentation restent cohérents après cette

1. Dans ce mémoire, nous utilisons le terme *nœud* pour désigner les nœuds des arbres de fragmentation et *pair* pour désigner les nœuds du réseau pair-à-pair.

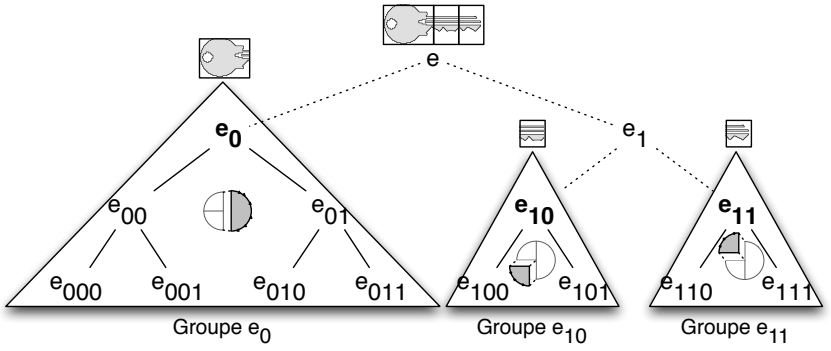


FIGURE 4.1 – Arbre de fragmentation avec des identifiants sur 3 bits. Il y a trois groupes de fragmentation $\mathring{e_0}$, $\mathring{e_{10}}$ et $\mathring{e_{11}}$ connaissant leur sous-arbre de fragmentation respectif. Seuls les fragment en gras (e_0, e_{10} et e_{11}) sont stockés : leurs sous-fragments peuvent être calculés et leurs fragments parents ne sont connus de personne. La somme des racines $e_0 + e_{10} + e_{11}$ vaut e et aucun pair ne connaît e ou e_1.

opération. Enfin, nous démontrons que la concurrence avec un processus de signature distribuée ne pose pas de problème.

Dans la section 4.2, nous décrivons le rafraîchissement, opération complémentaire à la division, qui permet de modifier les arbres de fragmentation au cours de la vie du système et ainsi d'assurer la confidentialité des fragments.

4.1.1 Arbre de fragmentation initial

L'arbre de fragmentation global est un arbre binaire dont la racine est e, l'exposant secret partagé entre les pairs. Cet arbre contient tous les fragments possibles. Chaque nœud de l'arbre est étiqueté par l'identifiant du fragment qu'il représente. Cet arbre est construit récursivement à l'initialisation du système de telle sorte que pour chaque nœud représentant un fragment e_i, ses fils e_{i0} et e_{i1} soient tels que $e_{i0} = RNG_{h(e_i)}$ et $e_{i1} = e_i - RNG_{h(e_i)}$, où $RNG_{h(e_i)}$ est l'entier généré aléatoirement à partir de la graine $h(e_i)$. Cette construction des fragments respecte le schéma proposé précédemment. Le sous-arbre de racine e_i peut être entièrement généré en connaissent seulement e_i et peut ainsi être facilement stocké et transféré. De plus, la fonction h est une fonction de hachage à sens unique et il n'est pas possible de calculer le parent d'un nœud. Les identifiants de fragment ont une longueur bornée (au plus la taille des identifiants des pairs) et l'arbre de fragmentation est donc fini : il peut être représenté comme illustré dans la figure 4.1 où les feuilles sont étiquetées par les plus longs identifiants possibles.

L'arbre de fragmentation global n'est connu par aucun pair en entier. Chaque pair appartenant au groupe $\mathring{e_i}$ connaît le sous-arbre de fragmentation de racine e_i, noté $\widehat{e_i}$ et contenant tous les fragments e_{i*}. À partir de l'unique connaissance de leur fragment local e_i, les pairs peuvent calculer tous les sous-fragments nécessaires. À chaque instant, chaque groupe de fragmentation connaît un sous-arbre différent des autres et la somme des racines de tous ces sous-arbres vaut e.

Les sous-arbres de fragmentation sont également illustrés figure 4.1.

Définition 14 (*Arbre de fragmentation $\widehat{e_i}$*)

L'arbre de fragmentation $\widehat{e_i}$ est un arbre binaire contenant tous les fragments dont l'identifiant commence par i. Il est construit récursivement avec :

- $e_{i0} = RNG_{h(e_i)}$
- $e_{i1} = e_i - RNG_{h(e_i)}$

Les pairs du groupe de fragmentation $\mathring{e}_i$ connaissent tous $\widehat{e_i}$. La racine e_i de cet arbre est le fragment de ce groupe.

Pour les arbres de fragmentation, nous adaptons la propriété de cohérence (propriété 4) définie dans l'introduction de ce chapitre :

Propriété 7 (*Cohérence des arbres de fragmentation*)

Pour tout fragment e_i, racine ou nœud d'un arbre de fragmentation, tous les arbres le contenant connaissent la même valeur pour e_i.

4.1.2 Protocole de division

Chaque pair connaît et maintient la liste des autres membres de son groupe de fragmentation. Chaque pair peut donc surveiller les autres membres de son groupe $\mathring{e}_i$ ainsi que les nouveaux arrivants et détecter quand le groupe dépasse le seuil des g_{max} membres. Plus précisément, comme chaque groupe créé doit lui-même être composé de g_{min} membres, la division n'est réalisée que si $\mathring{e}_{i0}$ et $\mathring{e}_{i1}$ sont tous deux composés de plus de g_{min} pairs. Si un pair décide de diviser le groupe, il passe du groupe $\mathring{e}_i$ à $\mathring{e}_{i0}$ ou $\mathring{e}_{i1}$ en fonction de son identifiant. Ensuite, il génère l'arbre de fragmentation $\widehat{e_{i0}}$ ou $\widehat{e_{i1}}$ en utilisant son arbre de fragmentation courant $\widehat{e_i}$.

Par exemple, un pair dont l'identifiant binaire commence par 00 et qui appartient au groupe $\mathring{e}_0$ connaît l'arbre de fragmentation $\widehat{e_0}$. Quand ce pair décide de diviser le groupe, il va dans le groupe $\mathring{e}_{00}$ et choisit le sous-arbre $\widehat{e_{00}}$ comme arbre de fragmentation. Connaissant l'arbre de fragmentation courant, chaque pair peut diviser le groupe de son côté, tout en effectuant une opération cohérente avec celle de chaque autre pair. Cette division est illustrée figure 4.2.

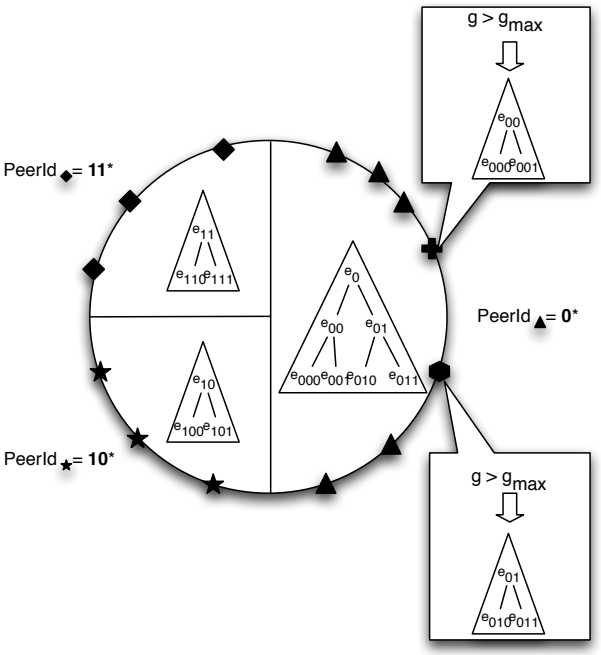
Cependant, les anciens pairs de $\mathring{e}_i$ connaissent à la fois $\widehat{e_{i0}}$ et $\widehat{e_{i1}}$: ils connaissent tous les fragments e_{i*} dérivés depuis e_i. L'opération de rafraîchissement modifie les arbres de fragmentation et traite précisément ce problème de confidentialité des fragments.

4.1.3 Cohérence des arbres de fragmentation

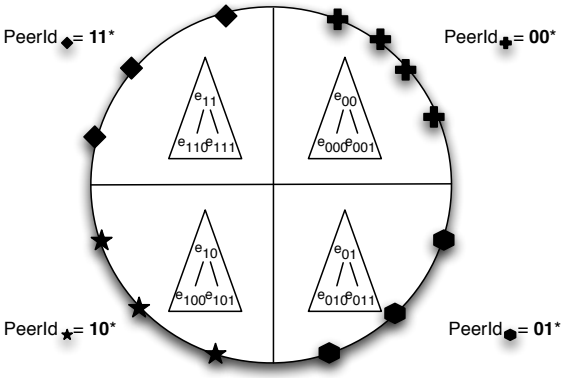
Avant l'opération de division, les arbres de fragmentation sont cohérents et chaque pair de $\mathring{e}_i$ connaît donc les même valeurs pour tous les nœuds des sous-arbres $\widehat{e_{i0}}$ et $\widehat{e_{i1}}$. Chaque pair utilisant sa version locale du sous-arbre $\widehat{e_{i0}}$ ou $\widehat{e_{i1}}$ et les nœuds de ces sous-arbres n'étant pas modifiés durant l'opération, les arbres sont toujours cohérents après la division.

4.1.4 Concurrence avec un processus de signature

Cette opération n'étant pas synchronisée entre les pairs d'un groupe, certains pairs peuvent mettre plus de temps que d'autres à détecter le besoin de diviser



(a) Avant la division, certains pairs constatent que la taille du groupe est supérieure à 6 et divisent localement leur groupe.



(b) Après la division, tous les pairs de $\mathring{e_0}$ ont divisé le groupe.

FIGURE 4.2 – Division du groupe $\mathring{e_0}$, avec $g_{max} = 6$.

et ainsi réaliser l'opération de division plus tard. Dans ce cas, certains pairs sont dans $\mathring{e}_i$, d'autres dans $\mathring{e}_{i0}$ et les autres dans $\mathring{e}_{i1}$. Étant donné l'algorithme de signature distribuée présenté, chacun de ces pairs peut recevoir une requête pour signer avec e_i :

- si un pair de $\mathring{e}_i$ reçoit cette requête, ce pair signe avec e_i et retourne le résultat ;
- si un pair de $\mathring{e}_{i0}$ reçoit cette requête, ce pair doit transmettre récursivement la requête à lui-même et à un pair de $\mathring{e}_{i1}$. Ce second pair est soit dans $\mathring{e}_i$ soit dans $\mathring{e}_{i1}$ et, dans tous les cas, connaît e_{i1} et peut signer avec ce fragment.

Cette désynchronisation passagère est donc compatible avec le processus de signature.

4.2 Opération de rafraîchissement

Quand un groupe $\mathring{e}_i$ se divise en deux groupes $\mathring{e}_{i0}$ et $\mathring{e}_{i1}$, chaque pair connaît le fragment du groupe frère puisqu'il connaissait $\widehat{e_i}$. De plus, un attaquant connaissant $e_i = e_{i0} + e_{i1}$ peut utiliser ce fragment pour signer à la place des deux groupes $\mathring{e}_{i0}$ et $\mathring{e}_{i1}$. Le rafraîchissement de e_{i0} et e_{i1} permet de rendre inutilisables les anciennes versions connues par un attaquant.

Pour rafraîchir son fragment, le groupe $\mathring{e}_{i0}$ procède de la manière suivante :

1. $\mathring{e}_{i0}$ choisit aléatoirement un autre groupe $\mathring{e}_x$;
2. $\mathring{e}_{i0}$ et $\mathring{e}_x$ échangent une valeur aléatoire Δ ;
3. Δ est ajouté à e_{i0} et soustrait de e_x.

Après cette opération, les nouvelles versions de e_{i0} et e_x ne sont connues que dans leurs groupes respectifs. En effet, un pair de $\mathring{e}_{i1}$, qui connaissait e_{i0}, ne connaît pas la valeur aléatoire Δ utilisée pour rafraîchir : ce pair ne connaît donc pas la nouvelle version de e_{i0}. De plus, e_{i0} a changé et donc $e_{i0} + e_{i1}$ a également changé : la valeur e_i, qui était connue par le groupe père, n'est plus synchronisée avec le partage actuel et ne peut plus être utilisée pour obtenir une signature avec e.

Les différentes opérations de maintenance n'étant pas synchronisées, elles peuvent être exécutées de manière concurrente. Par exemple, un groupe peut être en train de se diviser lorsqu'une opération de rafraîchissement est initialisée. La division change l'état du groupe, notamment son fragment de clé, et le rafraîchissement ne peut donc pas se fonder sur l'état courant du groupe (qui peut être différent entre les pairs). Le rafraîchissement est donc plutôt fondé sur des modifications des arbres de fragmentation qui, eux, sont toujours cohérents entre les pairs (si un pair n'a pas divisé son groupe, son arbre de fragmentation contient les arbres de fragmentation des pairs ayant divisé le groupe).

Nous présentons d'abord comment les arbres de fragmentation sont modifiés pour matérialiser un rafraîchissement. Nous présentons ensuite le protocole de rafraîchissement. Nous montrons que les arbres restent cohérents après un rafraîchissement. Nous expliquons comment gérer la concurrence avec un processus de signature distribuée. Nous décrivons enfin l'utilisation de cette opération.

4.2.1 Arbres de fragmentation modifiés

Comme expliqué précédemment, les identifiants de fragments ont une longueur bornée et les arbres de fragmentation sont donc finis. Un rafraîchissement

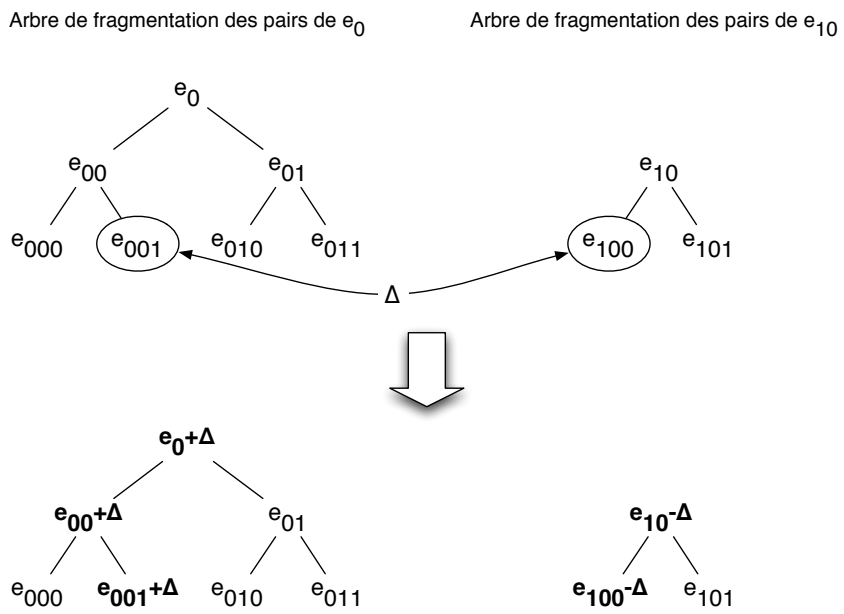


FIGURE 4.3 – Opération de rafraîchissement entre les feuilles e_{001} et e_{100} dans les groupes de fragmentation $\mathring{e_0}$ and $\mathring{e_{10}}$.

consiste à ajouter une valeur aléatoire Δ à une feuille et à la soustraire d'une autre feuille. Quand la valeur d'une feuille change, toutes les valeurs de cette feuille à la racine de l'arbre sont mises à jour pour maintenir la propriété que pour tout nœud e_i de l'arbre, $e_i = e_{i0} + e_{i1}$. Si un pair connaît l'arbre de fragmentation $\widehat{e_0}$ et si Δ est ajouté à la feuille e_{001}, alors Δ est également ajouté à e_{00} et e_0, comme montré sur la figure 4.3.

Cependant, quand la valeur e_i devient $e'_i = e_i + \Delta$, e_{i0} n'est plus égal à $RNG_{h(e'_i)}$ et ne peut plus être calculé automatiquement à partir de l'ancien e_i qui a été effacé. Quand une feuille est mise à jour, cette feuille et tous ses ancêtres sont mis à jour et définis explicitement, ce qui permet ensuite de pouvoir calculer tous les fragments de l'arbre. Ce schéma est ilustré figure 4.3, où les nœuds mis à jour en gras sont explicitement définis et stockés, alors que les autres nœuds sont implicites et peuvent être calculés. Nous montrons dans le chapitre 5 qu'un arbre de fragmentation occupe moins d'un méga-octet.

Enfin, les anciennes valeurs des fragments sont effacées pour empêcher un groupe d'attaquants de retrouver la clé complète illégitimement. En effet, le rafraîchissement a pour but de désynchroniser l'ancienne version d'un fragment avec le partage actuel ; si l'ancienne version n'est pas supprimée, un attaquant peut combiner les versions initiales de différents fragments obtenus à des moments différents, et ainsi obtenir la clé secrète complète sans posséder un pair dans chaque groupe à un même instant.

4.2.2 Protocole de rafraîchissement

Le protocole de rafraîchissement doit garantir la cohérence du partage de la clé. Ainsi, si un membre honnête réalise le rafraîchissement, alors tous les pairs honnêtes concernés doivent le faire. Un rafraîchissement se déroule de la manière suivante :

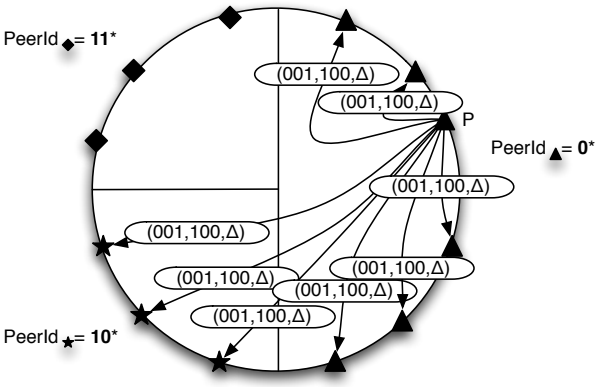
1. l'initiateur P crée une requête de rafraîchissement entre sa feuille locale, correspondant à la feuille de son arbre de fragmentation ayant le plus grand préfixe commun avec son identifiant, et une autre feuille distante r choisie aléatoirement de l'autre côté de la racine de l'arbre global de fragmentation, en utilisant une valeur Δ. La feuille distante est choisie de l'autre côté de l'arbre pour assurer que les pairs connaissant la feuille r ne connaissent pas également la feuille locale, ce qui est utilisé dans la preuve de confidentialité des fragments section 4.6. Dans la figure 4.3, l'identifiant de P est 001101 et la feuille locale est donc e_{001} ; la feuille distante est e_{100} ;
2. P transmet la requête à chacun des pairs de son groupe et à chaque pair du groupe distant. Pour obtenir le groupe distant responsable de la feuille r, dans Kademlia qui utilise la distance XOR, P recherche la ressource d'identifiant $r00\ldots00$ et obtient les x pairs partageant le plus long préfixe commun avec cette ressource : avec $x = g_{min}$, tous ces pairs appartiennent au même groupe de fragmentation et la probabilité de n'obtenir que des attaquants est minimisée ;
3. chaque pair recevant cette requête pour la première fois la retransmet à tous les membres des deux groupes concernés ;
4. un condensat de la requête de certification est conservé avec les deux feuilles afin de ne pas réexécuter un rafraîchissement identique.

Si un pair honnête reçoit la requête, alors tous les pairs honnêtes la reçoivent et l'appliquent, ce qui assure la cohérence des arbres de fragmentation. Cette opération est illustrée figure 4.4.

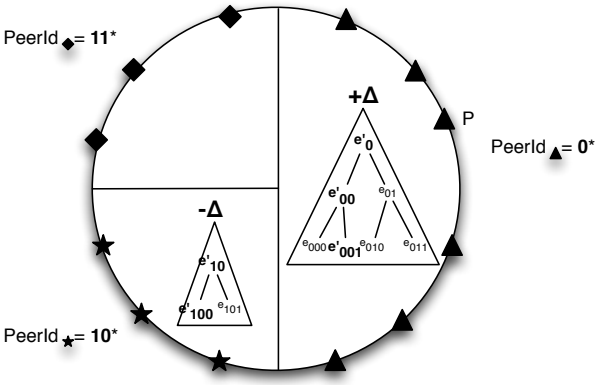
4.2.3 Cohérence des arbres de fragmentation

Comme expliqué précédemment, les opérations de maintenance ne sont pas synchronisées et peuvent donc être exécutées de manière concurrente. Le rafraîchissement doit ainsi pouvoir être exécuté en parallèle d'une opération de division ou de fusion de groupe, durant laquelle certains pairs appartiennent à e°_i et d'autres à e°_{i0} ou e°_{i1}. Nous étudions ici une opération de rafraîchissement touchant un tel groupe en cours de division ou de fusion :

- si un pair appartenant à e°_i reçoit une requête de rafraîchissement le concernant, il transmet cette requête à tous les membres de son groupe. Selon son point de vue, le groupe contient tous les pairs dont l'identifiant commence par i et il transmet donc cette requête à la fois aux pairs de e°_i, e°_{i0} et e°_{i1}. Les pairs concernés peuvent donc mettre à jour leur arbre de fragmentation ;
- si un pair appartenant à e°_{i0} reçoit une requête de rafraîchissement, il la transmet également à tout son groupe. Selon son point de vue, le groupe contient tous les pairs dont l'identifiant commence par $i0$: les pairs correspondants sont soit dans e°_{i0}, soit dans e°_i :



(a) P transmet la requête de rafraîchissement contenant l'identifiant de sa feuille locale 001, l'identifiant de la feuille distante 100 et la valeur aléatoire Δ. Chaque pair retransmet également cette requête (non illustré ici pour des raisons de lisibilité).



(b) Après le rafraîchissement, tous les pairs de e°_0 et e°_{10} ont mis à jour leur arbre de fragmentation. Les pairs de e°_{11} ne connaissent pas la valeur Δ utilisée.

FIGURE 4.4 – Rafraîchissement du groupe e°_0 avec le groupe e°_{10}.

- si un de ces pairs est honnête et appartient au groupe $\mathring{e}_i$, alors il reçoit cette requête et la transmet à tous les autres membres du groupe $\mathring{e}_i$ qui peuvent ainsi mettre à jour leur arbre de fragmentation ;
- sinon, tous ces pairs sont dans $\mathring{e}_{i0}$ ou les pairs restants dans $\mathring{e}_i$ sont malhonnêtes. Les pairs honnêtes restant dans $\mathring{e}_i$, qui devraient appartenir au groupe $\mathring{e}_{i1}$, ne sont pas informés du rafraîchissement. Cependant, ces pairs en retard pensent toujours appartenir à $\mathring{e}_i$ et pourraient réaliser des signatures partielles avec l'ancienne version de e_i. Dans ce cas spécifique où tous les pairs du groupe immédiatement supérieur devraient être dans le groupe frère, un pair recevant une requête de rafraîchissement applique une tolérance et retransmet la requête au groupe supérieur.

Dans tous les cas, les arbres de fragmentation restent cohérents après un rafraîchissement, au sens de la propriété 7.

4.2.4 Concurrence avec un processus de signature

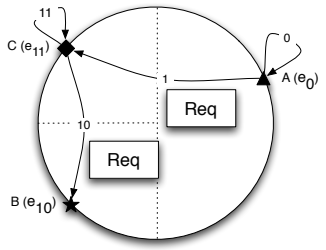
Le rafraîchissement peut interférer avec le processus de signature distribuée. En effet, lors d'une signature, les différents pairs sont appelés successivement et, si le partage de la clé évolue pendant le processus, certains pairs ont pu répondre avec des versions non compatibles de leur fragment. Dans l'exemple présenté figure 4.5, il est possible que le groupe $\mathring{e}_0$ réalise sa signature partielle une seconde après le groupe $\mathring{e}_{10}$ et, si le rafraîchissement a été effectué durant cette seconde, la fragmentation est valide mais la signature globale est erronée. La fréquence de ce cas dépend de la fréquence des opérations de rafraîchissement et du temps nécessaire à réaliser une certification complète. Nous proposons deux possibilités pour limiter ce problème :

1. même sans une synchronisation forte, les pairs peuvent être munis d'une horloge proche. Dans ce cas, les pairs conservent les anciennes versions de leurs fragments quelques dizaines de secondes et utilisent le fragment correspondant à l'heure de la requête initiale de certification, qui est incluse dans la requête et donc commune à tous les pairs participant ;
2. un pair ayant juste rafraîchi son fragment peut également retourner les deux possibilités de signature, à charge pour le pair initial de discriminer la bonne version en fonction de la clé publique du réseau connue.

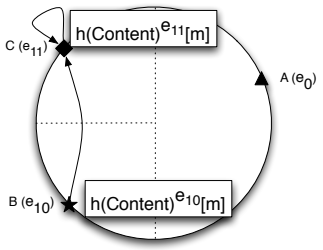
4.2.5 Utilisation

Afin de rester stockables et transférables, les arbres de fragmentation doivent être représentés de manière compacte. Leur génération initiale permet de les calculer intégralement à partir de la connaissance de la racine ; ensuite, chaque rafraîchissement contraint à stocker un chemin de la racine vers une feuille. Un trop grand nombre de rafraîchissements rendrait ces arbres coûteux à stocker et à transférer et un attaquant pourrait exploiter cette limite. Le nombre d'opérations de rafraîchissement permises doit donc être limité.

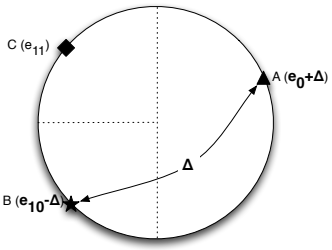
Dans le même temps, si un attaquant connaît un fragment e_i et si le nouveau groupe $\mathring{e}_{i1}$ rafraîchit avec un autre groupe contenant également un attaquant complice, alors les attaquants peuvent déduire la nouvelle version de e_{i1}. En effet, les attaquants connaissent l'ancienne version de e_{i1} et la valeur Δ utilisée



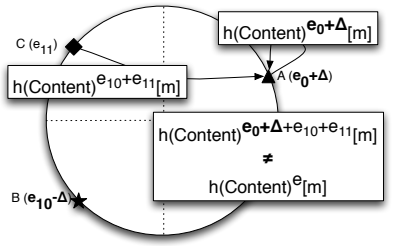
(a) A transmet la requête Req à signer



(b) B et C réalisent leurs signatures partielles avec e_{10} et e_{11}



(c) Un rafraîchissement est réalisé entre le groupe $\overset{\circ}{e_0}$ et le groupe $\overset{\circ}{e_{10}}$: Δ est ajouté à e_0 et soustrait de e_{10}



(d) A signe avec son nouveau fragment $e_0+\Delta$ et la signature complète est incorrecte

FIGURE 4.5 – Concurrence entre un rafraîchissement et un processus de signature distribuée. La signature finale est incorrecte, bien que tous les participants soient honnêtes.

pour rafraîchir. Le nombre d'opérations de rafraîchissement doit donc également garantir avec une forte probabilité qu'au moins une opération est réalisée avec un groupe de pairs honnêtes. Nous calculons d'abord que 10 opérations de rafraîchissement après une division garantissent la confidentialité du nouveau fragment. Nous expliquons ensuite que cette limitation peut être obtenue en limitant chaque pair à une seule opération de rafraîchissement et montrons comment appliquer cette limite.

4.2.5.1 Nombre d'opérations par groupe

Si k est le ratio d'attaquants et g la taille d'un groupe de fragmentation, la probabilité qu'un groupe soit constitué uniquement de pairs honnêtes vaut $P_{HonestGroup} = (1-k)^g$. Dans un système idéal, où les fragments ne sont connus que dans leurs groupes respectifs, les attaquants connaissent donc un fragment avec une probabilité $1 - P_{HonestGroup}$. Dans le système présenté, un attaquant peut connaître un fragment s'il est dans le bon groupe de fragmentation ou s'il connaît toutes les valeurs aléatoires utilisées pour le rafraîchir, ce qui arrive si chaque groupe tiers utilisé pour rafraîchir contient au moins un attaquant. Nous notons que cette description correspond à une majora-

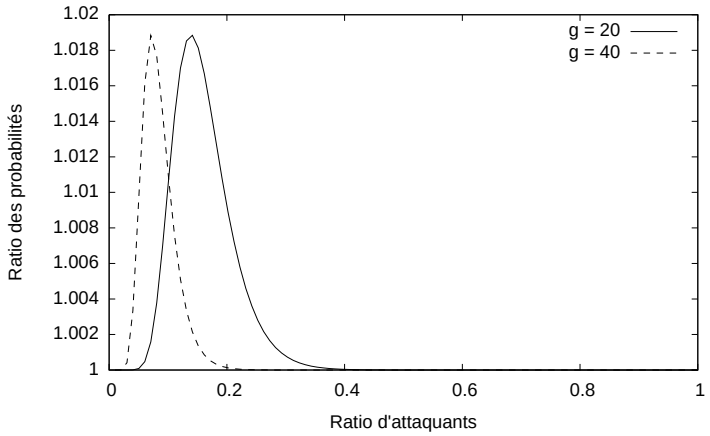


FIGURE 4.6 – Rapport entre les probabilités qu'un groupe d'attaquants connaisse un fragment dans un système idéal et dans notre système, pour des groupes de 20 ou 40 membres et après 20 rafraîchissements.

tion de la probabilité d'attaque, car les attaquants ont également besoin de connaître le fragment parent. Avec r opérations de rafraîchissement, la probabilité qu'au moins un rafraîchissement soit échangé avec un groupe sans attaquant vaut $P_{HonestRefresh} = 1 - (1 - P_{HonestGroup})^r$. Dans notre système, un groupe d'attaquants connaît donc un fragment donné avec une probabilité $1 - (P_{HonestGroup} \times P_{HonestRefresh})$, qui ne dépend pas de la taille du réseau.

Nous considérons le rapport entre les probabilités qu'un groupe d'attaquants connaisse un fragment dans notre système et un système idéal, qui vaut :

$$\boxed{P_{\frac{réel}{idéal}} = \frac{1 - (P_{HonestGroup} \times P_{HonestRefresh})}{1 - P_{HonestGroup}}}$$

D'après la figure 4.6, vingt opérations de rafraîchissement permettent de maintenir ce rapport entre 1 et $1,02$ pour des groupes composés de 20 à 40 membres, comme proposé dans ce mémoire. Nous considérons qu'une différence de 2% par rapport au cas idéal est satisfaisante et que vingt rafraîchissements permettent donc d'apporter une confidentialité suffisante aux fragments. Chaque groupe doit donc initialiser 10 rafraîchissements après une division, puisqu'il en reçoit également 10 en moyenne.

4.2.5.2 Nombre d'opérations par pair

Considérons maintenant que chaque pair est limité à une unique opération de rafraîchissement, d'abord dans le cas où aucun pair ne quitte le réseau. Dans ce cas, chaque groupe est créé avec $g_{min} = 20$ membres et existe jusqu'à être composé de $g_{max} = 40$ membres : 20 nouveaux membres rejoignent donc le système et si chaque pair peut lancer un rafraîchissement, le groupe peut lancer 20 opérations de rafraîchissement. Dans un cas réel, des pairs quittent puis rejoignent le réseau, ce qui change deux points :

– les pairs qui se connectent sont soit nouveaux soit anciens, et un ancien pair peut avoir déjà utilisé son opération de rafraîchissement ;

– beaucoup plus de 20 pairs joignent un groupe pendant sa vie (car d'autres pairs quittent le réseau), ce qui assure avec une grande probabilité que certains de ces pairs sont nouveaux.

Cependant, un cas critique sans rafraîchissement possible peut arriver si un groupe e°_{i0} passe de 20 à 40 membres uniquement avec d'anciens pairs qui ont déjà utilisé leur rafraîchissement. Ce cas peut seulement arriver si ce groupe était divisé en e°_{i00} et e°_{i01}, puis a fusionné en e°_{i0} puis $\hat{e}_i$ avant de se diviser à nouveau. Cela correspond à une diminution de plus de la moitié des pairs dans cette zone de l'espace, ce qui est rare et peut donc être géré si les pairs économisent leurs opérations de rafraîchissement. Quand un groupe se divise, les pairs attendent un délai aléatoire avant de lancer leur opération de rafraîchissement, puis, dès que 10 opérations ont été lancées, ils gardent leur possibilité pour un usage ultérieur. Les anciens pairs rejoignant le réseau peuvent donc toujours posséder des opérations de rafraîchissement.

Afin de limiter chaque pair à une unique opération de rafraîchissement, les générations de la feuille distante r et de la valeur aléatoire Δ sont contraintes. Le générateur aléatoire utilise comme graine la signature (unique et non devinable par un tiers, par définition) d'une donnée prédéfinie commune à tout le système, en utilisant la clé secrète du pair. La requête de rafraîchissement contient ensuite cette signature et chaque autre pair peut vérifier les valeurs aléatoires générées. Par exemple, pour un nœud A et si la donnée prédéfinie commune à tous les pairs vaut z, le générateur aléatoire utilise la signature de z avec la clé secrète de A comme graine. A génère la feuille distante r et Δ à partir de ce générateur et transmet la signature de z aux autres pairs. Muni du même générateur, chaque pair peut (1) vérifier que cette signature est bien celle de A et (2) générer la feuille distante et la valeur Δ associées. La feuille locale de A est contrainte par l'identifiant certifié de A.

4.3 Opération de fusion

Quand un groupe e°_{i0} est composé de moins de g_{min} membres, ce groupe doit fusionner avec e°_{i1} dans un groupe $\hat{e}_i$ connaissant $e_i = e_{i0} + e_{i1}$ (le cas de e°_{i1} est symétrique). Durant une opération de fusion, les pairs doivent télécharger l'arbre de fragmentation du groupe frère et nous expliquons d'abord comment le faire de manière efficace. Nous décrivons ensuite le protocole de fusion. Nous traitons séparément le cas de la fusion de groupes de niveau différents. Nous montrons ensuite que les arbres de fragmentation restent cohérents tant que le nombre d'attaquants dans le réseau est inférieur à 20%. Enfin, nous expliquons que l'opération de fusion ne pose pas de problème de concurrence avec une signature distribuée.

4.3.1 Téléchargement des arbres de fragmentation

Le téléchargement d'un arbre de fragmentation doit être efficace, tout en empêchant un attaquant de fournir des données erronées. Nous définissons une fonction de hachage w, inspirée des *arbres de hash de Merkle* [Mer87], afin d'attacher un contrôle d'intégrité à chaque nœud de l'arbre. Pour chaque nœud interne défini e_i, $w(e_i) = h(e_i, w(e_{i0}), w(e_{i1}))$ et pour chaque feuille ou nœud implicite (non défini) e_i, $w(e_i) = h(e_i)$, h étant une fonction de hachage telle

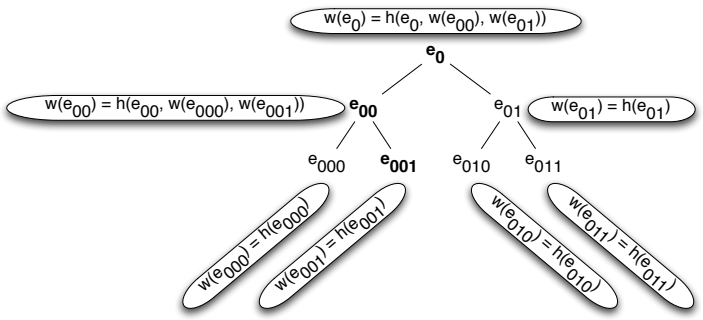


FIGURE 4.7 – Schéma de hachage utilisé pour vérifier l'intégrité d'un arbre de fragmentation.

que SHA-1. La connaissance de $w(e_i)$ permet de vérifier l'intégrité de tout le sous-arbre $\widehat{e_i}$. Ce schéma de hachage est illustré figure 4.7.

Le téléchargement d'un arbre de fragmentation est réalisé en trois étapes – obtention du contrôle d'intégrité de l'arbre, obtention de l'arbre et mise à jour de l'arbre – que nous détaillons dans cette sous-section :

1. le téléchargeur demande à tous les pairs du groupe frère e°_{i1} la valeur de $w(e_{i1})$ permettant de contrôler l'intégrité de l'arbre. Comme nous supposons que la majorité des pairs sont honnêtes, le téléchargeur peut utiliser la valeur retournée par plus de la moitié des pairs comme valeur honnête et ce $w(e_{i1})$ permet d'identifier une version de l'arbre.
   Si un rafraîchissement est en cours, il est possible que des pairs honnêtes renvoient des valeurs différentes. Dans ce cas, si aucune valeur n'est retournée par plus de la moitié des membres, le téléchargeur réessaie jusqu'à obtenir un $w(e_{i1})$ majoritaire ;
2. le téléchargeur demande les différentes parties de l'arbre défini par $w(e_{i1})$ à différents pairs. Chaque pair contacté peut connaître :
   - la version caractérisée par $w(e_{i1})$, auquel cas il retourne la partie demandée accompagnée des valeurs w pour chaque nœud ;
   - une version plus ancienne de l'arbre, s'il n'a pas encore reçu un rafraîchissement, auquel cas il retourne une erreur ;
   - une version plus récente, auquel cas il a gardé en mémoire les quelques dernières opérations de rafraîchissement qu'il peut défaire pour retourner l'arbre demandé.

   Grâce à la connaissance de $w(e_{i1})$, le téléchargeur peut vérifier l'intégrité de l'arbre à partir de la racine et redemander les parties corrompues ;
3. le téléchargeur met à jour les feuilles de son arbre de fragmentation pour connaître la toute dernière version de cet arbre. Il envoie les condensats et les identifiants de chaque feuille définie à chaque autre pair qui répond avec les feuilles mises à jour ou nouvellement définies. Le téléchargeur applique les modifications proposées par plus de la moitié des pairs.

À l'issue de cette dernière étape, le téléchargeur connaît un arbre de fragmentation à jour et vérifié. Le coût de cette opération de téléchargement est évalué dans le chapitre 5.

4.3.2 Protocole de fusion

Nous décrivons le protocole de fusion d'abord dans le cas ordinaire de la fusion de groupes de même niveau puis dans le cas de la fusion de groupes de niveaux différents.

4.3.2.1 Fusion de groupes de même niveau

Chaque pair décide localement de fusionner dans deux cas :

1. quand son propre groupe de fragmentation est composé de moins de g_{min} membres ;
2. quand le groupe de fragmentation frère est composé de moins de g_{min} membres. Dans ce second cas, le groupe frère a besoin de fusionner et le pair doit également décider de fusionner.

Une fois qu'un pair a décidé de fusionner, il demande périodiquement aux pairs du groupe frère leur arbre de fragmentation. Les pairs répondent à cette requête s'ils ont également décidé localement de fusionner. Finalement, chaque pair fusionne et obtient le même arbre de fragmentation $\widehat{e_i}$ dont la racine e_i vaut $e_{i0} + e_{i1}$. Cette opération est illustrée figure 4.8

4.3.2.2 Fusion de groupes de niveaux différents

Il est également possible qu'un groupe e°_{i0} composé de moins de g_{min} membres doive fusionner mais que le groupe frère e°_{i1} n'existe pas, si ce groupe s'est divisé en e°_{i10} et e°_{i11}. Sur la figure 4.8(b), ce cas de figure correspond à $\mathring{e}_0$ qui devrait fusionner avec $\mathring{e}_1$, qui n'existe pas.

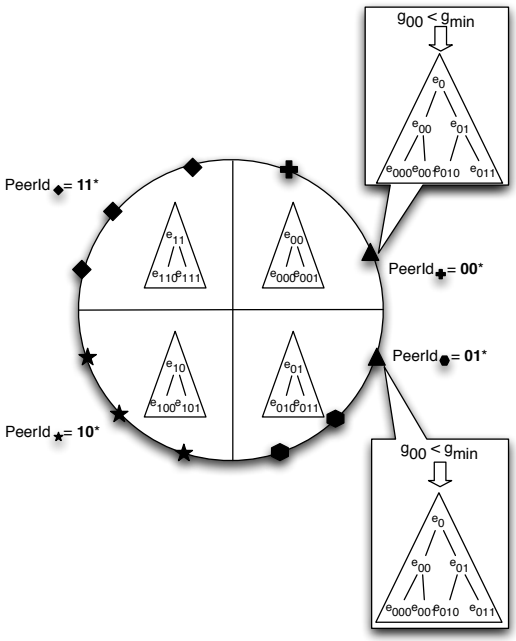
Dans ce cas, e°_{i0} doit fusionner, mais, si e°_{i10} et e°_{i11} ne surveillaient que leurs frères, ils n'accepteraient pas de fusionner avec e°_{i0}. Chaque pair doit donc surveiller les pairs de chaque groupe avec lequel il pourrait fusionner. Dans ce cas :

- e°_{i0} surveille e°_{i1}, même si e°_{i1} n'existe pas : ce n'est pas un problème car chaque fils de e°_{i1} est composé de plus de g_{min} membres et e°_{i0} détecte donc suffisamment de pairs ;
- e°_{i10} et e°_{i11} se surveillent mutuellement et surveillent également les frères de leurs ancêtres, dont e°_{i0}.

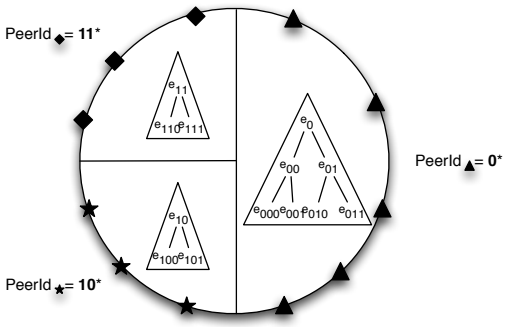
Les groupes e°_{i10} et e°_{i11} peuvent ainsi détecter que e°_{i0} doit fusionner, fusionner dans e°_{i1} puis fusionner avec e°_{i0} dans $\mathring{e}_i$. Sur la figure 4.8(b), les pairs de e°_{10} surveillent à la fois le nombre de pairs dans e°_{11} et dans $\mathring{e}_0$. Les pairs d'un groupe e°_{101} surveilleraient les pairs de e°_{100}, e°_{11} et $\mathring{e}_0$.

Cette possibilité doit être gérée, mais, comme sa probabilité est faible (les pairs ont des identifiants uniformément répartis), elle reste marginale et n'affecte pas le ratio de pairs devant coopérer pour obtenir une signature. Le nombre de groupes de fragmentation à surveiller est logarithmique par rapport à la taille du réseau et n'est, par exemple, que de 15 pour un réseau contenant un million de pairs. De plus, comme nous utilisons Kademlia, la surveillance des groupes de fragmentation est équivalente à la maintenance des *k-buckets*[2] utilisés pour le routage et est donc une opération sans coût tant que g_{min} est plus petit que la taille des *k-buckets*. Dans la pratique, la taille usuelle des *k-buckets* est de 20, ce qui correspond à notre valeur de g_{min}.

2. Les parties de l'*overlay* surveillées correspondent chacune à un k-bucket.



(a) Avant la fusion, certains pairs constatent que la taille du groupe e°_{00} est inférieure à 3 et fusionnent localement leur groupe.



(b) Après la fusion, tous les pairs de e°_{0} ont fusionné.

FIGURE 4.8 – Fusion des groupes e°_{00} et e°_{01}, avec $g_{min} = 3$.

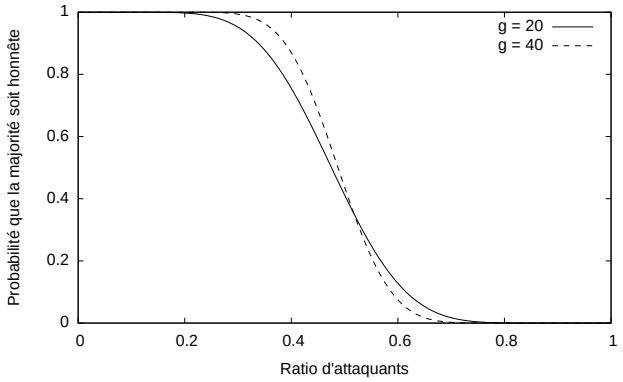


FIGURE 4.9 – Probabilité que le groupe frère soit composé d'une majorité de pairs honnêtes en fonction du ratio d'attaquants. Les groupes sont composés de $g = 20$ ou $g = 40$ membres.

4.3.3 Cohérence des arbres de fragmentation

Durant la fusion, chaque pair obtient un contrôle d'intégrité de l'arbre téléchargé de la part de tous les membres du groupe frère et choisit la réponse majoritaire. Tant que la majorité des membres du groupe frère est honnête, le contrôle d'intégrité utilisé est le bon et chaque pair obtient donc le bon arbre frère. Nous calculons la probabilité que la majorité des membres du groupe frère soient honnêtes.

Soit k le ratio de pairs malhonnêtes. Un pair est attaquant avec une probabilité k et honnête avec une probabilité $1 - k$: i pairs parmi g sont honnêtes avec une probabilité $C_i^g \times (1-k)^i \times k^{g-i}$. La majorité des pairs d'un groupe constitué de g membres est honnête avec la probabilité :

$$\boxed{P_{MajHonest} = \sum_{i=\lfloor \frac{g}{2} \rfloor + 1}^{g} C_i^g \times (1-k)^i \times k^{g-i}}$$

Cette probabilité est tracée sur la figure 4.9 pour des groupes constitués de $g_{min} = 20$ à $g_{max} = 40$ membres. Dans ces conditions, cette probabilité est très élevée en dessous de 20% d'attaquants, auquel cas chaque pair obtient bien le bon contrôle d'intégrité et le bon arbre de fragmentation.

En l'absence de cette attaque, les anciens pairs des e_{i0}° et e_{i1}° connaissent les mêmes arbres $\widehat{e_{i0}}$ et $\widehat{e_{i1}}$. Le seul nouveau nœud est la racine du nouvel arbre $\widehat{e_i}$ qui est entièrement déterminée par $\widehat{e_{i0}}$ et $\widehat{e_{i1}}$: les arbres de fragmentation restent cohérents après une fusion tant que le réseau contient moins de 20% d'attaquants.

4.3.4 Concurrence avec un processus de signature

L'opération de fusion peut exhiber les mêmes désynchronisations temporaires que la division (des pairs dans $\mathring{e}_i$, e_{i0}° et e_{i1}°) ; ces désynchronisations n'impactent pas le processus de signature et sont gérées de la même manière.

4.4 Opération de connexion

Lorsqu'un pair se connecte, il doit obtenir un fragment de la clé secrète du réseau. Nous décrivons le protocole de connexion, analysons la cohérence des arbres de fragmentation et vérifions la concurrence de cette opération avec une signature distribuée.

4.4.1 Protocole de connexion

Quand un pair rejoint le réseau, il rejoint le groupe de fragmentation incluant son identifiant et obtient l'état de son groupe dans l'ordre suivant :

1. l'identifiant de groupe, en demandant à ses voisins ;
2. la racine de l'arbre de fragmentation, en demandant à chaque pair de son groupe. À partir de ce moment, il peut participer aux opérations de signature distribuée ;
3. le reste de l'arbre de fragmentation, téléchargé en arrière-plan comme expliqué précédemment.

4.4.2 Cohérence des arbres de fragmentation

Durant la connexion, le nouveau pair obtient un contrôle d'intégrité de l'arbre téléchargé de la part de tous les membres de son groupe et choisit la réponse majoritaire. Tant que la majorité des membres du groupe est honnête, le contrôle d'intégrité utilisé est le bon et le nouveau pair obtient donc le bon arbre de fragmentation.

Tout comme dans le cas de la fusion, les arbres de fragmentation restent donc cohérents après la connexion d'un nouveau pair tant que le réseau contient moins de 20% d'attaquants.

4.4.3 Concurrence avec un processus de signature

Un nouveau pair peut recevoir une demande de signature partielle durant l'obtention de l'état de son groupe. Si le pair possède déjà son fragment, il répond à la requête ; sinon, il peut soit retarder sa réponse, soit renvoyer un message d'erreur, auquel cas un autre pair est interrogé.

4.5 Opération de déconnexion

Lorsqu'un pair se déconnecte, son groupe de fragmentation doit le retirer de la liste des membres. Nous décrivons le protocole de déconnexion, la cohérence des arbres de fragmentation et la concurrence de cette opération avec une signature distribuée.

4.5.1 Protocole de déconnexion

Quand un pair quitte le réseau, il sauvegarde son arbre de fragmentation courant afin de n'avoir à télécharger que les mises à jour lorsqu'il se reconnecte. Les autres pairs du groupe détectent qu'il ne répond plus et le retirent de leur liste des membres.

4.5.2 Cohérence des arbres de fragmentation

La déconnexion d'un pair n'agit pas sur la cohérence des arbres de fragmentation.

4.5.3 Concurrence avec un processus de signature

Le pair quittant le réseau peut avoir été sollicité pour une signature partielle juste avant son départ. Dans ce cas, le pair l'ayant sollicité n'obtient pas de réponse et, après un délai, interroge un autre pair du même groupe.

4.6 Preuve de la confidentialité et de l'intégrité

Dans cette section, nous prouvons que les algorithmes de maintenance proposés garantissent la confidentialité des fragments (propriété 5) et l'intégrité du partage de la clé (propriété 6).

Nous notons :

- P l'ensemble des pairs ;
- P_0 l'ensemble des pairs dont l'identifiant commence par 0 (ils ne connaissent que des fragments commençant par 0) ;
- P_1 l'ensemble des pairs dont l'identifiant commence par 1 (ils ne connaissent que des fragments commençant par 1) ;
- $P_x(e_i)$ l'ensemble des pairs P_0 ou P_1 dont l'identifiant commence différemment de e_i (ils sont de l'autre côté de l'arbre de fragmentation) ;
- E est l'ensemble des fragments courants ;
- $\forall p \in P$, $p \triangleleft x$ dénote le fait que le pair p connaît la valeur x ;
- $\forall p \in P$, $p \not\triangleleft x$ dénote le fait que le pair p ne connaît pas la valeur x.

4.6.1 Confidentialité des fragments

Nous prouvons ici le théorème de confidentialité des fragments :

Théorème 1 (*Confidentialité des fragments*)

Une signature valide ne peut être obtenue que par la coopération d'un membre de chaque groupe de fragmentation.

Afin de prouver ce théorème, nous prouvons deux lemmes énonçant respectivement que les membres d'un groupe sont les seuls à connaître la valeur courante de leur fragment (lemme 1) et que les anciennes versions des fragments ne peuvent pas être utilisées avec le partage de clé actuel (lemme 3)

4.6.1.1 Les membres d'un groupe sont les seuls à connaître la valeur courante de leur fragment

Nous prouvons les lemmes 1 et 2 conjointement. Le théorème 1 utilise le lemme 1 et une partie de la preuve du lemme 1 utilise le lemme 2.

Lemme 1

$$\forall e_i \in E, \forall p \in P \backslash \mathring{e_i}, p \not\triangleleft e_i$$

Pour tous les groupes $\mathring{e_i}$, les pairs qui n'appartiennent pas au groupe $\mathring{e_i}$ ne connaissent pas la valeur actuelle de e_i.

Lemme 2

$$\forall e_i \in \widehat{e}, \forall p \in P_x(e_i), p \not\triangleleft e_i$$

Les pairs situés d'un côté de l'arbre de fragmentation global ne connaissent aucun nœud de cet arbre situé de l'autre côté de la racine.

Hypothèse 1

À l'initialisation du système, les lemmes 1 et 2 sont vérifiés.

Cette hypothèse se justifie aisément. Si la clé secrète du réseau est générée de manière centralisée par une unique entité fondant le réseau, cette entité connaît le fragment e et uniquement ce fragment, puisqu'il n'y a qu'un seul fragment et un seul groupe. Si la clé secrète est générée de manière distribuée, chaque groupe de fragmentation est créé à partir de zéro et ne connaît donc que son arbre de fragmentation respectif.

À partir de ce point de départ, nous prouvons les lemmes 1 et 2 par induction sur l'état du système. Supposons que ces deux lemmes sont vérifiés à un instant donné, les pairs peuvent alors exécuter une division, un rafraîchissement, une fusion, une connexion ou une déconnexion. Lors de la connexion, un pair obtient son fragment courant et, lors de la déconnexion, un pair n'obtient pas d'information : ces deux opérations n'influent pas sur la confidentialité des fragments et nous ne nous intéressons donc qu'aux trois opérations de maintenance principales (division, rafraîchissement et fusion).

Division du groupe $\mathring{e_i}$

Après la division du groupe $\mathring{e_i}$, la confidentialité est assurée si les deux propriétés suivantes sont valides :

- $\forall p \in P \backslash \mathring{e_{i0}}, p \not\triangleleft e_{i0}$;
- $\forall p \in P \backslash \mathring{e_{i1}}, p \not\triangleleft e_{i1}$.

Nous nous intéressons au groupe $\mathring{e_{i0}}$ (le cas de $\mathring{e_{i1}}$ est symétrique). D'après le lemme 2, nous savons que :

$$\forall p \in P_x(e_{i0}), p \not\triangleleft e_{i0}$$

Après une division, les pairs de $\mathring{e_{i0}}$ doivent exécuter un rafraîchissement avec $\mathring{e_j}$ choisi de l'autre côté de l'arbre pour la valeur Δ qui suit la contrainte suivante :

$$\forall p \in P \backslash \mathring{e_{i0}} \backslash \mathring{e_j}, p \not\triangleleft \Delta$$

En l'absence de collusion, nous en déduisons que :

$$\begin{array}{c} \forall p \in \mathring{e_{i1}}, p \not\triangleleft \Delta \\ \forall p \in \mathring{e_j}, p \not\triangleleft e_{i0} \quad (1) \\ \forall p \in P \backslash \mathring{e_{i0}}, p \not\triangleleft \Delta \vee p \not\triangleleft e_{i0} \end{array}$$

Enfin, le fragment e_{i0} devient $e_{i0} + \Delta$.
D'où :

$$\boxed{\forall p \in P \backslash \mathring{e_{i0}}, p \not\triangleleft e_{i0}, \text{ ce qui implique également } \forall p \in P_x(e_{i0}), p \not\triangleleft e_{i0}}$$

En cas de collusion, un attaquant peut être présent dans $\mathring{e_{i1}}$ et le rafraîchissement peut être exécuté avec un groupe $\mathring{e_j}$ contenant également un attaquant. Dans ce cas, la propriété (1) $\forall p \in \mathring{e_j}, p \not\triangleleft e_{i0}$ n'est pas valide. Cependant, aussitôt que $\mathring{e_{i0}}$ rafraîchit avec un groupe entièrement honnête, ce qui arrive avec une forte probabilité tant qu'il existe un nombre suffisant de pairs honnêtes dans le réseau, cette propriété (1) est validée et les attaquants ne connaissent plus la nouvelle valeur de e_{i0}. Plusieurs opérations de rafraîchissement après une division garantissent avec une forte probabilité que l'un des rafraîchissements est honnête.

Les lemmes 1 et 2 sont donc toujours valides après une division suivie d'un rafraîchissement.

Rafraîchissement entre $\mathring{e_i}$ et $\mathring{e_j}$
Après le rafraîchissement entre $\mathring{e_i}$ et $\mathring{e_j}$, la confidentialité est assurée si les deux propriétés suivantes sont valides :

- $\forall p \in P \backslash \mathring{e_i}, p \not\triangleleft e_i$;
- $\forall p \in P \backslash \mathring{e_j}, p \not\triangleleft e_j$.

Nous nous intéressons au groupe $\mathring{e_i}$ (le cas de $\mathring{e_j}$ est symétrique). D'après le lemme 1 :

$$\forall p \in P \backslash \mathring{e_i}, p \not\triangleleft e_i$$

Après le rafraîchissement, e_i vaut $e_i + \Delta$.
D'où :

$$\boxed{\forall p \in P \backslash \mathring{e_i}, p \not\triangleleft e_i, \text{ ce qui implique également } \forall p \in P_x(e_i), p \not\triangleleft e_i}$$

Les lemmes 1 et 2 sont donc toujours valides après un rafraîchissement.

Fusion entre $\mathring{e_{i0}}$ et $\mathring{e_{i1}}$
Après la fusion entre $\mathring{e_{i0}}$ et $\mathring{e_{i1}}$, la confidentialité est assurée si :

$$\forall p \in P \backslash \mathring{e_i}, p \not\triangleleft e_i$$

D'après le lemme 1 :

$$\begin{array}{c} \forall p \in P \backslash \mathring{e_{i0}}, p \not\triangleleft e_{i0} \\ \forall p \in P \backslash \mathring{e_{i1}}, p \not\triangleleft e_{i1} \end{array}$$

Lors de la fusion, un nouveau fragment e_i est créé à partir des deux fragments e_{i0} et e_{i1}. D'où :

$$\boxed{\forall p \in P \backslash \mathring{e_i}, p \not\triangleleft e_i, \text{ ce qui implique également } \forall p \in P_x(e_i), p \not\triangleleft e_i}$$

Les lemmes 1 et 2 sont donc toujours valides après une fusion.

4.6.1.2 Les anciennes versions des fragments ne peuvent pas être utilisées avec le partage de clé actuel

Le théorème 1 utilise également le lemme 3 :

Lemme 3

Les anciennes versions des fragments ne peuvent pas être utilisées conjointement aux fragments actuels pour obtenir une signature valide.

Nous montrons qu'il n'est possible d'utiliser ni des fragments ancêtres (fragments de groupes s'étant divisés), ni d'anciennes versions des fragments actuels.

Soit E l'ensemble des fragments actuels, la somme des éléments de E vaut e. La signature distribuée utilise la relation

$$h(o)^e[m] = h(o)^{\sum_{e_x \in E} e_x}[m] = \prod_{e_x \in E} h(o)^{e_x}[m]$$

Pour substituer un fragment ancêtre e_x à un ou des fragments dans ce calcul, les fragments retirés doivent avoir une somme e_x. Les seuls fragments candidats à cette opération sont les fragments dérivés depuis e_x (tous les fragments des groupes s'étant créés par divisions successives du groupe $\mathring{e_x}$). Cependant, ces fragments ont été modifiés lors des rafraîchissements et leur somme ne vaut plus e_x : le fragment e_x ne peut pas leur être substitué dans un calcul de signature.

Pour substituer un fragment obsolète e_x à la version actuelle de e_x, les deux versions doivent avoir la même valeur, ce qui n'est pas le cas par définition : seule la version actuelle peut être utilisée pour le calcul de signature.

Le lemme 3 est donc toujours vérifié.

4.6.1.3 Déduction du théorème de confidentialité

Étant donné le lemme 1, les pairs appartenant à un groupe $\mathring{e_i}$ sont les seuls à connaître le fragment e_i.

Étant donné le lemme 3, il n'est pas possible d'utiliser d'anciennes versions de fragments pour obtenir une signature.

Tant que le nombre de pairs honnêtes reste suffisamment élevé, une signature ne peut donc être obtenue que par la coopération d'un membre de chaque groupe (théorème 1). Un grand nombre d'attaquants peut corrompre la sécurité du rafraîchissement et invalider le théorème 1, ce que nous avons traité dans la section 4.2.■

4.6.2 Intégrité du partage

Nous prouvons ici le théorème d'intégrité du partage :

Théorème 2 (*Intégrité du partage*)

À chaque instant, la somme des fragments vaut la clé secrète e.

Hypothèse 2

À l'initialisation du système, le théorème 2 est vérifié.

La génération et la distribution initiale de la clé secrète vérifient l'hypothèse 2. À partir de ce point de départ, nous prouvons le théorème 2 par induction sur l'état du système. Supposons que le théorème est vérifié à un instant donné, les pairs peuvent alors exécuter une division, un rafraîchissement ou une fusion.

Si les pairs exécutent une division, la somme des deux fragments créés vaut l'ancien fragment. Comme les groupes sont cohérents, chaque pair connaissait le même fragment et choisit bien la même valeur pour les nouveaux fragments : la somme globale des fragments reste donc inchangée et vaut toujours e.

Si les pairs exécutent un rafraîchissement, une valeur aléatoire est ajoutée à un fragment et soustraite d'un autre : la somme globale ne change pas et vaut toujours e.

Si les pairs exécutent une fusion, le nouveau fragment vaut la somme des deux anciens fragments. Si le groupe formé est bien cohérent, chacun des pairs de ce nouveau groupe connaît les mêmes valeurs pour les anciens fragments et le nouveau fragment vaut bien la somme des deux fragments fusionnés : la somme globale reste la même.

Tant que les groupes sont cohérents, ce qui est le cas tant que le nombre d'attaquants reste suffisamment faible (moins de 20% avec les tailles de groupes proposées), la somme des fragments vaut e à chaque instant (théorème 2). ■

Les opérations de maintenance proposées garantissent donc la confidentialité des fragments et l'intégrité du partage de la clé secrète du réseau tant que le nombre de pairs honnêtes reste suffisamment élevé.

Résumé

Dans ce chapitre, nous avons présenté la maintenance de la fragmentation de la clé de réseau. Les opérations proposées sont réalisées sans synchronisation et sans consensus byzantins. Pour cela, une nouvelle structure d'*arbres de fragmentation* a été introduite permettant de définir tous les fragments possibles. Le fragment d'un groupe est la racine de son arbre de fragmentation.

Dans la section 4.1, nous avons présenté l'opération de *division*. Chaque pair surveille la taille de son groupe courant et, lorsqu'il détecte que la taille du groupe dépasse la borne fixée g_{max}, il divise localement son groupe. Le pair choisit alors l'un des deux sous-arbres de son arbre de fragmentation actuel. Chaque pair divise le groupe lorsqu'il détecte sa taille supérieure à g_{max}, et ce à des instants différents. Finalement, chaque pair a divisé son groupe et l'ancien groupe n'existe plus.

Dans la section 4.2, nous avons présenté l'opération de *rafraîchissement*. Pour rafraîchir deux fragments ensemble, une valeur aléatoire est échangée entre deux groupes de fragmentation. Cette opération peut impliquer un groupe en cours de division sans rompre l'intégrité de la fragmentation : dans ce cas, un seul des deux groupes en train d'être créés est affecté par le rafraîchissement.

Dans la section 4.3, nous avons présenté l'opération de *fusion*. Chaque pair surveille la taille de son groupe courant et des groupes avec lesquels il pourrait fusionner. Lorsqu'il détecte que son groupe ou le groupe frère est composé de moins de g_{min} membres, le pair commence à demander l'arbre de fragmentation du groupe frère et accepte de communiquer le sien aux pairs du groupe frère. Lorsque suffisamment de pairs ont décidé de fusionner, chaque pair obtient

l'arbre du groupe frère et peut recréer le nouveau fragment en additionnant les deux anciens fragments.

Dans la section 4.4, nous avons présenté l'opération de *connexion* d'un pair. Pour obtenir son arbre de fragmentation, un nouveau pair obtient d'abord un contrôle d'intégrité de cet arbre en interrogeant tous les membres de son groupe puis télécharge cet arbre sans redondance.

Dans la section 4.5, nous avons présenté l'opération de *déconnexion* d'un pair. Lorsqu'un pair quitte le réseau, les autres membres de son groupe détectent qu'il ne répond plus et le retirent de leur liste des membres du groupe.

Enfin, dans la section 4.6, nous avons prouvé que les opérations de maintenance présentées garantissaient la confidentialité des fragments et l'intégrité du partage tant que le nombre de pairs honnêtes restait suffisamment élevé.

Ce travail a été publié à IEEE P2P 2009 [LMV09].

Chapitre 5

Implémentation et résultats expérimentaux

« Les chiffres sont comme les gens.
Si on les torture assez, on peut leur faire dire n'importe quoi. »
Anonyme

Dans ce chapitre, nous nous intéressons à l'implémentation de l'autorité de certification distribuée et à son évaluation.

Dans la section 5.1, nous décrivons l'implémentation réalisée en Java des algorithmes précédemment présentés. Cette implémentation utilise la bibliothèque pair-à-pair *Overlay Weaver*[1] [STS08] et est générique vis-à-vis du mécanisme de décision utilisé par l'autorité de certification.

Dans la section 5.2, nous présentons les résultats expérimentaux du modèle de partage de la clé et de sa mise en œuvre face à des attaquants, obtenus en utilisant le simulateur *PeerSim* [JMJV].

5.1 Implémentation

L'implémentation préliminaire du système proposé a été réalisée en utilisant la bibliothèque pair-à-pair *Overlay Weaver*. *Overlay Weaver* est écrit en Java et propose une abstraction du routage pair-à-pair et des DHT en utilisant plusieurs protocoles structurés dont Chord, Pastry et Kademlia. Son développement est actif et il propose un système stable et utilisable depuis plusieurs années.

La signature distribuée utilise essentiellement la fonction de routage vers un identifiant donné, fonction qui retourne la liste des pairs voisins de cet identifiant. Cette fonction permet à l'algorithme de signature de trouver les pairs éligibles à une requête récursive ainsi que de découvrir un groupe distant, dans le cas du rafraîchissement par exemple. Notre implémentation est générique dans le sens où elle ne dépend ni du contenu du certificat ni du mécanisme de décision utilisé pour choisir de coopérer ou non. L'implémentation est schématisée figure 5.1.

1. http ://overlayweaver.sourceforge.net

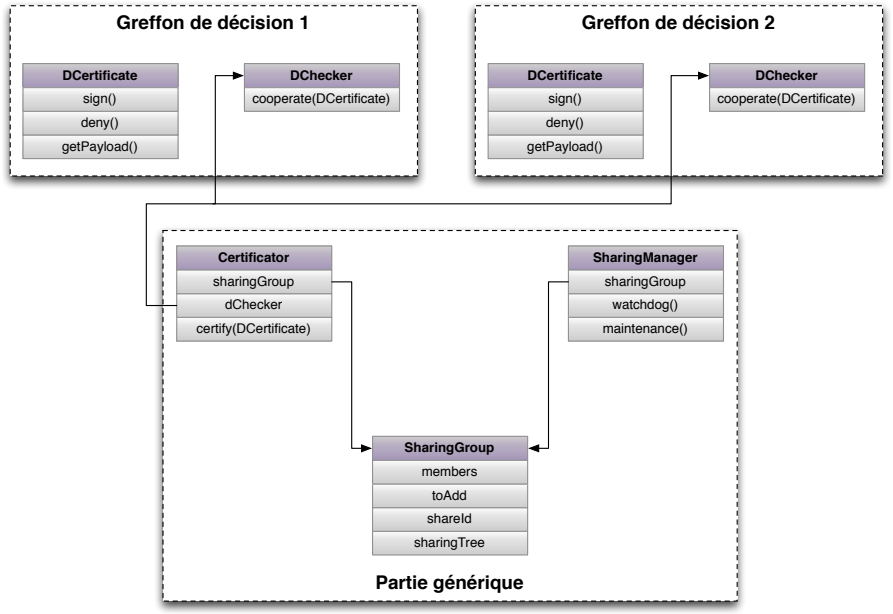


FIGURE 5.1 – Schéma de l'implémentation.

5.1.1 Architecture générale

Chaque pair décrit son état dans un objet de type *SharingGroup*. Cet objet contient principalement les champs suivants :

- *members* contient la liste des membres présents dans le groupe. L'élimination des pairs déconnectés est actuellement réalisée par un mécanisme propre de *watchdog*, mais il est prévu d'utiliser des primitives d'*Overlay Weaver* pour remplir ce rôle ;
- *toAdd* contient les pairs annoncés comme présents par d'autres pairs, mais avec lesquels aucune communication n'a encore eu lieu. Un pair ajouté à cette liste est sollicité pour vérifier sa présence et recevra des notifications du pair local ; il n'est en revanche pas comptabilisé pour évaluer la taille du groupe ;
- *shareId* est l'identifiant du groupe courant et est partagé entre l'algorithme de signature et les algorithmes de maintenance ;
- *sharingTree* est l'arbre de fragmentation courant et est également partagé entre l'algorithme de signature et les algorithmes de maintenance.

Cette classe fournit également des fonctions de communication permettant de la multidiffusion vers un groupe.

5.1.2 Algorithme de signature

L'implémentation de la signature distribuée est réalisée dans la classe *Certificator* en utilisant plusieurs *threads*. Un *thread* s'occupe de recevoir les messages et de les rediriger vers les threads traitants, avec un thread traitant par requête de certification. Un thread traitant chaque requête, tous les messages sont lus

de manière bloquante et écrits dans des files FIFO afin de simplifier l'implémentation. Le *thread* traitant réalise l'algorithme de signature :

- si le pair refuse de coopérer, il renvoie une signature vide ainsi que la liste des pairs de son groupe. Le pair ayant initialisé la requête peut ainsi rapidement demander aux autres pairs de ce même groupe s'ils acceptent de coopérer à cette signature ;
- si le pair accepte de coopérer et possède le fragment demandé, il réalise la signature partielle et retourne le résultat ;
- si le pair accepte de coopérer mais ne possède pas le fragment demandé, il doit continuer l'algorithme récursivement. Une des deux requêtes générées est renvoyée localement, l'autre est envoyée à $nbAsks$ pairs (la sélection des pairs honnêtes n'est pas encore implémentée). Les réponses reçues sont ensuite comparées pour obtenir le chiffrement honnête, si les pairs acceptent de coopérer. Les deux signatures partielles sont combinées et la combinaison est retournée au pair appelant.

Enfin, chaque demande de signature partielle est enregistrée et chaque signature partielle obtenue est mise en cache. En effet, si chaque signature partielle est demandée à plusieurs pairs, un même pair peut être appelé plusieurs fois pour la même requête. Dans ce cas, seule la première requête est exécutée et les suivantes reçoivent la même réponse, soit déjà en cache, soit dès que disponible.

5.1.3 Algorithmes de maintenance

La maintenance est réalisée dans la classe *SharingManager* avec deux *threads* d'arrière plan.

Le premier *thread* est un *watchdog* qui maintient à jour la liste des membres du groupe. Ce *thread* envoie des *pings* aux autres membres et nettoie la liste des pairs déconnectés.

Le second *thread* est responsable de la réponse aux *pings*, de l'insertion initiale et de la maintenance du groupe. La réponse aux *pings* est systématique, quel que soit l'état du pair. L'insertion est réalisée en recherchant le groupe contenant l'identifiant du pair, puis en demandant les informations du groupe. Enfin, une fois le pair inséré, la maintenance est exécutée périodiquement pour décider et exécuter la division, la fusion ou le rafraîchissement.

5.1.4 Greffons de décision

L'application de la certification distribuée est découplée de sa maintenance à travers un système générique permettant d'insérer de nouveaux greffons de décision. Les greffons de décision implémentent deux interfaces *DCertificate* et *DChecker* :

- *DCertificate* représente un certificat. Une classe implémentant cette interface doit fournir des méthodes pour définir la signature, marquer le certificat comme refusé et obtenir la charge utile du certificat ;
- *DChecker* représente le module de décision. Cette interface ne contient qu'une seule fonction, *cooperate*, qui prend en paramètre un certificat *DCertificate* et rend une décision de coopération sous la forme d'un booléen.

Cette approche permet d'utiliser simplement la certification distribuée au niveau applicatif. Ainsi, pour fournir à chaque pair un nom intelligible unique,

la classe implémentant *DCertificate* doit contenir un champ *name* comme charge utile et la classe implémentant *DChecker* doit tester si ce nom a déjà été utilisé dans le réseau, en le recherchant.

5.1.5 Évaluation de l'implémentation

Nous avons déployé cette implémentation sur la plate-forme de test à grande échelle PlanetLab [2]. PlanetLab regroupe plus de 800 machines réparties à travers le monde et permet de tester des algorithmes pair-à-pair à moyenne échelle.

Nous avons réalisé de premiers tests sur le temps nécessaire pour obtenir une signature distribuée. Le réseau est constitué d'environ 300 pairs et forme 100 groupes de fragmentation constitués de $g_{min} = 2$ à $g_{max} = 4$ membres : la petite taille des groupes permet de décomposer le réseau en autant de groupes que dans un réseau de 3000 pairs avec des groupes de 20 à 40 membres. Avec 100 groupes, l'obtention d'une signature distribuée prend environ 10 secondes ; dans ce cas, l'arbre déployé sur les groupes durant le processus de signature a une hauteur comprise entre 6 et 7.

Dans un réseau de 10 000 pairs avec des groupes de 20 à 40 membres, il y a entre 250 et 500 groupes et un arbre de signature a une hauteur comprise entre 8 et 9, soit deux étapes supplémentaires : nous pouvons supposer qu'une signature dans ce cas prendrait environ 30% de temps supplémentaire, soit un total de 13 secondes. Chaque pair demandant peu de signatures (admission dans le réseau, obtention d'un nom), un pair peut raisonnablement attendre ce délai avant d'obtenir son certificat.

5.2 Résultats expérimentaux

Dans cette section, nous présentons les résultats obtenus sur le modèle de partage de la clé, l'algorithme de signature distribuée et les protocoles de maintenance du partage de la clé. Les simulations sont réalisées avec *PeerSim* [JMJV], un simulateur pair-à-pair extensible écrit en Java. Les groupes de fragmentation son composés de $g_{min} = 20$ à $g_{max} = 40$ membres, ce qui induit $0.025 < t < 0.05$ (une signature n'est possible que par la collaboration de $2,5$ à 5% des pairs). Nous évaluons :

- le modèle présenté dans le chapitre 2 (probabilité de révéler la clé secrète ou de perdre un fragment). Nous montrons que le modèle tolère environ 20% d'attaquants ;
- l'algorithme de signature distribuée proposé dans le chapitre 3 en présence d'attaquants (probabilité de succès en fonction du nombre de pairs ou du ratio d'attaquants). Nous montrons que sa version sans redondance est peu robuste alors que sa version avec sélection automatique des pairs tolère un pourcentage important d'attaquants pour un coût qui n'est que légèrement supérieur ;
- les algorithmes de maintenance proposés dans le chapitre 4 (sécurité des fragments et quantité de données à transférer). Nous montrons que les fragments restent aussi longs que la clé secrète et qu'un attaquant ne peut donc pas les deviner par recherche exhaustive. Nous montrons également que les arbres de fragmentation sont stockables et transférables.

2. http ://www.planet-lab.eu

Dans ces simulations, nous ne prenons pas en compte les attaques sur les pairs. Par exemple, une attaque de type *ver* pourrait permettre à un attaquant de contrôler les machines d'un grand nombre d'utilisateurs. Nous considérons ici que ces attaques sont hors de notre champ d'investigation et les pairs sont donc honnêtes ou attaquants uniquement en fonction du choix de l'utilisateur.

5.2.1 Modèle de l'autorité de certification distribuée

Nous évaluons ici le modèle de partage de la clé présenté dans le chapitre 2. Nous nous intéressons à la probabilité pour un groupe d'attaquants d'obtenir tous les fragments (et donc pouvoir calculer la clé secrète) ou de posséder tous les pairs d'un groupe de fragmentation (et donc rendre la signature distribuée indisponible).

5.2.1.1 Probabilité de révéler la clé secrète

La probabilité $P_{revealed}$, calculée dans le chapitre 2, exprime la probabilité qu'un attaquant soit présent dans chaque groupe de fragmentation. Nous évaluons cette probabilité théorique et la comparons aux résultats expérimentaux correspondants.

La figure 5.2 illustre la probabilité de cette attaque en fonction du ratio d'attaquants, dans un réseau composé de 10 000 pairs et pour des groupes de taille comprise entre 20 et 40 membres. La courbe expérimentale est bornée par les deux courbes théoriques correspondant à l'ensemble des groupes composés soit de 20, soit de 40 membres. Dans ce cas, l'attaque est théoriquement possible à partir de 10% d'attaquants mais n'apparaît dans la pratique qu'à partir de 20% d'attaquants. La courbe expérimentale est en fait presque superposée à la

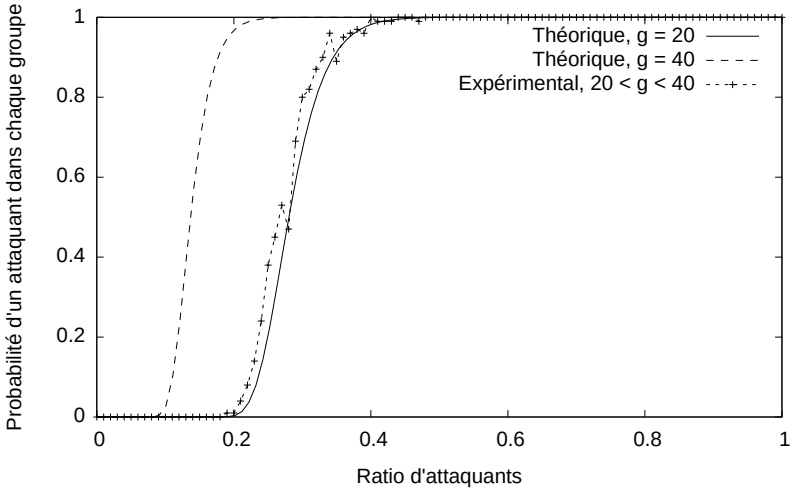


FIGURE 5.2 – Probabilité qu'il y ait un attaquant dans chaque groupe en fonction du ratio d'attaquants, dans un réseau composé de 10 000 pairs. Les deux courbes théoriques correspondent à des groupes de fragmentation composés respectivement de 20 et 40 membres et la courbe expérimentale est réalisée avec des groupes de taille comprise entre 20 et 40 membres.

courbe théorique avec des groupes de 20 membres : ce sont en effet ces petits groupes qui sont les plus difficiles à atteindre pour l'attaquant.

La figure 5.3 illustre le ratio d'attaquants nécessaire pour mener à bien cette attaque dans un réseau composé de 100 à 100 000 pairs. Ce ratio dépasse très vite les 5% pour quelques milliers de nœuds. La courbe théorique utilisant des groupes de 20 membres, qui dans la figure 5.2 est très proche des résultats expérimentaux, dépasse même très vite le seuil de tolérance des 20%. Un réseau de 100 000 pairs tolère au moins 15% d'attaquants (cas théorique avec $g = 40$) et probablement même 30% (cas théorique avec $g = 20$, correspondant au cas expérimental dans la figure 5.2).

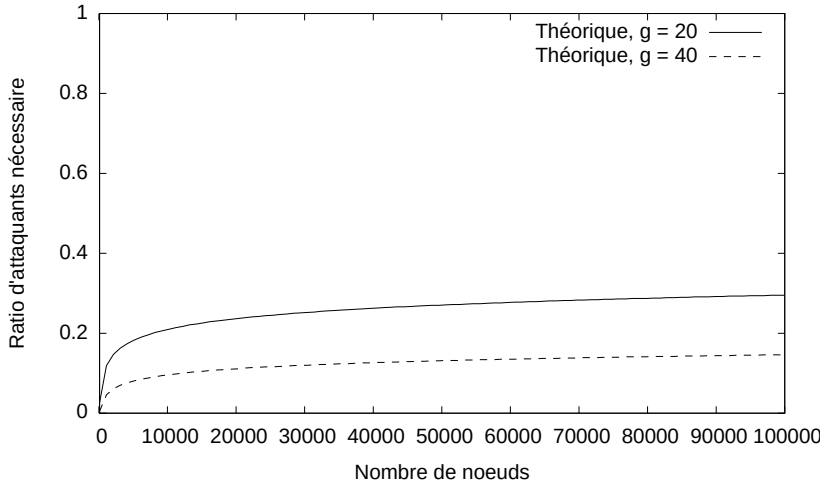


FIGURE 5.3 – Ratio d'attaquants nécessaire pour obtenir tous les fragments avec une probabilité supérieure à $0,01$ dans un réseau composé de 100 à 100 000 pairs.

5.2.1.2 Probabilité de perdre un fragment

La probabilité P_{broken}, calculée dans le chapitre 2, exprime la probabilité qu'un groupe de fragmentation ne soit composé que d'attaquants, auquel cas le fragment du groupe est perdu. Nous évaluons cette probabilité théorique et la comparons aux résultats expérimentaux correspondants.

La figure 5.4 illustre la probabilité de cette attaque en fonction du ratio d'attaquants, dans un réseau composé de 10 000 pairs et pour des groupes de taille comprise entre 20 et 40 membres. La courbe expérimentale est bornée par les deux courbes théoriques correspondant à l'ensemble des groupes composés soit de 20, soit de 40 membres. Dans ce cas, cette attaque est possible à partir de 60% d'attaquants. La courbe expérimentale est de nouveau presque superposée à la courbe théorique avec des groupes de 20 membres : ce sont en effet ces petits groupes qui sont le plus tôt contrôlés par des attaquants.

La figure 5.5 illustre le ratio d'attaquants nécessaire pour contrôler un groupe de fragmentation dans un réseau composé de 100 à 100 000 pairs. Ce ratio reste élevé et 50% d'attaquants ne peuvent pas contrôler un groupe entier dans un réseau composé de 100 000 pairs.

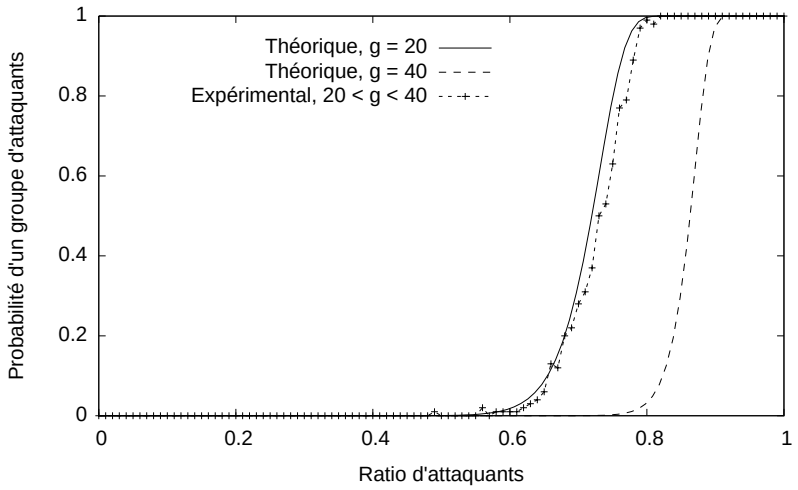


FIGURE 5.4 – Probabilité qu'un groupe ne soit composé que d'attaquants en fonction du ratio d'attaquants, dans un réseau composé de 10 000 pairs. Les deux courbes théoriques correspondent à des groupes de fragmentation composés respectivement de 20 et 40 membres et la courbe expérimentale est réalisée avec des groupes de taille comprise entre 20 et 40 membres.

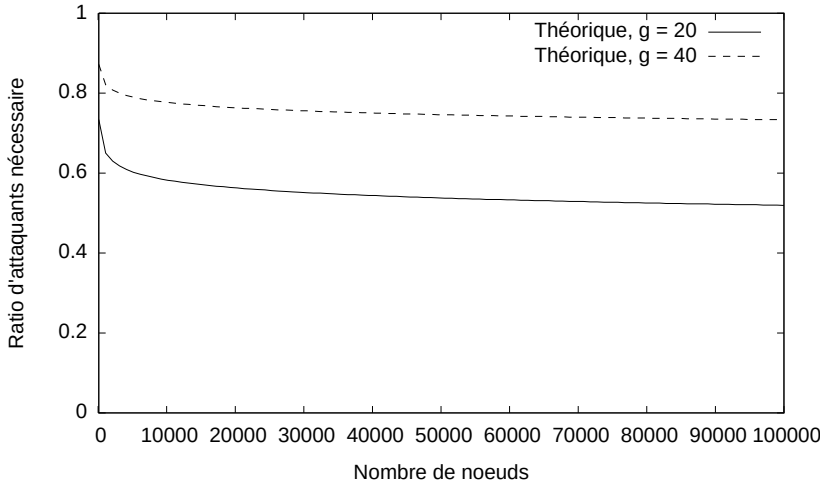


FIGURE 5.5 – Ratio d'attaquants nécessaire pour détruire un fragment avec une probabilité supérieure à $0,01$ dans un réseau composé de 100 à 100 000 pairs.

5.2.2 Signature distribuée

Nous présentons ici les résultats obtenus sur l'algorithme de signature distribuée, avec et sans le mécanisme de sélection des pairs. Nous étudions la probabilité de succès de la signature distribuée en fonction de la taille du réseau, puis en fonction du ratio d'attaquants. Nous montrons que la sélection des pairs permet d'obtenir une probabilité de réussite meilleure en interrogeant beaucoup moins de pairs : la sélection permet une signature robuste et efficace. Les résultats expérimentaux sont complétés par la probabilité de succès $P_{success}$ calculée dans le chapitre 3. Le contexte d'évaluation est le suivant :

- *avec la sélection des pairs*, chaque pair interroge $nbAsksMax$ pairs lors de la première signature partielle qu'il demande pour un fragment donné ; ensuite, ce pair n'interroge qu'un seul pair ayant rendu une réponse majoritaire pour ce fragment. Lorsque des pairs se déconnectent, les pairs les utilisant pour signer sélectionnent un nouveau pair pour les remplacer. Enfin, les pairs signalent les réponses reçues qu'ils pensent erronées et ce signal est utilisé par chaque pair pour réinitialiser sa sélection de pairs, le cas échéant. Dans ce cas avec la sélection des pairs, nous étudions la probabilité de réussite de l'algorithme de signature distribuée, ainsi que le nombre de pairs interrogés en moyenne pour chaque signature partielle ;
- *sans la sélection des pairs*, chaque pair interroge $nbAsks$ pairs pour obtenir chaque signature partielle. Nous étudions, dans ce cas, la probabilité de réussite de l'algorithme de signature.

5.2.2.1 Modélisation du réseau

La dynamicité des pairs influe sur le mécanisme de sélection et nous faisons l'hypothèse que les pairs sont connectés 10% du temps. D'un côté, cette hypothèse est optimiste pour des systèmes de partage de fichiers où les utilisateurs se déconnectent après avoir téléchargé un fichier ; d'un autre côté, cette hypothèse est pessimiste pour un service de voix sur IP où les utilisateurs restent en ligne la plupart du temps pour pouvoir recevoir des appels. Si les utilisateurs ne sont connectés que 10% du temps, alors le réseau contient 10 fois moins de pairs que le nombre global d'utilisateurs du système. Dans les simulations présentées, le nombre de pairs correspond toujours au nombre de pairs connectés et non au nombre d'utilisateurs total.

Chaque utilisateur demande trois certificats, ce qui correspond au cumul des deux applications proposées dans les chapitres 6 et 7. Nous considérons que le réseau met une année à atteindre une taille de 10 000 pairs, soit 100 000 utilisateurs. Dans ce cas, 300 000 certificats sont demandés en une année, soit 820 certificats par jour et 34 par heure.

Le comportement de chaque pair est modélisé par deux lois de Poisson :

- un pair reste en ligne une durée aléatoire correspondant à une loi de Poisson de moyenne 3 heures (100 certificats) ;
- un pair reste ensuite déconnecté une durée aléatoire correspondant à une loi de Poisson de moyenne 27 heures (900 certificats).

Si le réseau grandit plus rapidement et si chaque pair reste connecté le même temps, alors chaque pair réalise un plus grand nombre de signatures partielles avant de se déconnecter et peut donc affiner sa sélection des pairs honnêtes. Au contraire, une croissance moins rapide induit que plus de signatures partielles

sont demandées alors qu'aucun pair n'a encore été sélectionné, auquel cas la requête est transmise avec redondance et donc moins efficacement ; cependant, cette perte d'efficacité est compensée par la diminution du nombre de signatures demandées.

5.2.2.2 Probabilité de réussite en fonction du nombre de pairs

Dans la figure 5.6, nous faisons varier le nombre de pairs dans le réseau avec un pourcentage d'attaquants constant de 10% (ce qui est un pourcentage important, puisque nous ne considérons pas les attaques de type *ver*). Comme analysé dans le chapitre 3, plus le réseau est grand, plus il est difficile d'obtenir une signature valide. Avec 500 pairs, 20% des signatures distribuées sans sélection avec $nbAsks = 1$ réussissent ; avec plus de 1500 pairs, aucune signature avec $nbAsks = 1$ ne réussit.

L'ajout de redondance sans sélection, avec $nbAsks = 5$, permet de gérer plus d'attaquants mais la probabilité de succès décroît tout de même nettement avec la taille du réseau.

Avec sélection et en interrogeant initialement $nbAsksMax = 5$ pairs, les résultats sont meilleurs et permettent une forte probabilité de succès (presque 0.9) dans un réseau de 10 000 pairs contenant 10% d'attaquants. Le mécanisme de retour arrière, qui signale aux pairs incriminés les mauvais chiffrements détectés et leur permet de réinitialiser les mauvaises sélections, permet d'atteindre une meilleure robustesse que l'interrogation systématique de 5 pairs sans sélection. De plus, le nombre de pairs interrogés en moyenne est inférieur à $1,2$. La sélection permet d'atteindre une robustesse élevée pour un coût presque optimal.

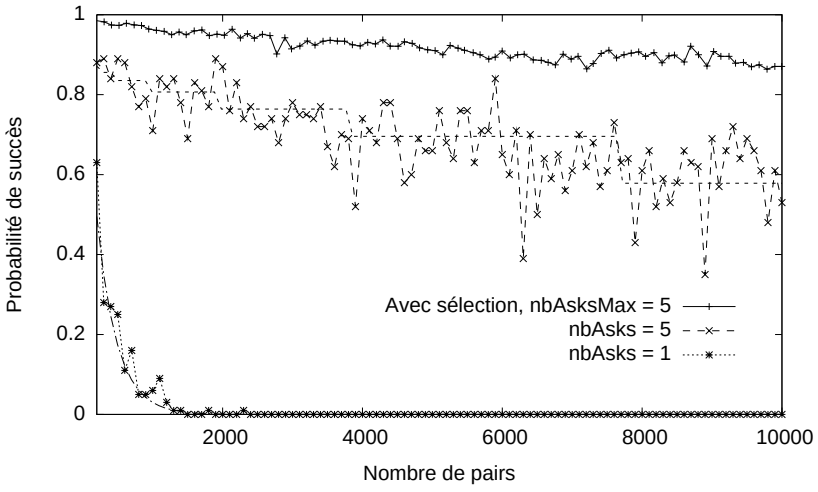


FIGURE 5.6 – Pourcentage de succès des différents algorithmes en fonction du nombre de pairs, dans un réseau contenant 10% d'attaquants. Les courbes théoriques sont également tracées.

5.2.2.3 Probabilité de réussite en fonction du pourcentage d'attaquants

Dans la figure 5.7, nous faisons varier le pourcentage d'attaquants à taille réseau constante de 5000 pairs, ce qui représente une valeur médiane de la figure 5.6. Avec 1% d'attaquants, seulement 20% des certifications avec $nbAsks = 1$ réussissent. Avec plus de 2% d'attaquants, l'algorithme de signature n'est plus utilisable avec $nbAsks = 1$.

L'ajout de redondance sans sélection, avec $nbAsks = 5$, permet de gérer plus d'attaquants mais la probabilité de succès décroît nettement à partir de 10% d'attaquants.

Avec sélection, la probabilité de réussite est plus élevée. Avec sélection parmi $nbAsksMax = 5$ pairs, la probabilité est plus élevée mais décroît tout de même nettement à partir de 10% d'attaquants ; avec sélection parmi $nbAsksMax = 7$ pairs, la probabilité de succès reste supérieure à 0.8 avec 15% d'attaquants. La sélection permet d'utiliser une valeur $nbAsksMax$ importante, puisque, dans les deux cas présentés ici, le nombre moyen de pairs interrogés est toujours inférieur à $1,2$; sans sélection, le coût de la redondance est rapidement prohibitif.

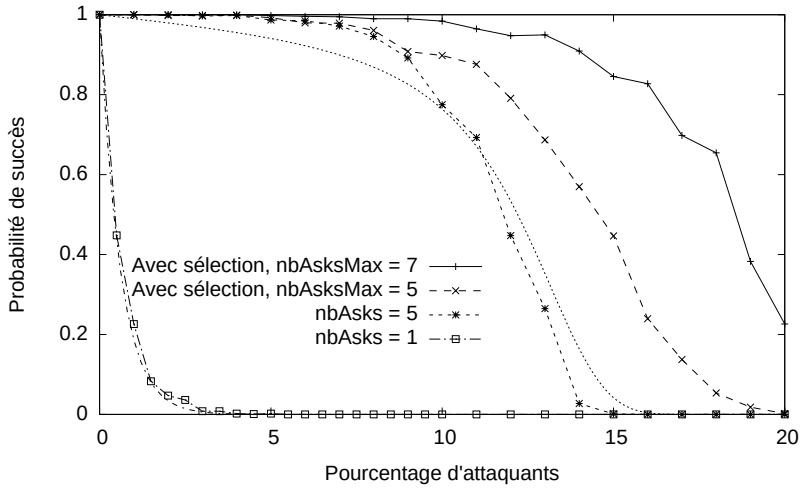


FIGURE 5.7 – Pourcentage de succès des différents algorithmes en fonction du pourcentage d'attaquants, dans un réseau composé de 5000 pairs. Les courbes théoriques sont également tracées.

5.2.3 Maintenance de la fragmentation de la clé

Dans cette section, nous évaluons la sécurité et l'efficacité des opérations de maintenance proposées. Nous nous intéressons plus précisément à la taille des fragments et des arbres de fragmentation en fonction de la taille du réseau. La clé secrète du réseau fait 1024 bits et toutes les valeurs aléatoires utilisées pour diviser ou rafraîchir font également 1024 bits.

Comme précédemment, nous faisons l'hypothèse que les pairs sont connectés 10% du temps. Comme chaque utilisateur peut avoir utilisé son opération de

rafraîchissement, nous considérons que 10 fois plus de rafraîchissements que le nombre de pairs ont été réalisés.

Nous évaluons d'abord la taille des fragments de clé et montrons qu'ils restent presque aussi longs que la clé secrète du réseau, quand le réseau grandit. Nous analysons ensuite la taille des groupes de fragmentation et montrons qu'ils peuvent être facilement sauvegardés et transférés.

5.2.3.1 Taille des fragments

À l'initialisation, tous les fragments sont générés à partir de l'exposant secret e. Chaque fragment e_{i0} est généré aléatoirement sur 1024 bits (entre 0 et 2^{1023}) et chaque fragment e_{i1} est défini tel que $e_{i1} = e_i - e_{i0}$. Ensuite, les opérations de rafraîchissement modifient ces fragments en ajoutant ou soustrayant des valeurs aléatoires sur 1024 bits (entre 0 et 2^{1023}). La figure 5.8 représente la taille des fragments en fonction du nombre de pairs. Pour chaque taille de réseau, 10 fois plus de rafraîchissements ont été exécutés. Nous pouvons constater que la taille moyenne des fragments est indépendante du nombre de pairs et que la taille minimale ne diminue que légèrement quand le réseau grandit. Même dans un très grand réseau, un attaquant ne peut pas deviner un fragment.

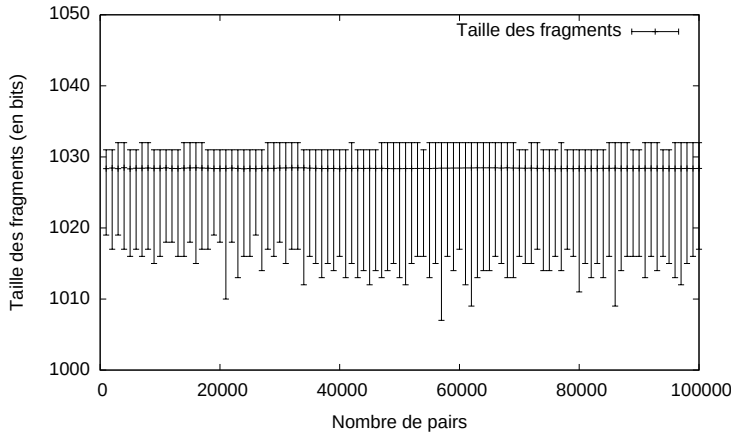


FIGURE 5.8 – Taille des fragments en fonction du nombre de pairs. Pour chaque nombre de pairs, le tic inférieur indique la taille du plus petit fragment, le tic du milieu la taille moyenne et le tic supérieur la taille du plus grand fragment.

5.2.3.2 Taille des arbres de fragmentation

Chaque pair doit stocker un arbre de fragmentation dont certains nœuds sont explicitement définis et nous évaluons ici l'espace occupé par ces arbres. Nous considérons que les identifiants de groupe sont au plus de 40 bits, ce qui correspond à des arbres binaires de fragmentation de hauteur bornée par 40 (au plus $2^{40} \approx 10^{12}$ groupes, qui représentent au moins 20×10^{12} pairs). Les fragments font 1024 bits.

Comme les pairs sont connectés 10% du temps, les arbres de fragmentation contiennent les rafraîchissements de dix fois plus d'utilisateurs que de pairs connectés. Pour des groupes de 40 membres, 800 feuilles de l'arbre sont ainsi

définies : les 400 définies par les utilisateurs du groupe, connectés ou non, et les 400 reçues à partir d'autres groupes. Nous obtenons ainsi une borne supérieure sur la taille des arbres de $800 \times 40 \times 1024$ bits, soit environ 4 Mo.

Cependant, dans la pratique, beaucoup de nœuds internes des arbres de fragmentation sont des ancêtres communs de plusieurs feuilles définies, auquel cas la définition d'une feuille implique la définition de moins de 40 nœuds internes. À partir des simulations présentées figure 5.9, nous constatons que les arbres de fragmentation occupent moins d'1 Mo en moyenne. Ils peuvent donc être facilement téléchargés.

Nous pouvons également voir que la taille des arbres de fragmentation est presque indépendante de la taille du réseau. En fait, le nombre de feuilles définies est constant et ne dépend que de la taille des groupes ; seule la hauteur de l'arbre change et diminue même quand le réseau grandit. La représentation choisie passe donc à l'échelle. Comme les pairs sont répartis uniformément dans les groupes, chaque groupe tend à se diviser au même moment, ce qui explique les oscillations de la courbe.

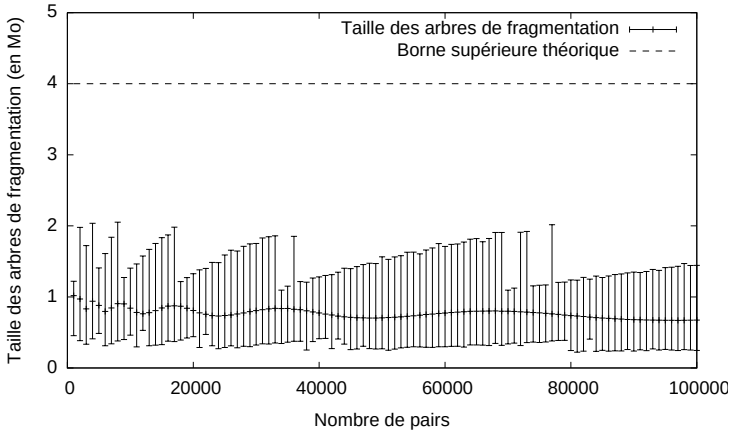


FIGURE 5.9 – Taille des arbres de fragmentation en fonction du nombre de pairs. Pour chaque nombre de pairs, le tic inférieur indique la taille du plus petit arbre, le tic du milieu la taille moyenne et le tic supérieur la taille du plus grand arbre.

Enfin, pour mettre à jour un arbre de fragmentation après l'avoir téléchargé, un pair doit envoyer les identifiants et condensats de toutes les feuilles définies à tous les membres de son groupe. S'il y a 800 feuilles identifiées sur 40 bits avec des condensats sur 160 bits, le téléchargeur envoie $(40 + 160) \times 800$ bits, soit 20 ko, à chaque autre pair (entre 20 et 40). Ce processus pourrait être optimisé en envoyant le condensat de plusieurs feuilles à la fois pour détecter rapidement quelles parties de l'arbre ont changé.

Résumé

Dans ce chapitre, nous avons présenté l'implémentation de l'autorité de certification distribuée ainsi que son évaluation expérimentale.

Dans la section 5.1, nous avons décrit l'implémentation réalisée. Cette implémentation est générique vis-à-vis du contenu du certificat et des critères utilisés

par chaque pair pour décider de coopérer ou non à une signature distribuée. Différentes applications peuvent ainsi utiliser le même partage de la clé. Cette implémentation a été testée sur PlanetLab avec environ 300 pairs.

Dans la section 5.2, nous avons évalué expérimentalement le modèle de partage de la clé, la signature distribuée et la maintenance de la fragmentation de la clé secrète. Le modèle de partage de la clé permet de tolérer environ 20% d'attaquants à partir de 5000 pairs. L'utilisation du mécanisme de sélection des pairs, proposé pour tolérer les pairs malveillants lors de la signature distribuée, permet en effet la signature dans un réseau contenant une proportion importante d'attaquants avec un coût faible. La maintenance de la fragmentation de la clé secrète conserve des fragments de taille importante, ce qui empêche un attaquant de pouvoir deviner un fragment non connu par essais successifs. Nous avons enfin montré que les arbres de fragmentation conservaient une taille raisonnable et pouvaient être stockés et transférés.

Deuxième partie

Applications de l'autorité de certification distribuée

Chapitre 6

Protection contre l'attaque sybile

« *On peut tromper une fois mille personnes,*
On peut tromper mille fois une personne,
Mais on ne peut pas tromper mille fois mille personnes. »
Émile, *in* La Cité de la peur

Dans ce chapitre, nous proposons une protection contre l'attaque sybile. Dans les réseaux pair-à-pair structurés, chaque pair a un identifiant unique choisi aléatoirement. Lors d'une attaque sybile, un attaquant trompe l'assignation des identifiants de pairs. Un attaquant crée un grand nombre d'identifiants quelconques ou un nombre plus restreint d'identifiants choisis. De plus, si un attaquant est capable de générer un grand nombre d'identifiants de pairs, il peut en choisir un sous-ensemble spécifique et mener une attaque ciblée. Un attaquant peut ainsi :

- contrôler une ressource et tous ses réplicats (censure, modification, espionnage des requêtes), en choisissant les identifiants de pairs sur lesquels sont placés cette ressource et ses réplicats ;
- contrôler les accès au réseau d'un pair honnête (attaque éclipse), en choisissant les identifiants de pairs que la victime utilise dans sa table de routage ;
- altérer les performances du réseau, en créant un grand nombre de pairs quelconques qui ne respectent pas le protocole attendu.

Une protection contre l'attaque sybile doit donc à la fois s'intéresser au nombre d'identifiants de chaque utilisateur et à l'aléa de ces identifiants. Nous définissons une telle protection de la façon suivante :

Définition 15 (*Protection contre l'attaque sybile*)

Une protection contre l'attaque sybile :
- *empêche un attaquant de générer un grand nombre d'identifiants de pairs ;*
- *contraint les identifiants des pairs à être choisis aléatoirement.*

Comme présenté dans l'état de l'art (section 1.2.1), nous distinguons trois approches principales contre l'attaque sybile, à savoir les autorités centralisées, les tests de ressources informatiques et les tests de ressources sociales. Dans ce chapitre, nous choisissons une protection sociale, qui nous paraît la plus équitable entre les pairs. Nous proposons un mécanisme de défense contre l'attaque sybile couplant *SybilGuard* [YKGF06a] et l'autorité de certification distribuée proposée en chapitre 2 :

- SybilGuard utilise les liens sociaux pour discriminer les pairs sybils des pairs honnêtes et fournit à chaque pair un oracle de décision pour évaluer l'honnêteté de chaque autre pair. SybilGuard contrôle ainsi le nombre de pairs sybils mais pas les identifiants de ces pairs ;
- en utilisant SybilGuard, l'autorité de certification distribuée attribue un certificat unique à chaque pair honnête et ce certificat est utilisé pour calculer l'identifiant du pair. L'autorité de certification distribuée contrôle les identifiants des pairs.

Nous avons vu qu'une autorité de certification était composée d'une partie décision (accepter ou non une requête de certification) et d'une partie signature. SybilGuard est utilisé ici pour la décision et la signature est réalisée avec l'algorithme de signature distribuée précédemment proposé dans le chapitre 2. Chaque pair utilise SybilGuard pour évaluer localement l'honnêteté d'un pair souhaitant rejoindre le réseau.

Dans la section 6.1, nous décrivons le fonctionnement de SybilGuard afin de pouvoir ensuite l'analyser de manière théorique et pratique dans les sections suivantes.

Dans la section 6.2, nous décrivons le processus d'obtention d'un certificat d'appartenance au réseau. En effet, chaque nouveau pair doit obtenir un certificat pour rejoindre le réseau et ce certificat est signé de manière distribuée par les pairs reconnaissant ce nouveau membre comme honnête. Un pair ne rejoint donc le réseau que si un ratio t des membres actuels le jugent honnête, chacun utilisant SybilGuard pour décider de sa coopération.

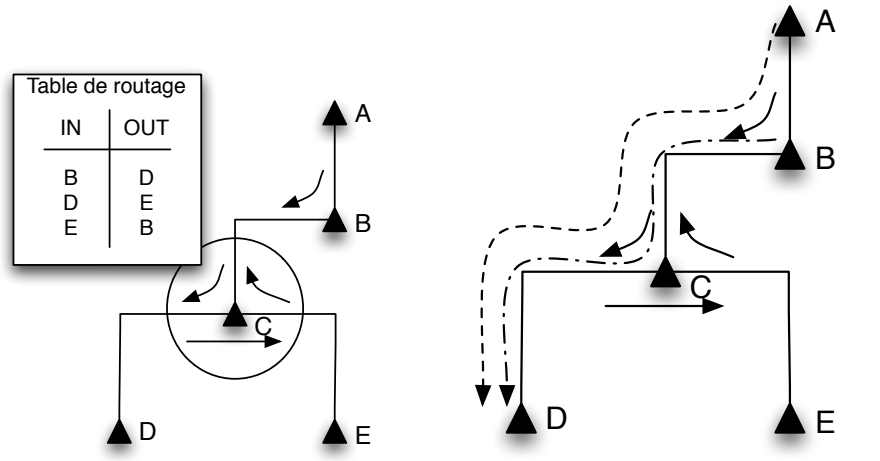
Enfin, dans les section 6.3 et 6.4, nous analysons la protection apportée par notre système et la comparons à SybilGuard seul, de manière à la fois théorique et expérimentale.

Définition 16 (*Pairs honnêtes, pairs sybils et attaquants physiques*)

- *Un pair* honnête *correspond à l'unique pair exécuté par un utilisateur donné ;*
- *Un pair* sybil *correspond à l'un des plusieurs pairs exécutés par un même attaquant physique ;*
- *Un* attaquant physique *est une personne physique unique exécutant plusieurs pairs sybils.*

6.1 SybilGuard

Dans [YKGF06a], Yu *et al.* proposent SybilGuard, un système de détection des pairs sybils basé sur les relations sociales. Notre solution s'appuie en partie sur SybilGuard et nous décrivons ce système dans cette section.



(a) Tables de routage de SybilGuard.

(b) Deux routes aléatoires partant de A et B. Une fois réunies, ces deux routes restent jointes.

FIGURE 6.1 – Routes aléatoires de SybilGuard. La table de routage de C est explicitée.

6.1.1 Graphe SybilGuard

SybilGuard est un système implanté sur chaque pair du réseau et permettant à chacun de ces pairs d'évaluer l'honnêteté d'un autre pair quelconque. Cette décision est basée sur un parcours particulier du graphe social.

Chaque pair maintient une liste d'amis qui sont ses voisins dans le graphe social. Cette liste représente des liens réels entre des personnes qui attestent de leur existence mutuelle. Les arcs ne sont pas dirigés et les relations sont donc considérées réciproques. Un nouvel utilisateur doit d'abord s'insérer dans le réseau SybilGuard avant de rejoindre le réseau pair-à-pair protégé.

Chaque SybilGuard possède plusieurs amis et donc plusieurs arcs : le pair crée une table de routage aléatoire par permutation aléatoire entre ces arcs. Ces tables de routage permettent de parcourir le graphe selon des *routes aléatoires*[1] ; ces routes aléatoires ont les deux propriétés de convergence et de traçabilité arrière et sont illustrées figure 6.1.

1. Les routes aléatoires ne sont pas des marches aléatoires. Les routes aléatoires sont aiguillées de manière déterministe selon les tables de routage générées aléatoirement la première fois, alors que les marches aléatoires sont redirigées aléatoirement à chaque saut.

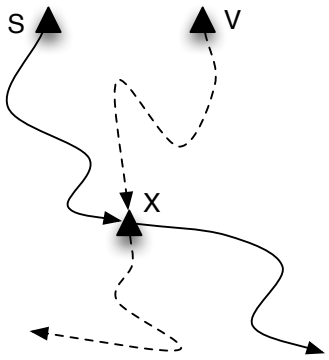


FIGURE 6.2 – Un vérifieur V accepte un pair suspect S si et seulement si deux de leurs routes aléatoires se croisent en un pair X.

Propriété 8 (*Convergence et Traçabilité arrière*)

Les routes aléatoires de SybilGuard ont les deux propriétés suivantes :

1. ***Convergence** : Deux routes qui entrent par le même arc ressortent par le même arc et restent ensuite jointes ;*
2. ***Traçabilité arrière** : Deux routes qui proviennent d'arcs différents ne peuvent pas ressortir par le même arc, car les tables de routage sont des permutations.*

À partir de ces propriétés, il n'y a qu'une seule route de longueur donnée qui traverse un arc à son i-ème saut : une route est entièrement déterminée par cette information.

Chaque pair génère une route aléatoire pour chacun de ses arcs. Un vérifieur V accepte un suspect S comme honnête si la majorité des routes aléatoires issues de V croisent une route issue de S. L'intersection entre deux routes est illustrée figure 6.2.

6.1.2 Représentation du graphe

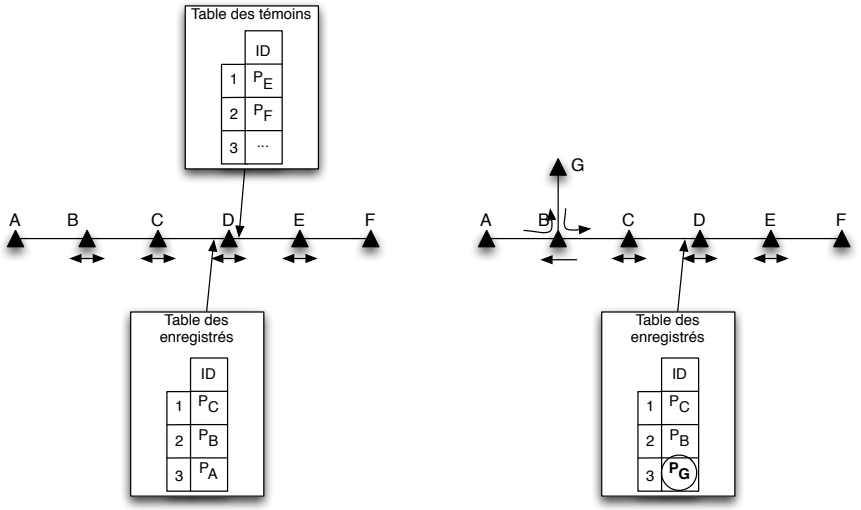
Le graphe SybilGuard est représenté par un ensemble de tables stockées par les pairs dans lesquelles chaque pair est identifié par sa clé publique. Pour chaque arc a, chaque pair P stocke une table des témoins et une table des enregistrés :

- la **table des témoins** contient tous les pairs sur la route aléatoire de P sortant par a ;
- la **table des enregistrés** contient tous les pairs dont la route aléatoire arrive à P par a.

Les deux tables d'un pair sont illustrées sur la figure 6.3(a).

Quand un pair ajoute un arc, il réinitialise sa table de routage aléatoirement, ce qui dévie les routes. Les tables sont ensuite mises à jour en conséquence, comme illustré figure 6.3(b).

Un vérifieur V accepte un suspect S comme un pair honnête s'il existe une majorité des routes aléatoires de V qui croisent une route de S. Le processus de test est le suivant :



(a) Tables des témoins et des enregistrés du pair D.

(b) B ajoute un voisin, réinitialise sa table de routage et la table des enregistrés de D évolue.

FIGURE 6.3 – Tables des témoins et des enregistrés.

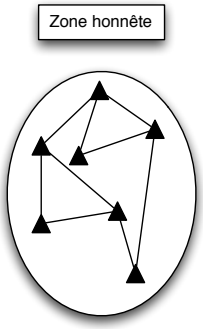
- S envoie ses tables des témoins à V ;
- V cherche un pair commun X entre ses témoins et ceux de V, un pair commun correspondant à une intersection de deux routes ;
- V contacte X pour vérifier que S a bien une route passant par X, ce qui est vrai si une des tables des enregistrés de X contient S.

6.1.3 Protection contre l'attaque sybile

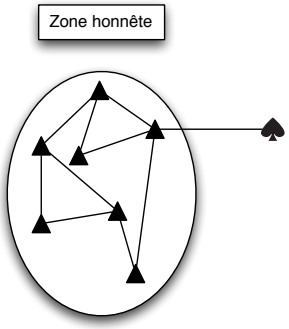
Une attaque sybile sur le réseau SybilGuard se déroule en deux étapes :

1. l'attaquant physique s'insère dans le réseau SybilGuard, en créant des arcs vers des amis. Ce nombre d'arcs est limité par le nombre d'amis de l'attaquant présents dans le réseau ;
2. l'attaquant crée un nombre illimité d'identités sybiles. Ces identités sybiles sont ajoutées en tant qu'amis du pair attaquant initial ou à sa place pour certains arcs vers la zone honnête.

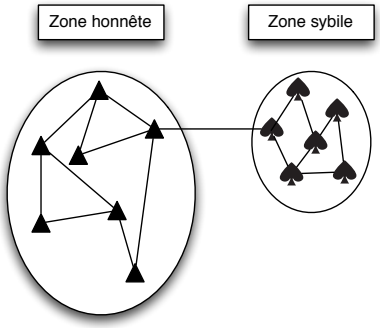
Le nombre de pairs sybils créés par un attaquant est illimité mais le nombre d'arcs vers le réseau honnête est limité : SybilGuard utilise cette limite. En effet, pour se faire accepter par un pair V, un pair S doit avoir une route qui croise une route du pair V. Grâce aux propriétés des routes aléatoires, toutes les routes des pairs sybils suivent les mêmes arcs d'entrée vers le réseau honnête et *convergent*. De plus, grâce à la *traçabilité arrière*, il n'existe qu'une seule route traversant un arc donné dans un sens à son i-ème saut, et donc un unique pair initiateur de cette route. Si la longueur des routes est limitée par un paramètre w, alors le nombre de routes de l'attaquant entrant dans la partie honnête du graphe social est également limité, ce qui limite le nombre de pairs sybils ayant une existence dans le graphe honnête. Cette protection est illustrée figure 6.4.



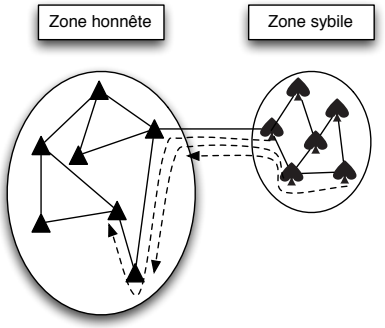
(a) Initialement, le réseau SybilGuard est un graphe social de pairs honnêtes ▲.



(b) Un attaquant ♠ crée un arc avec un pair honnête.



(c) L'attaquant crée autant de pairs sybils qu'il le souhaite, mais ne peut ajouter des arcs qu'entre eux.



(d) La longueur des routes étant limitée à 3, seuls trois pairs sybils peuvent créer des routes entrant dans la zone honnête (convergence et traçabilité arrière).

FIGURE 6.4 – Une attaque sybile dans le cas de SybilGuard. La longueur des routes aléatoires w vaut 3.

6.1.4 Longueur des routes aléatoires

La longueur des routes aléatoires w reflète un compromis entre la facilité d'insertion pour un pair honnête et la difficulté de créer des identités sybiles pour un attaquant. Le paramètre w doit être :

- suffisamment élevé pour garantir une forte probabilité d'intersection entre deux routes aléatoires honnêtes ;
- suffisamment faible pour empêcher un attaquant d'insérer un grand nombre de pairs sybils dans le graphe honnête.

Le calcul de la longueur optimale des routes w est effectué localement par chaque pair à partir d'un échantillonnage de tests selon une méthode proposée dans [YKGF06a]. Chaque pair situé sur une route utilise sa valeur de w pour propager la route et un attaquant ne peut donc pas créer des routes arbitrairement longues.

Un pair honnête peut éventuellement avoir une route aléatoire qui entre dans une partie sybile du graphe. Dans ce cas, cette route n'est plus protégée

par SybilGuard et croise un nombre illimité de pairs sybils : c'est pourquoi un pair n'est pas considéré comme honnête si une unique route croise une route du vérifieur mais plutôt si la majorité des routes du vérifieur croisent une route du suspect. Cette vérification permet de lisser le comportement de SybilGuard sur l'ensemble des routes de chaque pair.

La longueur des routes w garantissant une probabilité d'intersection élevée entre deux routes honnêtes est de l'ordre de $\Theta(\sqrt{n}\log n)$, où n est la taille du réseau [YKGF06a]. Chaque pair accepte ainsi de l'ordre de $\Theta(g \times w)$ pairs sybils par attaquant, où g est le nombre d'arcs entre cet attaquant et la partie honnête du graphe. La longueur w des routes joue donc un rôle important et nous montrons que la combinaison avec l'autorité de certification distribuée permet de réduire cette longueur.

SybilGuard seul ne propose aucun contrôle sur les identifiants des pairs. Chaque membre possède un couple de clés publique/secrète et chaque pair est identifié dans l'*overlay* pair-à-pair par le condensat de sa clé publique. Un attaquant peut ainsi générer un grand nombre de clés publiques avant de joindre SybilGuard, choisir celle dont le condensat est situé au bon endroit dans l'*overlay* [CDG+02] et enregistrer cette clé publique : SybilGuard limite le nombre de pairs sybils mais ne contraint pas leurs identifiants à être choisis aléatoirement. SybilGuard n'offre donc qu'une protection partielle contre l'attaque sybile et la combinaison avec l'autorité de certification distribuée permet de traiter le second aspect de la protection.

6.2 Contrôle d'admission résistant à l'attaque sybile

Dans cette section, nous présentons le couplage entre SybilGuard et l'autorité de certification distribuée. La protection contre l'attaque sybile est réalisée par un contrôle d'accès au réseau. Lors de sa première connexion, un pair doit obtenir un certificat délivré par l'autorité de certification distribuée.

L'insertion d'un nouveau pair S dans le réseau est réalisée en quatre étapes :

1. S génère un couple de clés publique/secrète noté (P_S, S_S) ;
2. S contacte un pair appartenant au réseau pour obtenir son certificat, qu'il n'obtient que s'il est considéré honnête par un membre de chaque groupe de fragmentation (chaque pair impliqué utilisant SybilGuard pour décider de coopérer ou non) ;
3. S rejoint l'*overlay* pair-à-pair en présentant son certificat, son identifiant étant le condensat de la signature de son certificat ;
4. S prend part au partage de la clé, obtient son arbre de fragmentation et peut à son tour participer au contrôle d'admission de nouveaux pairs.

Dans cette section, nous décrivons les points 2 (obtention du certificat) et 3 (identification du pair) qui sont spécifiques à cette application.

6.2.1 Obtention du certificat

Le contrôle du nombre de pairs sybils est réalisé lors de la deuxième étape du processus d'admission, à savoir durant l'obtention du certificat. La requête

de certification émise par le nouveau pair est $Req = (Content, Clues)$, qui contient :

- *Content*, qui représente les informations présentes dans le certificat final signé, contient P_S, la clé publique de S ;
- *Clues*, qui représente les informations fournies aux participants pour leur permettre de prendre leur décision, contient les tables des témoins de S représentées de manière compacte sous la forme d'un *filtre de Bloom* [Blo70]. Un filtre de Bloom est un résumé d'un ensemble dans lequel le test d'appartenance d'un élément peut renvoyer des faux positifs. Dans le cas de SybilGuard, les faux positifs correspondent à des intersections X et sont discriminés en contactant X.

Une signature distribuée est réalisée sur cette requête de certification. Un pair A, *bootstrap* de S, distribue la requête dans le réseau selon l'algorithme arborescent de signature distribuée.

Chaque pair participant à l'admission de S trouve dans la requête l'ensemble des tables de témoins de S : chaque participant peut donc évaluer l'honnêteté de S localement et décider de coopérer ou non à la signature. Les intersections trouvées sont ensuite vérifiées avec le pair X présent à l'intersection pour rejeter un attaquant fournissant de fausses tables et discriminer les faux positifs des filtres de Bloom. Les tables des enregistrés de X, dont la clé publique est P_X, sont stockées dans la DHT à l'emplacement $h(P_X)$ et sont signées par X, ce qui permet de réaliser cette vérification en l'absence de X.

S obtient son certificat si et seulement si un membre de chaque groupe de fragmentation, et donc un ratio t des pairs actuels, l'évalue honnête, en utilisant SybilGuard. Dans le cas où un pair refuse l'admission, la requête peut être retransmise à un autre pair du même groupe. L'obtention d'un certificat est illustrée figure 6.5.

Propriété 9 (*Limitation du nombre de pairs sybils*)

Un nouveau pair n'est accepté que si un ratio t des membres actuels le juge honnête, en utilisant SybilGuard. Un pair détecté comme sybil par tous les membres de l'un des groupes de fragmentation ne peut donc pas entrer dans le réseau.

6.2.2 Identification du pair

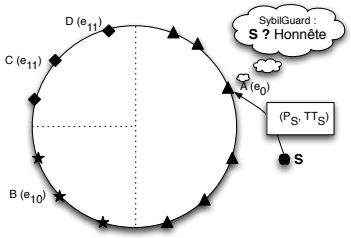
Une fois son certificat obtenu, S rejoint l'*overlay* en utilisant comme identifiant le condensat de la signature de son certificat. Si nous notons h la fonction de *padding* utilisée lors de la signature et h' la fonction de hachage utilisée pour obtenir l'identifiant de pair, alors l'identifiant du pair S vaut :

$$PeerId_S = h'(h(P_S)^e[m])$$

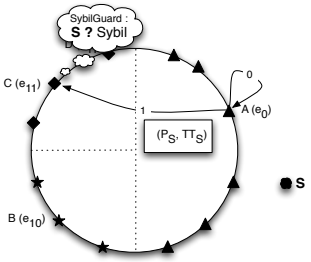
S peut initialement choisir sa clé publique P_S ; en revanche, S ne peut pas prévoir la signature de P_S (qui vaut $h(P_S)^e[m]$) et ne peut donc pas influencer le choix de son identifiant.

Propriété 10 (*Aléa des identifiants*)

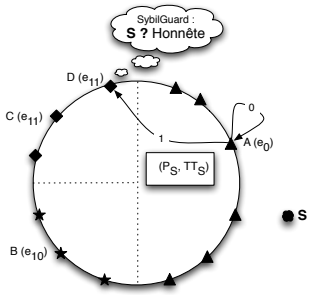
Un nouveau pair ne connaît son identifiant qu'une fois son certificat obtenu et donc, si l'utilisateur ne peut plus ensuite changer de clé publique, son identifiant est fixé et aléatoire.



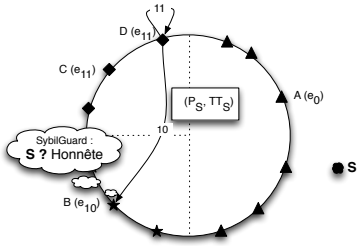
(a) Le suspect S transmet sa requête à son pair de *bootstrap* A. P_S est la clé publique de S et TT_S ses tables de témoins. A utilise SybilGuard et juge S honnête : A poursuit l'algorithme de signature.



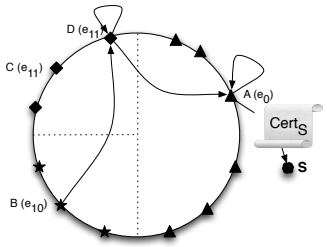
(b) A retransmet la requête à un pair C dont l'identifiant commence par 1. C utilise SybilGuard et juge S sybil : C refuse de poursuivre.



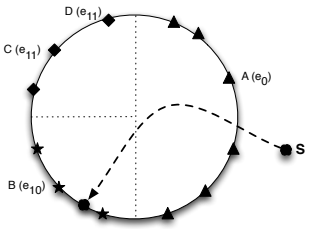
(c) A retransmet la requête à un autre pair D du même groupe que C. D accepte S et poursuit la signature.



(d) D transmet la requête à un pair B du groupe e°_{10} qui accepte S comme honnête.

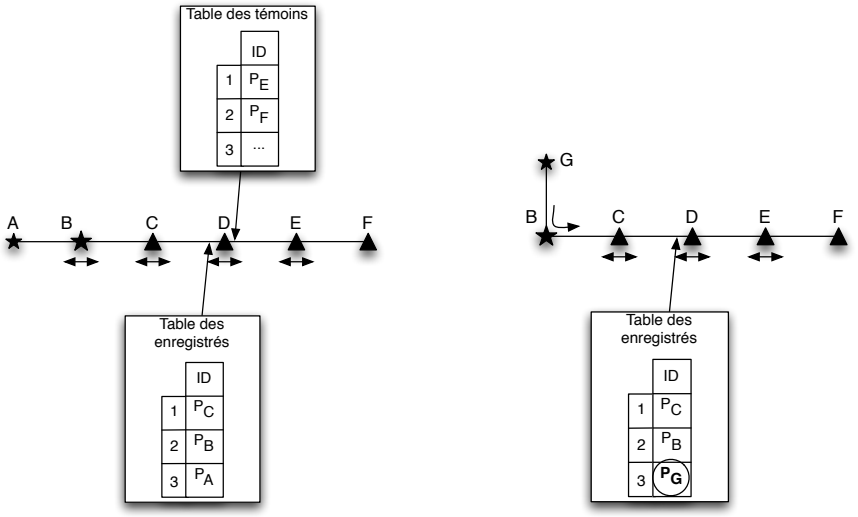


(e) Les trois pairs A, B et D réalisent conjointement le certificat de S.



(f) S rejoint le réseau à l'emplacement spécifié par le certificat.

FIGURE 6.5 – Contrôle d'admission d'un nouveau membre S. Le réseau est composé de trois groupes de fragmentation e°_0, e°_{10} et e°_{11}. S est accepté si un membre de chaque groupe le reconnaît honnête. À l'inverse, si tous les membres d'un groupe le jugent sybil, S n'obtient pas de certificat.



(a) L'attaquant B insère un premier pair sybil A, obtient son certificat et son identifiant aléatoire. B souhaite ensuite changer l'identifiant de A.

(b) B crée un autre pair G à la place de A, qui s'enregistre dans le réseau SybilGuard en remplaçant A. G peut ensuite obtenir un autre certificat avec un nouvel identifiant.

FIGURE 6.6 – Attaque sur la révocation des clés.

Le réseau SybilGuard ne doit pas permettre à un pair de changer sa clé publique. Dans la version originale de SybilGuard, la révocation de clé est autorisée et est utilisée pour maintenir les tables de routage SybilGuard. Quand deux pairs ajoutent un arc entre eux dans le graphe SybilGuard, ces deux pairs doivent mettre à jour leur table de routage SybilGuard pour diriger des routes vers et depuis ces nouveaux arcs : leur table de routage est réinitialisée aléatoirement. Les routes arrivant par ces nouveaux arcs ressortent par un ancien arc et écrasent l'ancienne route qui empruntait cet arc ; les routes déjà existantes sont déviées et les tables sont mises à jour, comme illustré figure 6.3. La possibilité de cet écrasement autorise un attaquant à changer sa clé publique, comme nous le montrons figure 6.6.

Avec la révocation des clés, un attaquant peut obtenir séquentiellement plusieurs certificats correspondant à des clés différentes, afin de finalement garder celui qui se trouve dans la partie de l'espace souhaitée. Nous envisageons deux contre-mesures :

- *limiter la fréquence des changements*, auquel cas chaque certificat a une durée de vie limitée et les tables de routage SybilGuard sont réinitialisées avec la même fréquence. Un pair ne peut donc mettre à jour sa clé publique qu'à la fin de la validité de son certificat. Cela permet de limiter le nombre d'identifiants qu'un attaquant peut obtenir mais induit une surcharge sur le réseau, puisque tous les pairs doivent renouveler leur certificat ;
- *interdire les changements*, auquel cas un certificat est obtenu à vie (ou à défaut pour une durée de plusieurs années) mais les tables de routage SybilGuard ne peuvent plus être modifiées. Au lieu de redistribuer toute la table à chaque ajout d'arc, deux nouveaux arcs successifs sont appai-

– $n = 1\ 000\ 000$ est le nombre de pairs du réseau
– $d = 24$ est le degré de chaque pair
– w est la longueur des routes aléatoires

FIGURE 6.7 – Résumé des notations et conditions

rés entre eux ; les anciens arcs conservent les mêmes routes et les tables d'enregistrement ne sont donc pas mises à jour. Les tables de routage SybilGuard ne sont, en revanche, plus des permutations aléatoires : les résultats expérimentaux présentés section 6.4 montrent que cela n'affecte que légèrement les performances de SybilGuard.

Nous choisissons la seconde contre-mesure, à savoir l'interdiction des changements de clé publique. Au lieu d'associer les arcs deux par deux, une stratégie plus évoluée et se rapprochant plus de la permutation aléatoire serait de permuter entre plus de deux nouveaux arcs. En contrepartie, tant que le pair n'a pas suffisamment d'arcs pour mettre à jour sa table de routage, les routes aléatoires passant par ces arcs sont interrompues.

6.3 Analyse théorique

Dans cette section, nous analysons le nombre de pairs sybils qui sont acceptés avec notre solution et nous comparons ces résultats avec SybilGuard seul. Nous montrons que la combinaison de SybilGuard avec l'autorité de certification distribuée permet d'obtenir de meilleurs résultats que SybilGuard seul, en plus de contraindre l'aléa des identifiants.

Nous considérons un graphe SybilGuard composé de $n = 1\ 000\ 000$ de personnes physiques et chaque personne a le même degré $d = 24$ (24 contacts), ce qui est résumé figure 6.7. Les simulations proposées dans la section 6.4 montrent qu'attribuer le même degré à chaque personne fournit une bonne approximation du résultat. Dans notre analyse, nous réutilisons certains résultats de [YKGF06a].

Nous évaluons d'abord SybilGuard seul puis SybilGuard avec l'autorité de certification distribuée. Dans chaque cas, nous calibrons la longueur minimale des routes permettant l'insertion de tous les pairs honnêtes et nous étudions les attaques sybiles dans ce cas. Plus précisément, nous calculons pour chacun des cas :

1. la longueur minimale w des routes aléatoires pour que chaque pair honnête soit accepté avec une probabilité de $0,999$;
2. le nombre de pairs sybils qu'un attaquant peut créer si les routes sont de cette longueur w ;
3. le nombre d'attaquants physiques nécessaire pour obtenir 250 000 pairs du réseau (25% de la taille initiale), auquel cas nous considérons le réseau inutilisable ;
4. le nombre d'attaquants physiques nécessaire pour que ces attaquants aient la possibilité d'insérer autant de pairs sybils que souhaité avec une probabilité de $0,001$.

Nous notons P_c la probabilité de collision entre deux routes aléatoires. La réciproque du paradoxe de l'anniversaire, détaillée dans [YKGF06b], fournit un

rapport des probabilités de collision pour deux longueurs de route données et les auteurs fournissent par ailleurs les valeurs de référence pour une probabilité de collision de $0,5$. Dans notre cas, la longueur $w(P_c)$ des routes aléatoires en fonction de la probabilité de collision P_c souhaitée entre deux routes aléatoires peut donc être calculée par rapport à la longueur nécessaire pour une probabilité de collision de $0,5$ selon le calcul suivant :

$$\frac{w(p)}{w(q)} = \frac{\sqrt{\ln\left(\frac{1}{1-p}\right)}}{\sqrt{\ln\left(\frac{1}{1-q}\right)}}$$

$$\Rightarrow \frac{w(P_c)}{w(0,5)} = \frac{\sqrt{\ln\left(\frac{1}{1-P_c}\right)}}{\sqrt{\ln\left(\frac{1}{0,5}\right)}}$$

$$\Leftrightarrow w(P_c) = w(0,5) \times \frac{\sqrt{\ln\left(\frac{1}{1-P_c}\right)}}{\sqrt{\ln\left(\frac{1}{0,5}\right)}}$$

Dans un réseau composé de 1 000 000 de pairs, $w(0,5)$ vaut 916,822 et peut être estimé localement par chaque pair utilisant SybilGuard.

6.3.1 SybilGuard seul, sans l'autorité de certification distribuée

Nous considérons qu'un pair s'est inséré avec succès quand il est accepté par chaque autre pair. Chaque pair a d connexions vers d'autres pairs et a donc d routes aléatoires de longueur w dans le réseau. Un pair V accepte un autre pair S si et seulement si plus de la moitié des routes de V croisent des routes de S.

6.3.1.1 Longueur des routes aléatoires

Nous calculons ici la longueur minimale des routes aléatoires pour que chaque pair honnête soit accepté. La probabilité qu'une route de V croise au moins une route de S parmi les d vaut $1-(1-P_c)^d$. La probabilité que i routes de V croisent au moins une route de S vaut $C_d^i(1-(1-P_c)^d)^i((1-P_c)^d)^{d-i}$. La probabilité que plus de la moitié des routes de V croisent des routes de S vaut donc :

$$\boxed{P_{SGaccept} = \sum_{i=\frac{d}{2}}^{d} C_d^i(1-(1-P_c)^d)^i((1-P_c)^d)^{d-i}}$$

La probabilité que les n pairs acceptent S vaut $P_{SGaccept}^n$. Pour obtenir une probabilité d'acceptation générale de $0,999$ pour un pair honnête, une résolution numérique fournit $P_c = 0,1053$. Cette probabilité de collision requiert des routes de longueur $w = 368$.

6.3.1.2 Nombre de pairs sybils par attaquant physique

Nous calculons ici le nombre de pairs sybils qu'un attaquant physique peut insérer dans le réseau. Nous considérons un attaquant physique ayant d contacts dans le réseau honnête et des routes de longueur w. En raison de la convergence et de la traçabilité arrière, il existe en tout $d \times w$ routes sortant de la composante sybile, quel que soit le nombre de pairs sybils générés. Les $d \times w$ routes sont toutes initiées par des pairs sybils, chaque pair sybil étant à l'origine d'une ou plusieurs routes. La stratégie de l'attaquant pour insérer le plus grand nombre de pairs sybils reflète un compromis entre deux possibilités :

- chaque pair sybil est à l'origine d'un faible nombre de routes, auquel cas beaucoup de pairs sybils génèrent des routes ;
- chaque pair sybil est à l'origine de beaucoup de routes, auquel cas chaque pair sybil est mieux inséré dans le réseau et a plus de chances d'être accepté.

L'attaquant dispose de d routes distinctes, selon les d arcs de sortie de la zone sybile, les autres routes étant des préfixes de ces d routes. L'attaquant doit donc attribuer entre 1 et d routes à chaque pair sybil qu'il souhaite insérer et nous calculons ici la valeur optimale. Bien que l'attaquant possède d routes de longueur w dans la zone honnête, d routes de longueur $w-1$, ..., d routes de longueur 1, nous calculons ici une majoration du nombre de pairs sybils en considérant chaque route de longueur w.

Un pair honnête V juge un autre pair S honnête si la moitié des routes de V croisent celles de S, éventuellement plusieurs fois la même route de S. Si S a j routes dans le réseau, alors la probabilité qu'une route de V croise l'une des j de S vaut $1-(1-P_c)^j$. La probabilité que i routes de V croisent une route de S parmi les j vaut $C_d^i \times (1-(1-P_c)^j)^i \times ((1-P_c)^j)^{d-i}$. La probabilité que la moitié des routes de V croise l'une des j routes de S vaut donc :

$$\boxed{P_{SGacceptSybil}(j) = \sum_{i=\frac{d}{2}}^{d} C_d^i \times (1-(1-P_c)^j)^i \times ((1-P_c)^j)^{d-i}}$$

Si l'attaquant attribue j routes à chaque pair sybil, alors l'attaquant crée $\frac{d \times w}{j}$ pairs sybils qui ont chacun une probabilité $P_{SGacceptSybil}(j)$ de s'insérer dans le réseau. En moyenne, l'attaquant peut insérer le nombre de pairs sybils :

$$\boxed{Nb_{SGsybils} = \frac{d \times w}{j} \times P_{SGacceptSybil}(j)}$$

Dans les conditions proposées ($d = 24$ et $w = 368$), cette courbe est tracée figure 6.8 et l'optimum est atteint pour $j = 8$, avec 950 pairs sybils par attaquant physique.

6.3.1.3 Nombre d'attaquants nécessaire pour contrôler 250 000 pairs du réseau

Nous calculons maintenant le nombre d'attaquants physiques nécessaire pour contrôler 250 000 pairs du réseau, auquel cas nous considérons le réseau corrompu. Si chaque attaquant peut créer 950 pairs sybils, alors 263 attaquants en collusion permettent d'obtenir 250 000 pairs sybils (25% de la taille initiale du réseau).

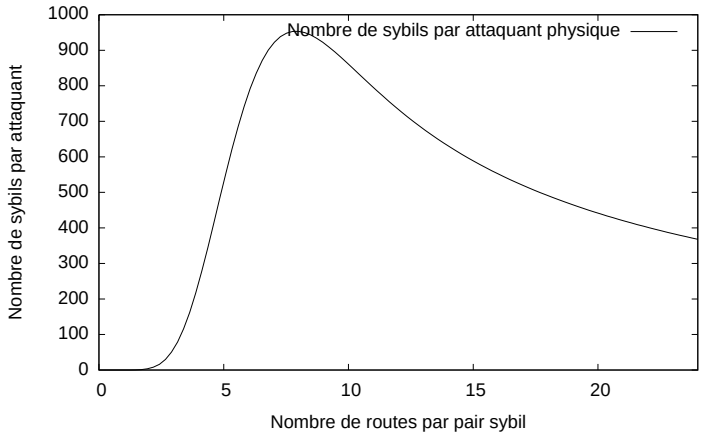


FIGURE 6.8 – Nombre de pairs sybils par attaquant physique, pour SybilGuard seul.

6.3.1.4 Nombre d'attaquants pour corrompre un pair honnête

Nous calculons enfin le nombre d'attaquants minimal pour que ces attaquants puissent créer un nombre illimité de pairs sybils vis à vis d'un pair honnête, ce qui arrive si au moins la moitié des routes du pair honnête entrent dans la région sybile. Nous considérons une répartition uniforme des arcs sybils. La probabilité qu'une route n'entre pas dans la région sybile vaut $(1-k)^w$ et la probabilité que i routes entrent dans la région sybile vaut $C_d^i(1-(1-k)^w)^i((1-k)^w)^{d-i}$. La probabilité qu'au moins la moitié des routes entrent dans la région sybile vaut donc :

$$\boxed{P_{SGsybil} = \sum_{i=\frac{d}{2}}^{d} C_d^i(1-(1-k)^w)^i((1-k)^w)^{d-i}}$$

Une résolution numérique avec $w = 368$ fournit $k = 0.00061$, ce qui correspond à 610 attaquants physiques, pour introduire un nombre illimité de pairs sybils dans la vue d'un pair honnête avec une probabilité $0,001$.

6.3.2 SybilGuard avec l'autorité de certification distribuée

Nous analysons maintenant SybilGuard couplé à l'autorité de certification distribuée. Dans ce cas, un nouveau pair doit être accepté par le SybilGuard d'un pair de chaque groupe de fragmentation. La taille du groupe $\mathring{e}_i$ est g_i et le nombre de groupes de fragmentation vaut s ; il y a donc $n = \sum_{i=1}^{s} g_i$ pairs dans le réseau.

6.3.2.1 Longueur des routes aléatoires

La probabilité qu'un pair accepte un autre pair vaut P_{accept}, calculé précédemment. La probabilité qu'il existe un pair dans un groupe donné de taille g_i qui accepte S vaut $1-(1-P_{SGaccept})^{g_i}$. La probabilité qu'un pair de chaque

groupe accepte S vaut donc :

$$\boxed{P_{DSGaccept} = \prod_{i=1}^{s} \left(1 - (1 - P_{SGaccept})^{g_i}\right)}$$

Si nous considérons des groupes composés de $g_{min} = 20$ à $g_{max} = 40$ membres, la résolution numérique fournit, pour que chaque pair honnête soit accepté avec une probabilité de $0,999$:

- $P_c = 0,0286$ si tous les groupes sont composés de $g_{min} = 20$ membres (il y a $s = 50\ 000$ groupes de fragmentation). La longueur des routes w doit valoir 188.
- $P_c = 0,0239$ si tous les groupes sont composés de $g_{max} = 40$ membres (il y a $s = 25\ 000$ groupes de fragmentation). La longueur des routes w doit valoir 172.

Les valeurs réelles obtenues avec des groupes composés de 20 à 40 membres se situent donc entre ces deux bornes.

6.3.2.2 Nombre de pairs sybils par attaquant physique

Tout comme dans la sous-section 6.3.1, l'attaquant attribue un nombre de routes j à chacun de ses pairs sybils. La probabilité qu'un pair accepte un pair sybil ayant j routes vaut $P_{SGacceptSybil}(j)$. La probabilité qu'un pair de chaque groupe accepte un pair sybil vaut donc, de la même manière que précédemment :

$$\boxed{P_{DSGacceptSybil}(j) = \prod_{i=1}^{s} \left(1 - (1 - P_{SGacceptSybil}(j))^{g_i}\right)}$$

Si l'attaquant attribue j routes à chaque pair sybil, alors l'attaquant crée $\frac{d \times w}{j}$ pairs sybils qui ont chacun une probabilité $P_{DSGacceptSybil}(j)$ de s'insérer dans le réseau. En moyenne, l'attaquant peut insérer le nombre de pairs sybils :

$$\boxed{Nb_{DSGsybils} = \frac{d \times w}{j} \times P_{DSGacceptSybil}(j)}$$

Dans les conditions proposées, la figure 6.9 illustre cette probabilité pour les deux tailles extrêmes de groupes g_{min} et g_{max}. L'optimum est atteint pour $g = 20$ et $j = 23$, avec 193 pairs sybils par attaquant physique.

6.3.2.3 Nombre d'attaquants nécessaire pour contrôler 250 000 pairs du réseau

Nous calculons maintenant le nombre d'attaquants physiques nécessaire pour contrôler 250 000 pairs du réseau, auquel cas nous considérons le réseau corrompu. Si chaque attaquant peut créer 193 pairs sybils, alors 1295 attaquants en collusion permettent d'obtenir 250 000 pairs sybils (25% de la taille initiale du réseau).

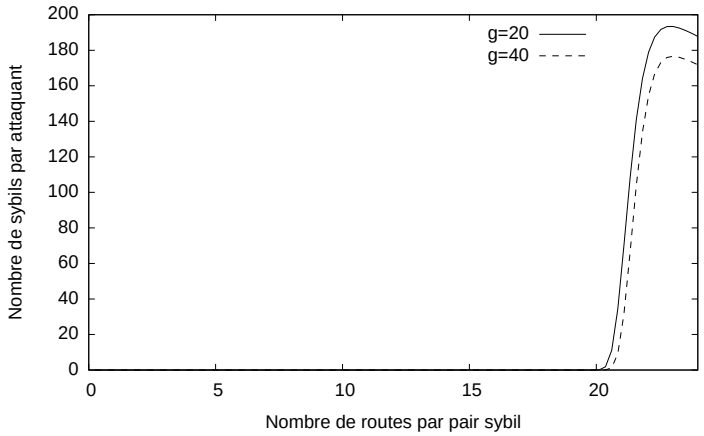


FIGURE 6.9 – Nombre de pairs sybils par attaquant physique, pour SybilGuard avec autorité de certification distribuée.

6.3.2.4 Nombre d'attaquants pour corrompre un pair honnête

Nous calculons enfin le nombre d'attaquants minimal pour que ces attaquants puissent créer un nombre illimité de pairs sybils. Les attaquants peuvent :

- posséder un pair dans chaque groupe de fragmentation, auquel cas ils peuvent générer de faux certificats. À partir de l'étude proposée dans le chapitre 2, 36% d'attaquants peuvent générer de faux certificats dans un réseau de 1 000 000 de pairs composé de groupes de $g_{min} = 20$ à $g_{max} = 40$ membres, avec une probabilité de $0,001$. Si chaque attaquant physique peut créer 193 pairs sybils, alors 1865 attaquants agissant en collusion peuvent générer de faux certificats ;
- corrompre le SybilGuard d'un pair de chaque groupe de fragmentation, auquel cas un pair de chaque groupe accepte un nombre illimité de pairs sybils. Cette situation arrive pour chaque pair avec une probabilité $P_{SGsybil}$. La probabilité qu'il y ait au moins un pair dans chaque groupe qui ne soit plus protégé par SybilGuard vaut :

$$\boxed{P_{DSGsybil} = \prod_{i=1}^{s} \left(1 - (1 - P_{SGsybil})^{g_i}\right)}$$

Par résolution numérique, nous obtenons que 3110 attaquants sont nécessaires pour réussir cette attaque avec une probabilité de $0,001$.

6.3.3 Comparaison

L'analyse théorique montre que l'association de SybilGuard avec l'autorité de certification distribuée permet d'utiliser des routes aléatoires plus courtes que ne le permet SybilGuard seul, pour une même probabilité d'insertion. Le nombre de pairs sybils étant directement dépendant de la longueur des routes utilisée, l'approche avec autorité de certification distribuée accepte moins de pairs sybils par attaquant physique.

	SybilGuard seul	SybilGuard distribué
w	368	188
Nb sybils par attaquant	< 950	< 193
Nb attaquants pour 25% de sybils	> 263	> 1295
Nb attaquants pour corrompre	> 610	> 1865

TABLE 6.1 – Comparaison de SybilGuard seul avec SybilGuard couplé à l'autorité de certification distribuée, dans un réseau composé de 1 000 000 pairs.

Les résultats obtenus, résumés tableau 6.1, posent une borne supérieure aux attaques sybiles dans le cas où chaque pair sybil a des routes de longueur w dans la zone honnête. Cependant, chaque pair sybil ajouté a, en fait, des routes moins longues dans la partie honnête du graphe social et les résultats présentés sont donc une majoration.

6.4 Étude expérimentale

Dans cette section, nous présentons des résultats expérimentaux obtenus dans un réseau de 10 000 pairs et nous les comparons aux résultats théoriques attendus. Les simulations sont réalisées en utilisant PeerSim [JMJV] et les attaquants physiques sont choisis aléatoirement dans le graphe social.

6.4.1 Cadre expérimental

Nous étudions la longueur des routes aléatoires et le nombre de pairs sybils par attaquant physique dans les trois cas suivants :

- SybilGuard seul, dans sa configuration originale permettant de révoquer les clés publiques des pairs ;
- SybilGuard avec l'autorité de certification distribuée et en autorisant la révocation des clés (*routes aléatoires*). Même si la révocation n'est pas compatible avec notre système, cette étude permet de borner les possibilités de notre système ;
- SybilGuard avec l'autorité de certification distribuée et en interdisant la révocation des clés (*routes biaisées*). Dans ce cas, chaque pair réalise ses tables de routage en appairant les connexions successives. Une stratégie plus évoluée, en attendant un troisième pair à chaque fois par exemple, permettrait de se rapprocher du cas des routes aléatoires.

Dans le cas de l'autorité de certification, les groupes sont composés de $g_{min} = 20$ à $g_{max} = 40$ membres et les tables de routage sans révocation, qui ne sont pas des permutations aléatoires, sont réalisées de la manière suivante :

- un premier pair est choisi aléatoirement pour initier le réseau ;
- un arc de ce pair est ajouté ainsi que le pair de l'autre côté de cet arc ;
- à chaque tour, un nouvel arc pointant vers un pair déjà ajouté est choisi et le pair à l'autre extrémité est ajouté, s'il ne l'était pas déjà ;
- les arcs sont associés deux par deux, dès que possible.

Comme dans [YKGF06a], nous modélisons le réseau social par un graphe de Kleinberg [Kle00], permettant la génération aléatoire de graphes petit monde. Dans [NNK+05], Liben-Nowell *et al.* ont montré l'adéquation entre ce modèle algorithmique et les réseaux sociaux réels.

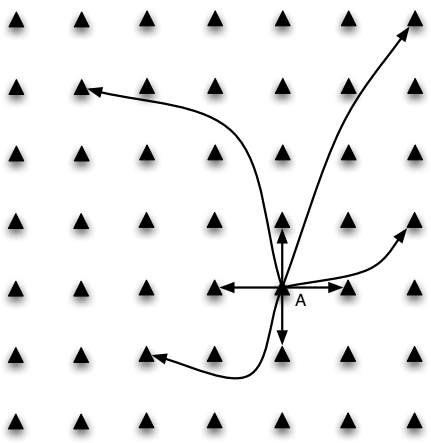


FIGURE 6.10 – Les voisins de A dans un graphe de Kleinberg, avec $p = q = 4$. A est connecté à ses 4 pairs les plus proches et à 4 pairs éloignés.

Dans un graphe de Kleinberg, les pairs sont placés dans une matrice et chaque pair est connecté à ses p voisins les plus proches et à q voisins éloignés choisis aléatoirement, la probabilité d'avoir chaque pair comme voisin éloigné étant inversement proportionnelle à la distance avec ce pair, selon un paramètre r. Cet algorithme est illustré figure 6.10.

Dans cette section, nous utilisons $p = q = 8$ et $r = 1.9$. Chaque pair a 8 voisins proches, 8 voisins éloignés et est en moyenne le voisin éloigné de 8 pairs : le degré moyen des pairs vaut 24.

6.4.2 Longueur des routes aléatoires

Nous étudions d'abord la longueur des routes aléatoires. La figure 6.11 illustre la probabilité d'insertion pour un pair honnête en fonction de la longueur des routes nécessaires selon les trois cas (SybilGuard seul, SybilGuard avec autorité et routes aléatoires, SybilGuard avec autorité et routes biaisées). En cohérence avec l'analyse, il apparaît bien que la combinaison de SybilGuard avec l'autorité de certification distribuée permet d'utiliser des routes plus courtes, ce qui permet d'accepter moins de pairs sybils.

Le tableau 6.2 compare la longueur des routes aléatoires nécessaire pour obtenir une probabilité d'insertion de $0,999$ pour chaque pair honnête, selon l'analyse théorique présentée et selon les expérimentations. Même si l'analyse théorique fournit une bonne approximation, un décalage existe clairement entre la théorie et l'expérience. Ce décalage a deux causes :

- dans l'analyse théorique, nous avons considéré que tous les pairs avaient un degré identique égal à 24 ; dans les simulations, les degrés sont en fait distribués entre 16 et 39. En répétant l'analyse théorique avec les degrés obtenus expérimentalement, nous obtenons le résultat $w = 47$ dans le cas de SybilGuard seul, très proche du $w = 45$ obtenu expérimentalement. Il serait intéressant de raffiner l'analyse avec une distribution de degrés ;
- la formule utilisée dans l'analyse pour calculer w en fonction de la probabilité de collision P_c repose sur des tables de routage aléatoires. Dans le

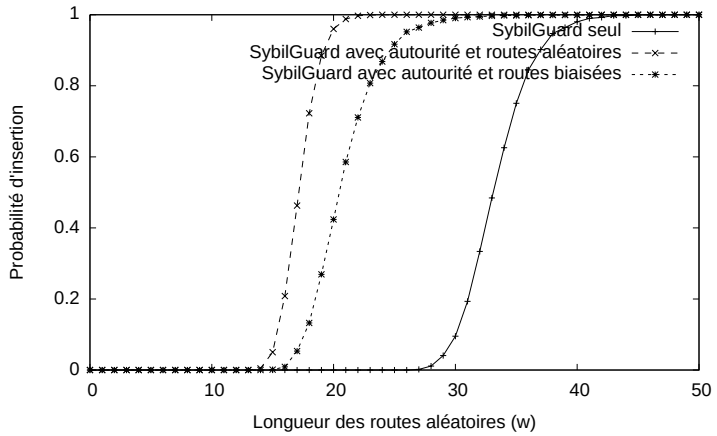


FIGURE 6.11 – Probabilité d'insertion en fonction de la longueur des routes aléatoires, pour les trois cas étudiés (SybilGuard seul, SybilGuard avec autorité et routes aléatoires, SybilGuard avec autorité et routes biaisées).

	Théorique	Expérimental
SybilGuard seul, routes aléatoires	38	45
SybilGuard + autorité, routes aléatoires	19	23
SybilGuard + autorité, routes biaisées	19	37

TABLE 6.2 – Comparaison des longueurs de routes aléatoires nécessaires.

cas des routes biaisées, les tables de routage ne sont plus totalement aléatoires et les performances sont dégradées. Nous constatons tout de même que les performances restent meilleures que celles de SybilGuard seul. Il serait intéressant d'étudier une stratégie de création des tables de routage plus évoluée.

6.4.3 Nombre de pairs sybils par attaquant

Nous étudions maintenant le nombre de pairs sybils qu'un attaquant peut créer. Pour créer un maximum de pairs sybils, l'attaquant doit

- maximiser le nombre de pairs sybils annoncés dans la zone honnête du réseau social ;
- maximiser le nombre de routes de chaque pair sybil, afin d'augmenter la probabilité que chacun d'eux soit reconnu comme honnête.

Ces deux conditions sont contradictoires et l'attaquant doit trouver un équilibre entre le nombre de pairs sybils annoncés et la qualité de leur insertion dans la zone honnête. Nous considérons ici des attaquants utilisant leur stratégie optimale que nous définissons de la manière suivante :

- soit un pair honnête V de degré d. Ce pair accepte un suspect S si les routes aléatoires du suspect croisent $\frac{d}{2}$ routes parmi les d routes de V. Si les routes sont de longueur w et si une route de V intersecte une route de S à son i-ème saut, S peut insérer $w - i$ pairs sybils derrière lui, sur cette même route, ce qui crée $w - i$ intersections entre V et l'attaquant. Cette

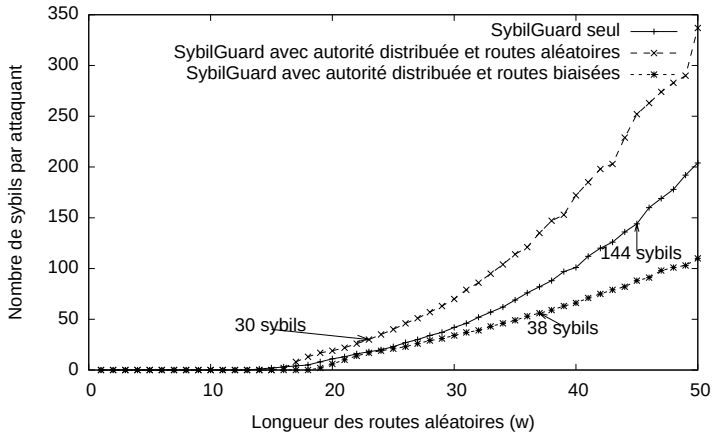


FIGURE 6.12 – Nombre de pairs sybils par attaquant physique.

	w	Nb sybils par attaquant	
		Théorique	Expérimental
SybilGuard seul, routes aléatoires	45	< 116	< 144
SybilGuard + autorité, routes aléatoires	23	< 26	< 30
SybilGuard + autorité, routes biaisées	37	< 42	< 38

TABLE 6.3 – Nombre de pairs sybils par attaquant physique.

opération est répétée sur toutes les routes ;

– de la même manière que dans l'analyse théorique, nous considérons que chaque intersection permet l'insertion de w pairs sybils sur la route au lieu de $w - i$, ce qui majore le nombre de pairs sybils acceptés ; sans cette majoration, le calcul de la stratégie optimale devient un problème combinatoire trop complexe à simuler (l'attaquant possède entre 1 et $d \times w$ pairs à attribuer entre 1 et $d \times w$ routes, il peut favoriser plusieurs routes longues pour le même pair ou au contraire attribuer les routes de manière équitable).

Pour chaque longueur de route w, nous calculons expérimentalement le meilleur compromis entre le nombre de pairs sybils annoncés et le nombre de routes de chaque pair sybil. Les résultats présentés sont toujours obtenus avec le compromis optimal.

La figure 6.12 montre le nombre de pairs sybils qu'un attaquant peut créer en fonction de la longueur des routes selon les trois cas étudiés (SybilGuard seul, SybilGuard avec autorité et routes aléatoires, SybilGuard avec autorité et routes biaisées). En cohérence avec l'analyse, il apparaît bien que la combinaison de SybilGuard avec l'autorité de certification distribuée, qui conduit à d'utiliser des routes plus courtes, permet d'accepter moins de pairs sybils. Avec SybilGuard seul, w vaut 45 et chaque attaquant physique peut insérer 144 pairs sybils ; avec SybilGuard et l'autorité de certification distribuée, avec des routes biaisées, w vaut 37 et le nombre de pairs sybils par attaquant chute à 38. L'utilisation d'une stratégie de routage plus évoluée pourrait permettre d'atteindre une limite à 30 pairs sybils par attaquant.

	Nb limité	Id aléatoire	Absence de centre
Approches centralisées	√	√	X
Tests calculatoires	X	√	√
Tests sociaux	√	X	√
Notre proposition	√	√	√

TABLE 6.4 – Comparaison de notre proposition contre l'attaque sybile avec les trois approches présentées dans l'état de l'art.

Le tableau 6.3 compare les résultats expérimentaux et théoriques. Dans chaque cas, nous choisissons la valeur w obtenue expérimentalement et nous comparons le nombre de pairs sybils par attaquant selon le calcul théorique et l'étude expérimentale avec cette longueur de route. Le couplage avec l'autorité de certification permet une meilleure résistance à l'attaque sybile.

Résumé

Dans ce chapitre, nous avons proposé un contrôle d'admission pour les réseaux pair-à-pair structurés résistant à l'attaque sybile. L'appartenance au réseau est matérialisée par un certificat contenant la clé publique du pair. La signature du certificat est distribuée dans le réseau et le nouveau pair n'obtient une signature valide que si un pair de chaque groupe de fragmentation, soit un ratio t des membres, le juge honnête, en utilisant SybilGuard. La combinaison de SybilGuard avec l'autorité de certification distribuée permet de traiter les deux aspects de l'attaque sybile, à savoir la limitation du nombre d'identités et l'aléa des identifiants obtenus.

Nous avons d'abord présenté SybilGuard et son fonctionnement. Chaque pair enregistre des routes aléatoires dans le graphe social. Dans des graphes sociaux, deux routes aléatoires ont une forte probabilité de se rencontrer rapidement. De plus, les routes aléatoires des pairs sybils d'un même attaquant convergent et restent ensuite confondues. Nous avons présenté l'implémentation des routes sous la forme de tables des témoins et des enregistrés stockées par chaque pair. Nous avons également décrit la vérification des pairs : un pair considère un autre pair comme sybil si leurs routes ne se croisent pas. SybilGuard permet de limiter le nombre de pairs sybils mais ne contraint pas l'aléa de leurs identifiants.

Nous avons ensuite proposé le couplage de SybilGuard avec l'autorité de certification distribuée. Un nouveau pair doit d'abord s'insérer dans le réseau SybilGuard pour ensuite demander à accéder au réseau pair-à-pair. Le droit d'accès est matérialisé par un certificat, signé avec l'algorithme distribué, qui n'est obtenu que si un pair de chaque groupe de fragmentation accepte de coopérer à la signature distribuée : un pair n'obtient un certificat que si un membre de chaque groupe de fragmentation le considère honnête, en utilisant SybilGuard. L'identifiant du pair est ensuite dérivé du certificat obtenu, que l'attaquant ne peut pas prévoir. L'autorité de certification distribuée garantit ainsi l'aléa des identifiants de pairs.

Enfin, nous avons analysé et évalué expérimentalement SybilGuard seul et le couplage proposé. Nous avons obtenu des résultats satisfaisants et nous avons montré que le couplage permettait de mieux limiter le nombre de pairs sybils.

Dans le tableau 6.4, nous comparons notre proposition aux trois approches identifiées dans l'état de l'art. Ce tableau prend en compte la limitation du nombre de pairs par personne physique, la contrainte sur l'aléa des identifiants et l'absence de centre.

Ce travail a été publié à COPS 2008 [LMV08c]. Par ailleurs, ce contrôle d'accès au réseau a été implémenté dans notre prototype, dans le cadre du stage de Laurent Bonnet, en quatrième année d'INSA, encadré par l'auteur.

Chapitre 7

Service de nommage sécurisé

« *I always thought it would be better to be a fake somebody than a real nobody.* »
Tom Ripley, *in* The Talented Mr. Ripley

Nous proposons dans ce chapitre un service de nommage sécurisé permettant la distribution de clés cryptographiques. Chaque pair peut enregistrer une identité intelligible en utilisant l'autorité de certification distribuée présentée dans la première partie de ce mémoire ; la possession d'une identité est ensuite matérialisée par un certificat contenant la clé publique de son propriétaire et signé avec la clé secrète de l'autorité de certification distribuée. L'obtention de la clé publique d'un pair tiers est réalisée par une requête contenant l'identité enregistrée de ce pair.

Comme nous l'avons présenté dans l'état de l'art, la cryptographie permet d'assurer la confidentialité et l'intégrité des données stockées dans la DHT. Nous utilisons ici la cryptographie asymétrique (par exemple RSA) dans laquelle chaque pair est muni d'une paire de clés publique/secrète :

- la *signature* cryptographique assure l'intégrité des données. Pour assurer l'intégrité d'une donnée, un pair A signe cette donnée avec sa clé secrète ; le pair B accédant ensuite à cette donnée peut tester l'intégrité de la donnée en vérifiant la signature grâce à la clé publique de A ;
- le *chiffrement* cryptographique assure la confidentialité des données. Pour assurer la confidentialité d'une donnée, un pair A chiffre cette donnée avec la clé publique de B ; le pair B autorisé à lire la donnée peut ensuite la déchiffrer en utilisant sa clé secrète.

La difficulté dans ce cas réside dans la distribution des clés : chaque pair doit en effet pouvoir obtenir de manière sûre la clé publique des autres pairs avec qui il souhaite communiquer. Dans un réseau pair-à-pair, la distribution de ces clés peut être réalisée suivant les procédures suivantes :

- la *distribution explicite* permet aux utilisateurs d'échanger leurs clés publiques par courrier électronique ou lors d'une rencontre, par exemple. L'utilisation d'un moyen tiers pour distribuer les clés est peu pratique,

puisque non intégré à l'application pair-à-pair. De plus, l'obtention de la clé ne peut être effectuée en temps réél, à la demande. Enfin, la sécurité de la distribution dépend de la sécurité du moyen tiers utilisé : le courrier électronique n'est, par exemple, généralement pas sécurisé (peu de gens utilisent PGP) ;

- la *connaissance du condensat de la clé recherchée*, utilisée par exemple dans les chemins auto-signés de CFS [DKK+01] où le chemin vers la ressource recherchée contient le condensat de la clé publique du propriétaire du fichier, permet de vérifier une clé publique obtenue de manière non sûre. L'obtention de ce condensat se heurte cependant aux mêmes problèmes que l'échange de clés par un moyen tiers : un condensat SHA-1 est composé de 40 chiffres hexadécimaux et n'est donc pas intelligible pour une personne : ce condensat doit être préalablement obtenu électroniquement par un moyen tiers ;
- une *requête dans la DHT d'un nom intelligible*, auquel cas chaque pair enregistre sa clé publique sous la clé $h(name)$ dans la DHT, *name* étant une chaîne de caractères intelligible telle qu'un pseudonyme. Bien que ce système soit facilement utilisable, l'obtention de la clé n'est pas sécurisée. La requête d'un pair vers $h(name)$ peut être interceptée par un attaquant qui peut renvoyer une mauvaise valeur ; les réplicats eux-mêmes peuvent contenir des valeurs conflictuelles. L'utilisation de routage redondant dans ce cas améliore la sécurité, mais augmente également la latence et la quantité de données à échanger, ce qui pose problème dans les systèmes nécessitant une forte réactivité (DNS, messagerie instantanée, système de fichiers distribué). Le routage redondant est, de plus, vulnérable à une attaque éclipse sur un pair, qui consiste à remplir ses tables de routage d'attaquants ;
- une *infrastructure à clé publique* est généralement utilisée pour distribuer les clés de manière sécurisée et automatique. Comme nous l'avons vu dans l'état de l'art, des deux types d'infrastructures à clé publique – réseaux de confiance et autorités de certification –, seules les autorités de certification permettent de nommer les entités de manière unique. Dans les réseaux de confiance, chaque utilisateur a une vue locale différente et plusieurs utilisateurs peuvent avoir le même nom dans des parties distinctes du réseau (les réseaux de confiance n'imposent pas la cohérence des vues locales) ; avec une autorité de certification, chaque utilisateur possède un certificat liant sa clé publique à un nom intelligible, signé par l'autorité de certification, l'autorité de certification se chargeant de vérifier l'unicité des noms demandés et de distribuer les certificats. La sécurité repose dans ce cas sur cette autorité ponctuelle, ce qui va à l'encontre du paradigme pair-à-pair. De plus, l'opération de cette autorité a un coût important.

Nous proposons dans ce chapitre un système de distribution de clés sécurisé utilisant l'autorité de certification distribuée proposée dans la première partie de ce mémoire. Chaque pair peut enregistrer une identité, sous la forme d'une chaîne de caractères arbitraire, qui est ensuite prouvable et unique (aucun autre pair ne peut enregistrer le même nom). Les applications directes peuvent être :

- un système DNS pair-à-pair où les utilisateurs enregistrent des domaines ;
- une messagerie instantanée pair-à-pair où les utilisateurs enregistrent un pseudonyme ;

– un système de fichiers distribué dans lequel chaque utilisateur possède un répertoire racine nommé et peut accorder des droits de lecture et/ou écriture à d'autres utilisateurs nommés ;
– l'enregistrement des entités SybilGuard.

La possession d'une identité est matérialisée par un certificat. Ce certificat contient le nom *name* enregistré et la clé publique de son possesseur ; ce certificat est signé par la clé de réseau. L'enregistrement d'une identité se déroule en deux étapes principales :

1. le pair obtient le certificat associé au nom *name* souhaité ;
2. le pair insère ce certificat dans la DHT de manière permanente, à la position $h(name)$, afin de pouvoir être contacté.

Une fois un certificat enregistré en $h(name)$, l'autorité de certification refuse de signer un certificat pour ce même nom et aucun autre pair ne peut donc obtenir de certificat pour ce nom. Ce certificat est ensuite trouvé par une requête simple, sans redondance, vers $h(name)$: si une réponse est obtenue, alors le demandeur sait que ce certificat est le bon.

Dans le système proposé, le coût associé aux vérifications du nom demandé est réalisé lors de l'enregistrement de ce nom. À ce moment-là, un pair de chaque groupe de fragmentation vérifie le nom demandé, ce qui correspond à un routage redondant provenant de toutes les parties de l'*overlay*. L'enregistrement d'un nom peut prendre quelques minutes, ce que le demandeur peut tolérer. Ce coût est payé une fois pour toutes, les accès au nom certifié sont ensuite rapides.

Dans la section 7.1, nous analysons les contraintes de sécurité auxquelles le processus d'enregistrement doit répondre.

Dans la section 7.2, nous proposons le système d'enregistrement des noms. À la fin de cette procédure, un pair obtient un certificat liant sa clé publique à un nom unique et les autres pairs peuvent obtenir ce certificat.

Dans la section 7.3, nous évaluons la sécurité du système proposé en revenant sur les contraintes définies dans la section 7.1.

Enfin, dans la section 7.4, nous présentons deux cas d'utilisation :
– l'accélération du déploiement de SybilGuard en simplifiant la création des liens entre les entités SybilGuard ;
– un système de voix sur IP pair-à-pair permettant l'authentification par nom d'utilisateur et mot de passe, sans devoir transporter sa clé privée avec soi.

7.1 Sécurité de l'enregistrement

La possession d'une identité est matérialisée par un certificat qui doit être stocké dans la DHT. Afin de fournir une identité unique à un pair honnête A, le système proposé doit assurer que si A obtient un certificat, alors A possède le seul certificat pour ce nom. Il faut donc :

1. empêcher un attaquant d'obtenir un certificat pour le même nom que A *après la demande* de A ;
2. notifier A si un certificat a déjà été obtenu *avant sa demande* pour cette même identité ;
3. empêcher un attaquant d'obtenir un certificat pour le même nom que A *pendant sa demande*.

7.1.1 Protection après la demande

Afin d'empêcher un attaquant de demander l'attribution du nom *name* après l'enregistrement par A, A insère le certificat obtenu dans la DHT à l'emplacement $h(name)$. Ensuite, lors de la signature d'un nouveau certificat, l'autorité de certification distribuée vérifie si un certificat est déjà enregistré pour ce nom, auquel cas la signature est refusée.

Le stockage dans la DHT, s'il est réalisé de manière permanente et sûre, permet à la fois d'être trouvé par les autres pairs et de maintenir l'unicité des noms enregistrés.

7.1.2 Protection avant la demande

Afin de garantir l'unicité des certificats, le système doit empêcher un attaquant d'obtenir un certificat pour un nom déjà enregistré mais doit également empêcher un attaquant d'obtenir un certificat et de ne pas l'enregistrer dans la DHT. Dans ce cas, l'attaquant possèderait le certificat pour un nom donné mais un pair honnête serait autorisé ensuite à enregistrer ce même nom, sans savoir qu'un attaquant possède déjà un certificat pour ce nom : l'attaquant possèderait ainsi le même nom qu'un pair honnête et pourrait se faire passer pour lui.

Toute demande de certificat doit donc laisser une trace dans la DHT avant que soit fourni le certificat signé. Avant de demander un certificat pour un nom *name*, un pair doit déposer une valeur signée par lui à l'emplacement $h(name)$. L'autorité de certification distribuée vérifie ce dépôt et ne procède à la signature distribuée que si le dépôt est effectif. Un pair honnête souhaitant le nom *name* peut ensuite vérifier si un certificat a déjà pu être délivré pour ce nom, même si son possesseur ne l'a finalement pas enregistré dans la DHT.

7.1.3 Protection pendant la demande

Un attaquant ne doit pas pouvoir demander l'obtention du nom *name* pendant qu'un pair honnête formule sa demande pour ce nom. Un attaquant peut, par exemple, être sollicité durant cette certification et tenter d'obtenir le même nom en parallèle. Entre le moment où un pair demande son certificat et le moment ou ce pair insère ce certificat dans la DHT, aucun autre pair ne doit être autorisé à demander le même nom.

Afin d'empêcher un attaquant de demander l'attribution du nom *name* pendant l'enregistrement par A, l'enregistrement nécessite une étape préliminaire durant laquelle A dépose une demande d'obtention du nom *name* qu'il garde secret : il obtient une preuve d'antériorité de la demande. Le nom *name* n'est pas divulgué durant cette étape mais un condensat liant *name* à la clé publique de A est publié. La certification, durant laquelle le nom *name* est publié, ne peut avoir lieu qu'après un certain délai : lors de la publication du nom *name*, il est trop tard pour un attaquant pour enregistrer ce même nom avant que le certificat honnête ne soit inséré dans la DHT.

De plus, pour empêcher qu'un attaquant piège un membre honnête en obtenant une preuve d'antériorité puis en attendant que le membre honnête tente d'insérer le même nom, le temps pour enregistrer le nom demandé est limité et ne peut être renouvelé.

Nous résolvons ces contraintes en découpant l'enregistrement en plusieurs étapes successives, l'enregistrement dépendant de la succession de ces étapes selon des contraintes temporelles. Dans le processus que nous proposons dans la section 7.2, il reste cependant un cas de certificat conflictuel : nous expliquons que ce cas est peu probable et montrons dans la section 7.3 comment le pair honnête peut détecter la présence d'un tel certificat conflictuel.

7.2 Processus d'enregistrement

Le processus d'enregistrement est décomposé en cinq étapes qui sont décrites dans cette section ; l'ensemble est résumé sur la figure 7.1. Nous considérons un pair A qui souhaite enregistrer le nom *name*. Nous évaluons la sécurité de ce processus, son adéquation aux contraintes décrites précédemment et le détail des recherches dans la DHT dans la section 7.3.

7.2.1 Première étape : prise de date

Le pair A, dont la clé publique est P_A et qui souhaite obtenir le nom *name*, prend date pour l'obtention de ce nom tout en gardant ce nom secret.

Si la date [1] courante est $date_0$, alors A génère un message contenant :

- $h(name, P_A)$: cette partie lie la chaîne *name* à la clé P_A sans révéler ni la valeur *name*, ni le condensat de *name*. Un attaquant observant ce message ne peut pas prendre date pour le même nom car il n'a pas la même clé publique ;
- $date_0$: cette date marque le début de la période de durée Δ allouée à A pour obtenir son certificat final.

A signe ce message avec sa clé privée S_A et stocke cette version signée dans la DHT à l'emplacement $h(name, P_A)$. Ce message de prise de date est stocké indéfiniment et ne peut pas être mis à jour : A a la possibilité d'obtenir le certificat pour le nom *name* jusqu'à la date $date_0 + \Delta$. Passé cette date, A ne sera plus *jamais* autorisé à enregistrer *name* car ce message de prise de date est fixe.

La prise de date, combinée avec le certificat d'antériorité obtenu à l'étape 2, empêche un attaquant d'obtenir le même certificat qu'un pair honnête pendant la certification légitime.

7.2.2 Deuxième étape : certificat d'antériorité

Le pair A obtient un certificat signé par la clé de réseau prouvant sa volonté d'obtenir le nom *name*. Le pair ne peut communiquer *name* directement car un attaquant pourrait alors demander le même nom avant la fin de la certification : comme le processus de certification n'est pas atomique, les deux demandes entreraient en conflit. Le pair doit donc communiquer une valeur le liant à *name* mais sans fournir *name*. L'unicité du nom demandé n'est pas vérifiée lors de cette étape.

1. Le terme *date* est utilisé dans le sens général jour et heure.

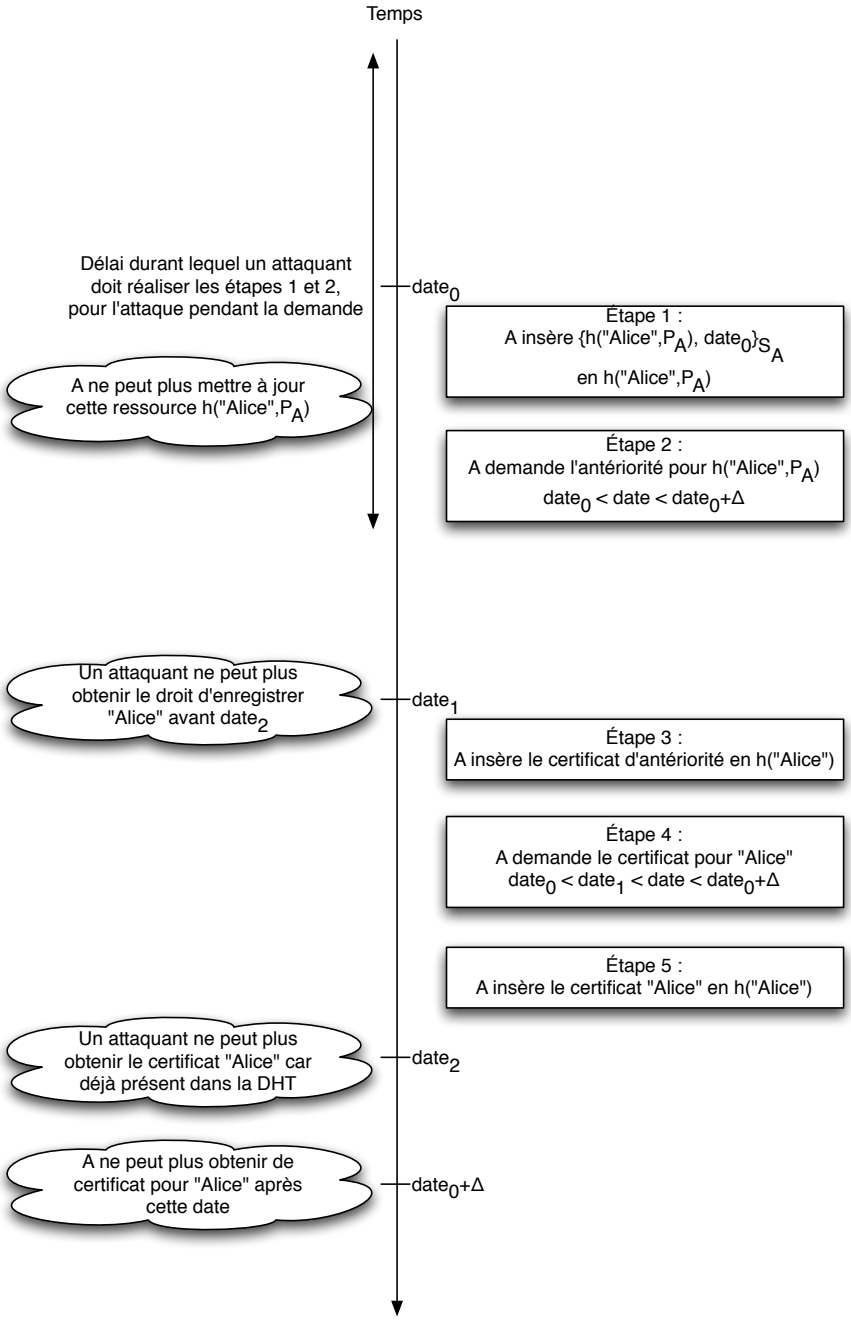


FIGURE 7.1 – Les cinq étapes de l'enregistrement du nom "Alice" par le pair A.

Le pair A souhaitant obtenir le nom *name* transmet une demande de certification $Req = (Content, Clues)$ telle que :

- $Content$ vaut $(h(name, P_A), date_0, date_1)$, où P_A est la clé publique de A et $date_1$ est la date à partir de laquelle A aura le droit d'obtenir le certificat pour le nom *name* ;
- $Clues$ est vide lors de cette étape.

Chaque pair sollicité lors de la signature distribuée vérifie la présence de la ressource $h(name, P_A)$ stockée lors de l'étape 1, valide la date $date_0$ proposée dans $Content$ qui doit être égale à celle de la ressource stockée, et coopère si :

- la date $date_1$ proposée est suffisamment éloignée ;
- $date_0 < date < date_0 + \Delta$, où $date$ est la date courante.

A sera autorisé à obtenir le certificat *name* à partir de la date $date_1$ et A ne dévoilera *name* qu'à ce moment-là : cette date doit donc être suffisamment éloignée pour empêcher un attaquant d'obtenir l'autorisation d'enregistrer *name*, révélé par A lors de la quatrième étape, avant que A n'ait enregistré son certificat dans la DHT. La requête contient un condensat de *name* avec P_A, ce qui lie le nom demandé à la clé publique de A : un attaquant ne peut pas utiliser un certificat créé pour cette clé car il ne connaît pas la clé secrète associée.

La preuve d'antériorité n'est signée que si elle est demandée pendant l'intervalle annoncé lors de la première étape.

Si la date $date_1$ proposée est suffisamment éloignée et si la date actuelle est suffisamment proche de la date $date_0$ enregistrée lors de l'étape 1, A obtient un certificat d'antériorité signé par la clé secrète du réseau. Ce certificat d'antériorité empêche un attaquant d'obtenir le même certificat qu'un pair honnête pendant la certification légitime.

Les premières expérimentations sur PlanetLab avec 100 groupes de fragmentation, ce qui correspond à un arbre de signature à environ 7 étages, montrent qu'un certificat est obtenu en environ 10 secondes dans ces conditions. Avec 100 000 groupes de fragmentation (et donc plus de 2 000 000 de membres), les arbres de signature ont environ 16 étages, ce qui laisse penser qu'une signature distribuée prendrait environ 30 secondes. Une date $date_1$ située 10 minutes dans le futur doit donc permettre à A d'obtenir son certificat final avant qu'un attaquant ne soit autorisé à demander un certificat pour le même nom.

7.2.3 Troisième étape : insertion du certificat d'antériorité

Le pair A insère son certificat d'antériorité dans la DHT et révèle le condensat $h(name)$ de l'identité voulue *name*.

A attend tout d'abord que la date $date_1$ soit atteinte : à partir de ce moment-là, il peut procéder successivement aux trois dernières étapes lui permettant d'enregistrer *name*. A insère alors son certificat d'antériorité en $h(name)$, emplacement où résidera également le certificat final.

La certification finale vérifie la présence de ce certificat d'antériorité en $h(name)$: A doit donc nécessairement insérer son certificat d'antériorité pour obtenir son certificat final. Une fois ce certificat d'antériorité inséré, un pair B souhaitant obtenir le nom *name* constatera sa présence et saura qu'un autre pair a pu obtenir ce certificat précédemment, même si cet autre pair n'a pas stocké son certificat final dans la DHT. Ce pair B pourra ainsi choisir un autre nom pour lequel aucun certificat n'a été généré.

Cette étape permet d'empêcher un attaquant d'obtenir un certificat tout en laissant un pair honnête obtenir un certificat pour le même nom.

7.2.4 Quatrième étape : obtention du certificat final

Le pair A obtient son certificat final liant sa clé publique au nom demandé. Cette quatrième étape ne peut démarrer qu'une fois la date $date_1$ révolue et avant la date $date_0 + \Delta$. Le pair A transmet alors une requête de certification $Req = (Content, Clues)$ telle que :

- $Content$ vaut $(name, P_A)$;
- $Clues$ contient le certificat d'antériorité précédemment obtenu, certificat qui contient $(h(name, P_A), date_0, date_1)$ signé avec la clé du réseau.

Chaque pair sollicité coopère si :

- la date $date_1$ est passée, ce qui empêche un attaquant de voler le nom réservé en créant une nouvelle demande quand le pair A révèle $name$ lors de la troisième étape ;
- la date $date_0 + \Delta$ n'est pas passée, ce qui empêche un attaquant de prendre date tôt pour finalement attendre qu'un pair honnête tente d'obtenir le même certificat, auquel cas l'attaquant peut obtenir un certificat pour le même nom ;
- la valeur $h(name, P_A)$ précédemment signée correspond bien au condensat des valeurs proposées dans $Content$;
- la ressource $h(name)$ de la DHT ne contient pas de certificat final mais contient bien le certificat d'antériorité.

Les trois premières vérifications permettent de vérifier la validité de la requête. La dernière vérification permet de garantir l'unicité des noms certifiés. En effet, un pair de chaque groupe de fragmentation réalise une requête dans la DHT, ce qui correspond à des routes redondantes ayant des points de départ différents vers une même ressource. En l'absence d'attaque sybile sur la ressource, un attaquant ne peut pas masquer la ressource $h(name)$ à tous les pairs la sollicitant.

Si le nom $name$ est libre, le pair A obtient finalement la signature de son certificat, signature qui vaut $(h(name, P_A))^e[m]$.

7.2.5 Cinquième étape : insertion du certificat final et vérification

A enregistre ce certificat dans la DHT à l'emplacement $h(name)$ et la distribution des clés est ainsi assurée. Un pair recherchant la clé publique de A peut l'obtenir en connaissant son pseudonyme, avec une requête vers $h(name)$. Aucun autre pair ne peut plus ensuite usurper le pseudonyme de A.

7.3 Évaluation de la sécurité

Dans cette section, nous évaluons la sécurité du processus d'enregistrement et montrons qu'il permet de protéger l'unicité des noms après la demande, avant la demande et pendant la demande.

7.3.1 Protection après la demande

Lorsqu'un pair honnête a acquis le nom *name*, aucun autre pair ne doit pouvoir obtenir de certificat pour *name*. Lors de l'étape 5, le pair enregistre son certificat dans la DHT à l'emplacement $h(name)$; ensuite, dès lors qu'un pair demande le même nom, le certificat ne sera pas signé lors de l'étape 4 si un des pairs sollicités (un pair par groupe de fragmentation) trouve le certificat déjà présent dans la DHT.

L'unicité du nom est donc protégée après la demande, si tous les pairs de l'un des groupes de fragmentation trouvent l'ancien certificat et refusent de participer à la nouvelle signature distribuée. Nous présentons dans cette sous-section l'impact des caches et d'une attaque par déni de service sur les recherches dans une DHT.

7.3.1.1 Gestion des caches

Lors de la signature du certificat, un pair de chaque groupe de fragmentation doit vérifier la présence de ce certificat. Lorsque le nombre de groupes de fragmentation augmente, cette demande augmente la charge subie par les réplicats de cette ressource et les mécanismes de cache de ressource sont généralement utilisés pour tolérer cette charge.

Dans le cas de Kademlia, chaque pair obtenant une ressource la stocke en cache sur le pair qu'il connaît de plus proche de la ressource mais ne connaissant pas cette ressource. L'absence de ressource, nécessaire dans notre cas, pourrait être mise en cache de la même manière. Dans notre cas, ce système de cache doit être renforcé et chaque pair acceptant de jouer le rôle de cache doit s'assurer qu'il stocke la bonne valeur, sinon un attaquant pourrait corrompre les caches de beaucoup de pairs :

- en leur faisant stocker l'absence d'un certificat final, en fait existant ;
- en leur faisant stocker la présence d'une donnée (par exemple un certificat d'antériorité), en fait non insérée dans la DHT.

Lors de la recherche d'une ressource dans Kademlia, le pair A exécute un processus itératif afin d'obtenir les k pairs les plus proches de la ressource, au sens de la distance XOR, pour ensuite leur demander la valeur de la ressource. Durant le processus itératif, si un pair consulté B possède la donnée en cache, il renvoie cette donnée et le processus est interrompu. Enfin, A stocke cette donnée sur le pair C le plus proche obtenu mais ne connaissant pas cette ressource. Ce pair C est ensuite utilisé comme cache et doit s'assurer que la donnée stockée est la bonne :

- si A a obtenu la donnée depuis les réplicats responsables de cette ressource, A possède les réponses datées et signées des k réplicats ainsi que leurs certificats d'appartenance au réseau. A transmet ces réponses à C qui évalue localement leur validité par un test de densité, comme proposé dans [CDG+02]. Comme les pairs sont uniformément distribués, leur densité est proche dans chaque partie de l'*overlay* et C peut comparer la densité des réplicats proposés avec son voisinage. Si la densité est proche, C sait que ces pairs sont bien les réplicats de cette ressource et accepte la valeur retournée. Sinon, C ne stocke pas la valeur retournée ;
- si A a obtenu la donnée depuis un cache, ce cache lui a également fourni la liste des réponses des réplicats réels. A transmet cette liste à C.

Afin d'éviter d'obtenir une route contrôlée par des attaquants, A utilise un routage redondant avec α routes différentes pour trouver les k réplicats.

En complément, le pair demandant l'enregistrement d'un nom peut également joindre les réponses signées des k réplicats à sa demande de certification ; les pairs sollicités lors de la signature distribuée peuvent ainsi réaliser le test de densité sur ces k réplicats et, le cas échéant, ne pas avoir besoin d'obtenir la ressource depuis la DHT. Il sera intéressant d'étudier différentes stratégies et paramètres pour trouver le bon compromis entre la sécurité et l'efficacité.

7.3.1.2 Attaque par déni de service

Malgré ces précautions, un attaquant peut tenter de détruire un certificat stocké en menant une attaque par déni de service sur l'ensemble des pairs le répliquant, auquel cas une seconde certification pour le même nom peut être réalisée.

Une attaque par déni de service demande cependant des ressources de bande passante importantes, qui peuvent par exemple être obtenues par la location d'un *botnet*. Une telle attaque a donc un coût non négligeable et bien supérieur à la corruption de caches par exemple. De plus, cette attaque peut être détectée :

- soit par les réplicats stockant le certificat conflictuel lorsque les pairs attaqués redeviennent disponibles, car les deux certificats conflictuels sont alors stockés par les mêmes pairs ;
- soit par la victime en recherchant périodiquement son certificat dans la DHT et en vérifiant la réponse obtenue.

7.3.2 Protection avant la demande

Lorsqu'un pair honnête acquiert le nom *name*, ce pair doit être sûr qu'aucun autre pair n'a pu obtenir un certificat pour le même nom précédemment. Si un autre pair avait obtenu un tel certificat (étape 4), alors ce pair aurait forcément inséré son certificat d'antériorité lors de l'étape 3, ce qui est testé à l'étape 4.

Un pair peut donc vérifier qu'aucun certificat n'existait pour un nom donné car les pairs participant à la signature distribuée lors de l'étape 4 vérifient qu'il n'existe pas d'autre certificat d'antériorité plus ancien.

7.3.3 Protection pendant la demande

Enfin, lorsqu'un pair enregistre le nom *name*, un autre pair ne doit pas pouvoir demander le même nom au même moment. Pour enregistrer *name*, un pair A doit avoir obtenu un certificat d'antériorité plusieurs minutes auparavant : ce certificat d'antériorité ne permet de connaître ni le nom souhaité, ni son condensat et ne peut pas être utilisé par un attaquant puisqu'il est lié à la clé publique de A.

Lorsque le nom est révélé, un pair B peut tenter d'obtenir le même nom. Dans ce cas, ce pair B doit d'abord obtenir son certificat d'antériorité, puis attendre le délai fixé avant d'obtenir ce certificat à l'étape 4. Or, après ce délai, le pair A a enregistré son certificat dans la DHT et la certification de B est refusée lors de l'étape 4.

La seule solution pour le pair B d'obtenir le même nom au même moment est d'anticiper l'obtention du certificat d'antériorité. Cependant, B n'a ni la

connaissance du nom demandé, ni la connaissance du moment où ce nom est demandé : la seule connaissance de B est qu'*un* nom est demandé à un instant. Pour obtenir le certificat d'antériorité, B doit insérer sa demande de prise de date, qui a une durée limitée (temps Δ), et doit donc demander le bon nom au bon moment (B ne peut plus mettre à jour sa prise de date ensuite). Il existe donc éventuellement, si l'attaquant devine le bon nom, une fenêtre de vulnérabilité durant laquelle il peut obtenir un certificat identique à A, comme illustré figure 7.1. Nous estimons, cependant, cette attaque suffisamment difficile pour être peu probable ; de plus, le pair A peut détecter cette double certification par la présence de deux certificats d'antériorité en $h(name)$.

En dehors de cette attaque difficile, un attaquant ne peut donc pas obtenir le même nom qu'un autre pair pendant son enregistrement.

7.4 Exemples d'utilisation

Nous décrivons dans cette section deux utilisations du service de nommage sécurisé, pour faciliter le déploiement de SybilGuard et pour proposer un système de voix sur IP pair-à-pair.

7.4.1 Utilisation avec SybilGuard

Dans la version originale de SybilGuard, la création des arcs de confiance entre les utilisateurs peut freiner le déploiement du réseau. En effet, pour créer un arc entre eux, deux amis doivent établir une communication sécurisée entre eux avec authentification mutuelle. Cette étape, complexe, nuit grandement au développement d'un réseau utilisant SybilGuard.

D'un point de vue cryptographique, l'authentification mutuelle entre deux utilisateurs peut être réalisée de deux manières :

- en cryptographie symétrique, les deux utilisateurs connaissent une même valeur secrète ;
- en cryptographie asymétrique, chaque utilisateur connaît la clé publique de l'autre utilisateur.

Nous choisissons une authentification mutuelle par cryptographie asymétrique et chaque utilisateur doit donc obtenir la clé publique de chacun de ses amis de manière sécurisée. Nous écartons la possibilité d'échanger ces clés par un moyen externe, ce qui est peu pratique et dépend de la sécurité du moyen externe utilisé.

Nous différencions les utilisateurs en deux catégories :

- ceux qui ont obtenu leur certificat d'appartenance au réseau sont dits *membres*. Nous considérons que ces utilisateurs ont de plus enregistré un nom intelligible dans la DHT permettant d'obtenir leur clé publique ;
- ceux qui n'ont pas encore obtenu leur certificat d'appartenance au réseau sont dits *extérieurs*. N'appartenant pas au réseau, ces utilisateurs n'ont pas pu enregistrer de nom dans la DHT. Bien qu'ils n'appartiennent pas au réseau pair-à-pair structuré support de la DHT, ces extérieurs doivent pouvoir échanger des clés cryptographiques : leur insertion dans le réseau SybilGuard (en créant des arcs de confiance avec d'autres membres) est nécessaire pour obtenir leur certificat d'appartenance au réseau, comme proposé dans le chapitre 6.

Il existe donc trois possibilités pour l'authentification mutuelle de deux utilisateurs :

1. membre et membre ;
2. membre et extérieur ;
3. extérieur et extérieur.

Le système d'admission étant conçu pour que tous les utilisateurs honnêtes puissent rejoindre l'*overlay* et donc devenir membres, le troisième cas entre deux extérieurs souhaitant rejoindre le réseau simultanément est peu fréquent : nous proposons dans cette sous-section une authentification mutuelle uniquement dans les deux premiers cas.

7.4.1.1 Authentification mutuelle entre membre et membre

Deux utilisateurs membres souhaitant créer un arc SybilGuard entre eux doivent chacun obtenir la clé publique de l'autre. Comme chacun de ces utilisateurs a enregistré un nom intelligible dans la DHT, chacun peut obtenir la clé publique présente dans le certificat de ce nom. Les deux pairs ont donc uniquement besoin de connaître leurs pseudonymes respectifs pour s'ajouter comme amis (comme dans un système traditionnel de messagerie instantanée).

7.4.1.2 Authentification mutuelle entre membre et extérieur

Un utilisateur membre A et un utilisateur extérieur B souhaitant créer un arc doivent également échanger leurs clés publiques. L'utilisateur extérieur B obtient la clé publique de l'utilisateur membre A puis lui transmet sa clé publique.

Pour pouvoir obtenir les certificats des utilisateurs membres, les pairs extérieurs obtiennent un accès restreint à la DHT. Ces pairs ne participent pas au routage, ne peuvent pas insérer de données et ne stockent pas de données : ils sont uniquement connectés à un membre du réseau qui retransmet leurs requêtes visant à obtenir les certificats d'autres membres. Connaissant le pseudonyme de A, B obtient ainsi son certificat et sa clé publique.

B doit ensuite transmettre sa clé publique à A et A doit être convaincu qu'il a bien reçu la clé de B. B transmet un message à A en utilisant la DHT pour joindre A : ce message, chiffré avec la clé publique de A [2], contient la clé publique de B et un identifiant de B pour A. Ce message étant chiffré, seul A peut le lire en utilisant sa clé secrète et aucun attaquant ne peut obtenir l'identifiant utilisé par B. Cet identifiant, par exemple le nom de B, permet à A de savoir qui lui a envoyé ce message et d'accepter la clé publique, le cas échéant, ce qui est sûr si :

- A sait qu'il attend un message de B ;
- les attaquants soit ne connaissent pas l'identité de B, soit ne connaissent pas l'instant où A attend le message de B.

Cette authentification mutuelle est, bien sûr, moins robuste que l'authentification entre deux membres du réseau. Nous pensons, cependant, qu'elle permet un niveau de sécurité satisfaisant tout en étant plus pratique et plus sécurisée

2. En pratique, une même paire de clés ne devant pas être utilisée pour réaliser à la fois du chiffrement et de la signature, chaque utilisateur possède deux paires de clés distinctes pour chiffrer et signer.

qu'un échange par courrier électronique n'utilisant pas PGP. Ce système a été mis en place dans notre implémentation de SybilGuard utilisant l'autorité de certification distribuée, implémentation qui a été principalement réalisée dans le cadre du stage de Laurent Bonnet, en quatrième année d'INSA, encadré par l'auteur.

7.4.2 Utilisation pour de la messagerie instantanée ou voix sur IP

Dans le cadre d'une application de messagerie instantanée ou de voix sur IP, il est intéressant de pouvoir s'interfacer avec les logiciels déjà existants, prévus pour un environnement non pair-à-pair. Ces logiciels sont généralement conçus pour s'authentifier par identifiant et mot de passe, et non pas avec une clé publique. Cette authentification par mot de passe permet également de se connecter facilement depuis plusieurs machines (au domicile, au travail, dans un cyber-café) sans avoir besoin de toujours garder sur soi une clé USB ou une carte à puce contenant sa clé privée. Nous nous intéressons plus précisément au protocole SIP de voix sur IP et proposons une authentification par mot de passe.

7.4.2.1 Fonctionnement de SIP

Le protocole SIP est un protocole de signalisation pour de la voix sur IP : ce protocole gère l'authentification, la connexion, la recherche du correspondant, la négociation des paramètres, etc., mais pas le transport des données audio ensuite échangées. Chaque utilisateur crée un compte sur un serveur sur lequel il renseigne son adresse IP ; de manière symétrique, la recherche d'un correspondant passe par l'obtention de son adresse IP en contactant son serveur.

Les identifiants sont de la même forme que les adresses de courrier électronique, à savoir `username@server.org` : pour contacter cet utilisateur, il faut demander son adresse IP courante à `server.org`. Les deux instances SIP clientes communiquent ensuite directement.

La mise en relation de deux utilisateurs dans un système SIP avec serveurs est illustrée figure 7.2.

Nous nous intéressons aux deux messages principaux :

- *Register* qui permet de s'authentifier sur son serveur et de mettre à jour son adresse IP. Ce message est envoyé à son propre serveur et contient notamment le nom d'utilisateur, son mot de passe ainsi que son adresse IP courante ;
- *Invite* qui permet d'obtenir l'adresse IP du correspondant souhaité. Ce message est également envoyé à son propre serveur qui se charge de la recherche et contient le nom complet `username@server.org` de l'utilisateur recherché. Le serveur répond avec l'adresse IP de l'utilisateur à contacter directement.

7.4.2.2 SIP pair-à-pair avec authentification par mot de passe

Chaque utilisateur exécute localement un pair de l'*overlay* qui lui sert de serveur : le serveur de chaque utilisateur SIP est `localhost`, comme illustré sur

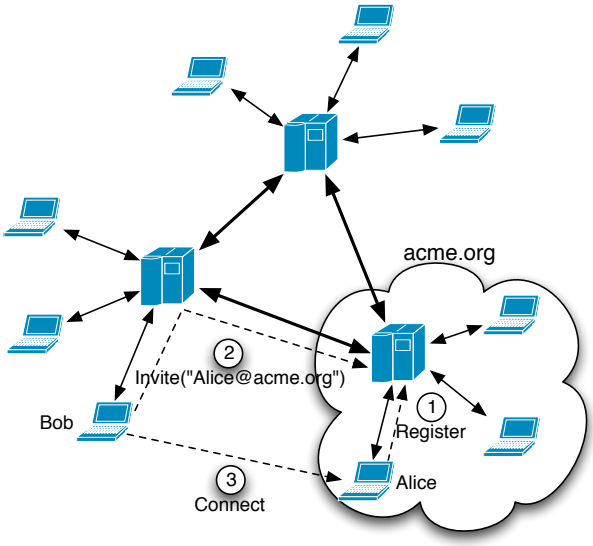


FIGURE 7.2 – Mise en relation de Bob avec Alice, enregistrée sur le serveur `acme.org`. Alice envoie son message *Register* à son serveur. Bob envoie le message *Invite* à son serveur, qui retransmet à `acme.org`, et Bob obtient l'adresse IP d'Alice. Bob contacte enfin directement le client SIP d'Alice.

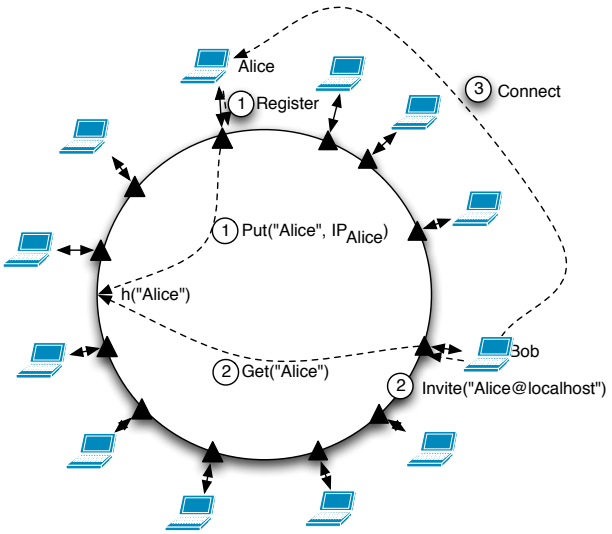


FIGURE 7.3 – Mise en relation de Bob avec Alice. Alice envoie son message *Register* à son pair qui insère son enregistrement dans la DHT. Bob envoie le message *Invite* à son pair qui recherche "Alice" dans la DHT ; Bob obtient ainsi l'adresse IP d'Alice. Enfin, Bob contacte directement le client SIP d'Alice, comme dans la version originale de SIP.

la figure 7.3. Les messages *Register* et *Invite* envoyés pas l'application SIP sont donc reçus par le pair local et sont utilisés de la manière suivante :

- *Register* correspond soit à l'enregistrement d'un nouveau nom, soit à la connexion sur un nom existant. Dans tous les cas, ce message contient le nom d'utilisateur, le mot de passe et l'adresse IP du client SIP. Le pair effectue une recherche dans la DHT pour obtenir le certificat associé à ce nom :
  - si ce certificat n'existe pas, le pair considère qu'il s'agit d'un nouvel enregistrement. Dans ce cas, le pair enregistre le nom demandé selon le protocole proposé dans ce chapitre : l'enregistrement ne se déroule avec succès que si le nom est effectivement libre. Le pair chiffre ensuite la clé secrète associée au certificat avec le mot de passe de l'utilisateur (*password-based encryption*, défini notamment dans le PKCS #5 [RSA99]) et stocke cette clé secrète chiffrée avec le certificat, au sein de l'*overlay*. La clé secrète chiffrée est téléchargeable par tous mais n'est déchiffrable que par son possesseur, à l'aide de son mot de passe ;
  - si ce certificat existe déjà, le pair télécharge la clé secrète chiffrée et la déchiffre avec le mot de passe proposé. Un test de chiffrement entre la clé secrète déchiffrée et la clé publique permet de signaler à l'utilisateur si l'authentification s'est déroulée avec succès ; si le mot de passe entré par l'utilisateur n'est pas le bon, la clé ne peut pas être déchiffrée et l'authentification échoue. Le pair met ensuite à jour l'adresse IP dans son certificat stocké dans l'*overlay*. Cette mise à jour ne fait pas intervenir la signature distribuée car cette mise à jour doit uniquement être signée par la clé secrète de l'utilisateur correspondant.
- *Invite* correspond à la recherche d'un autre utilisateur. Le pair obtient le certificat de l'utilisateur recherché par une requête dans la DHT, vérifie la signature du certificat avec la clé publique du réseau et transmet au client l'adresse IP de l'utilisateur recherché, contenue dans le certificat obtenu.

Le chiffrement par mot de passe offre une sécurité moindre que l'authentification forte par clé publique mais est néanmoins suffisant pour ce type d'application. Dans ce chiffrement, une clé de chiffrement symétrique (par exemple AES) est dérivée à partir du mot de passe. L'espace des clés possibles est réduit, tout comme dans une authentification par mot de passe classique couramment utilisée. Afin d'empêcher les attaques de déchiffrement par essais successifs sans connaître le mot de passe, deux protections sont mises en place :

- chaque chiffrement utilise, en plus du mot de passe, une graine aléatoire publique. Chaque chiffrement doit donc être attaqué séparément, deux mots de passe identiques ne génèrant pas la même clé ;
- le processus de dérivation de la clé à partir du mot de passe est relativement long. Ce processus reste simple pour dériver une seule clé mais augmente le temps nécessaire pour tester un grand nombre de clés.

Cette version pair-à-pair de SIP avec un système de nommage sécurisé a été implémentée dans notre prototype et testée avec succès avec trois clients SIP non modifiés (XMeeting [3], iSoftPhone [4] et Ekiga [5]). L'obtention de la clé publique

3. `http://xmeeting.sourceforge.net`
4. `http://www.call4mac.com`
5. `http://ekiga.org`

du correspondant pourrait également permettre de laisser un message vocal en cas d'absence, message qui ne serait déchiffrable que par son destinataire.

Résumé

Dans ce chapitre, nous avons proposé un système de distribution des clés cryptographiques. Les clés sont obtenues à partir de la connaissance du pseudonyme de l'utilisateur recherché, que cet utilisateur soit connecté ou non. La signature de ces clés par l'autorité de certification distribuée contraint l'unicité des identifiants. De plus, suite à un coût temporel initial supporté par le possesseur de la clé, l'obtention de la clé d'un autre utilisateur est réalisée avec une requête simple sans redondance et donc une latence minimale.

Dans la section 7.1, nous avons analysé les contraintes de sécurité sur l'enregistrement d'un nouveau nom unique. Nous avons montré comment empêcher un attaquant d'obtenir le même certificat qu'un pair honnête, que ce soit avant, pendant ou après l'obtention par le pair honnête.

Dans la section 7.2, nous avons proposé la procédure d'enregistrement d'un nom unique. Cette procédure en cinq étapes permet d'empêcher un attaquant d'obtenir le nom d'un pair honnête avant, après ou pendant la certification légitime. Durant cette procédure, les pairs de l'autorité de certification distribuée vérifient l'unicité du nom demandé.

Dans la section 7.3, nous avons évalué l'adéquation entre la procédure d'enregistrement proposée dans la section 7.2 et les contraintes de sécurité définies dans 7.1. Nous avons montré que la procédure proposée satisfait les contraintes, ne laissant qu'une possibilité d'attaque peu probable. Cette attaque peut de plus être détectée *a posteriori*, ce qui permet au pair honnête de savoir que son enregistrement est caduc.

Enfin, dans la section 7.4, nous avons présenté deux cas d'utilisation de la distribution de clés. Le premier permet de déployer simplement SybilGuard, en échangeant uniquement des pseudonymes au lieu d'échanger des condensats de clés publiques. Le second est un système de voix sur IP SIP pair-à-pair permettant l'authentification des pairs : la clé privée est stockée dans l'*overlay* et est déchiffrée grâce à un mot de passe, évitant à l'utilisateur de devoir garder sa clé privée avec lui quand il change de machine. La distribution de clés trouve aussi une utilisation dans les systèmes de fichiers distribués, où il devient possible d'accorder un droit en lecture ou écriture à un utilisateur nommé.

Dans le tableau 7.1, nous comparons notre proposition aux autres solutions présentées dans l'état de l'art. Ce tableau prend en compte la sécurité apportée par la solution (un échange de clé par mail n'est par exemple pas considéré sécurisé), la minimisation de la latence des requêtes et l'absence de centre.

Il serait nécessaire de proposer le renouvellement des certificats. Chaque certificat aurait une durée de vie limitée et serait renouvelé avec un nouveau couple de clés publique/privée, ce qui est effectué avec les autorités de certification centralisées. Le renouvellement permettant d'invalider les anciennes versions de la clé, un attaquant aurait alors un temps limité pour attaquer la clé privée. Ce renouvellement serait particulièrement intéressant dans le cas de la voix sur IP

6. La sécurité dépend de la redondance utilisée et de l'absence d'attaque éclipse sur le pair demandant la ressource.

	Sécurité	Latence min.	Absence de centre
Distribution explicite	X	X	√
Obtention du condensat	X	X	√
Requête redondante	√[6]	X	√
Autorité de certification	√	√	X
Notre proposition	√	√	√

TABLE 7.1 – Comparaison de notre proposition de nommage sécurisé avec les quatre approches présentées dans l'état de l'art.

où la clé secrète est chiffrée par le mot de passe de l'utilisateur, l'attaquant ayant ainsi un temps limité pour deviner le mot de passe.

Conclusion

Nous avons étudié dans ce mémoire la sécurité d'un type de réseau pair-à-pair, les réseaux structurés. En l'absence de centre et donc d'autorité ponctuelle, les solutions de sécurité habituelles ne sont pas transposables directement à ces réseaux. Dans ce mémoire, nous avons donc proposé de nouveaux mécanismes de sécurité pour les réseaux pair-à-pair structurés : une autorité de certification distribuée ainsi que deux applications de cette autorité.

Nous dressons ici un bilan de nos propositions, puis introduisons diverses perspectives pour de futurs travaux.

Bilan

Construction de l'autorité de certification distribuée

Dans la première partie de ce mémoire, nous avons présenté la construction de l'autorité de certification distribuée. L'autorité est représentée par un couple de clés RSA publique/secrète : la clé publiqué est connue de tous et la clé secrète est fragmentée dans le réseau.

Nous avons présenté le modèle de l'autorité de certification distribuée dans le chapitre 2. Le réseau est décomposé en groupes de fragmentation, chaque groupe connaissant un fragment de la clé et la somme des fragments étant égale à la clé secrète du réseau. La signature d'un certificat requiert la participation d'un pair de chaque groupe de fragmentation. La taille des groupes de fragmentation étant fixe, ces groupes se divisent ou fusionnent en fonction de l'évolution de la taille du réseau. Un membre de chaque groupe devant participer à chaque certification, le ratio de pairs devant coopérer pour obtenir un certificat est constant et ne dépend pas de la taille du réseau. Ainsi, un certificat ne peut être signé que si une fraction non négligeable des pairs accepte de coopérer.

À partir de cette fragmentation de la clé secrète, nous avons proposé dans le chapitre 3 un algorithme de signature arborescent permettant de signer un certificat légitime en répartissant la charge sur les pairs impliqués. Certains pairs peuvent être malveillants et renvoyer de mauvais chiffrements intermédiaires : l'algorithme de signature permet de sélectionner les pairs honnêtes, afin d'obtenir des signatures valides.

Nous avons également proposé dans le chapitre 4 des opérations permettant la maintenance des groupes de fragmentation (division, fusion, rafraîchissement)

sans synchronisation et sans consensus byzantins. Ces opérations reposent sur la connaissance par chaque groupe d'un arbre de fragmentation définissant tous les sous-fragments ; ces arbres sont modifiés par l'opération de rafraîchissement, afin de conserver la confidentialité des fragments (chaque fragment ne doit être connu que par un groupe).

Cette autorité de certification distribuée a été implémentée et évaluée dans le chapitre 5. L'implémentation a été testée avec succès sur la plate-forme de test PlanetLab. Les autres évaluations, réalisées par simulations et calculs théoriques, montrent que cette autorité reste fonctionnelle avec un nombre raisonnable d'attaquants (environ 20%).

Applications de l'autorité de certification distribuée

Dans la seconde partie de ce mémoire, nous avons présenté deux applications de l'autorité de certification distribuée dans des réseaux pair-à-pair structurés, à savoir une protection contre l'attaque sybile et un service de nommage sécurisé.

Afin de protéger contre l'attaque sybile, l'autorité de certification distribuée est couplée à SybilGuard dans le chapitre 6. L'appartenance au réseau est matérialisée par un certificat qui est signé avec la clé secrète du réseau. La clé secrète étant fragmentée entre les pairs, un membre souhaitant rejoindre le réseau doit, pour obtenir son certificat signé, obtenir la coopération d'un pourcentage fixé des pairs, chacun de ces pairs testant le nouveau pair avec SybilGuard. SybilGuard permet à chaque pair impliqué d'évaluer si le nouveau pair est sybil ou honnête et le nouveau pair n'obtient donc de certificat que si un pourcentage fixé des membres du réseau le juge honnête. De plus, l'identifiant de ce nouveau pair est dérivé du certificat obtenu et le nouveau pair ne peut donc pas influencer ce choix. La protection proposée limite donc à la fois le nombre de pairs sybils par attaquant et le choix des identifiants de pair.

Nous avons également proposé un service de nommage sécurisé permettant la distribution de clés cryptographiques dans le chapitre 7. Chaque pair peut enregistrer un nom intelligible unique dans le réseau. Pour cela, un pair demande un certificat signé par la clé de réseau liant sa clé publique au nom souhaité ; si le nom est libre, le pair obtient ce certificat. Ce pair insère alors ce certificat dans la DHT, afin que les autres pairs puissent l'obtenir et qu'aucun autre pair ne puisse enregistrer le même nom (en effet, lors de la signature d'un certificat, chaque pair impliqué vérifie l'absence de certificat antérieur pour le même nom, ce qui induit une redondance de requêtes puisque beaucoup de pairs sont impliqués). En définitive, ce système vérifie la présence d'un nom avec une très forte redondance lors de l'enregistrement, puis permet d'obtenir le certificat d'un autre utilisateur par une requête non redondante et donc avec une latence minimale.

Perspectives

À la suite de ce travail, nous envisageons les perspectives suivantes :

1. des groupes de taille variable permettraient de tolérer des attaques par déni de service et de diminuer le coût d'une certification ;
2. le renouvellement de la clé de réseau permettrait d'affecter une durée de vie limitée à cette clé ;

3. la délégation de signature à un pair de confiance, lors de la déconnexion, permettrait de diminuer le pourcentage d'attaquants (qui eux restent connectés) ;
4. le remplacement de SybilGuard par un autre test social (SybilLimit ou SybilInfer par exemple) pourrait améliorer la protection contre l'attaque sybile ;
5. des simulations dynamiques permettraient d'évaluer plus précisément la sécurité et le coût du système ;
6. la finalisation de l'implémentation permettrait la mise à disposition publique d'une application SIP pair-à-pair robuste à l'attaque sybile et permettant à chaque utilisateur de posséder un nom unique.

Nous détaillons ces perspectives ci-après.

Groupes de taille variable

Dans le système actuel, les groupes de fragmentation ont une taille comprise entre les bornes fixes g_{min} et g_{max}, ce qui maintient le ratio de pairs devant coopérer à une signature entre $\frac{1}{g_{max}}$ et $\frac{1}{g_{min}}$, quelle que soit la taille du réseau. Ce comportement, explicitement souhaité au début de notre travail, impose cependant certaines limitations :

- le nombre de pairs devant coopérer à une certification étant un pourcentage fixe du nombre de pairs du réseau (entre $2,5\%$ et 5% avec les valeurs utilisées dans ce mémoire), le nombre de pairs devant coopérer et la charge sur ces pairs augmentent linéairement avec la taille du réseau (même si le temps nécessaire à la certification n'augmente que logarithmiquement, grâce à l'algorithme de signature arborescent) ;
- une attaque par déni de service peut être envisagée contre un groupe de quelques dizaines de membres, auquel cas le service de signature distribuée peut être définitivement interrompu ;
- la robustesse de notre système, élevée dans le cadre de notre étude de grands réseaux pair-à-pair, est cependant faible pour de très petits réseaux (jusqu'à 100 pairs).

L'augmentation de la taille des groupes avec la taille du réseau (par exemple logarithmiquement) permet, tout en demandant toujours l'accord à un nombre de pairs significatif, de contourner ces trois limites :

- la taille des groupes augmentant avec la taille du réseau, le nombre de pairs devant coopérer à une certification augmente moins rapidement que la taille du réseau ;
- la taille des groupes est d'autant plus élevée que le risque d'attaque par déni de service augmente (un grand réseau est une cible plus probable), ce qui complique ces attaques par déni de service ;
- de manière symétrique, la taille des groupes dans un petit réseau est plus faible, ce qui augmente la robustesse pour ces petit réseaux.

L'augmentation de la taille des groupes a cependant un coût sur la sécurité et la maintenance, coût qui devra être évalué :

- au niveau de la sécurité, le nombre de pairs devant coopérer à une certification diminue. Si le facteur d'augmentation est bien choisi, ce nombre reste cependant suffisamment élevé pour assurer une sécurité suffisante ;

– au niveau de la maintenance, un plus grand nombre de pairs doit être surveillé dans chaque groupe. Les opérations de maintenance sans synchronisation ni consensus doivent permettre l'augmentation de la taille de groupes et il serait également envisageable de ne surveiller qu'une partie du groupe, en mutualisant le système de surveillance et en gérant les relations de confiance associées.

La taille du réseau peut être évaluée par le niveau du groupe de fragmentation courant (par rapport au fragment racine correspondant à la clé secrète en entier) et une taille de groupe différente peut correspondre à chaque niveau de groupe. Par exemple, les groupes $\mathring{e_0}$ et $\mathring{e_1}$ doivent être constitués de $g_{min} = 20$ à $g_{max} = 40$ membres. Lorsque ces deux groupes se divisent, les groupes créés $\mathring{e_{00}}$, $\mathring{e_{01}}$, $\mathring{e_{10}}$ et $\mathring{e_{11}}$ doivent être constitués de $g_{min} = 20$ à $g_{max} = 50$ membres. Ainsi, plus le réseau est grand, plus le nombre de groupes est élevé et donc plus les groupes ont des identifiants longs, plus la taille de ces groupes est élevée. Cette évolution de la taille des groupes reste cohérente entre les différents pairs puisque tous les pairs d'un même groupe utilisent toujours les mêmes valeurs de g_{min} et g_{max}, prédéfinies pour chaque niveau de groupe.

Renouvellement de la clé de réseau

Toute clé cryptographique ayant une durée de vie limitée (typiquement de une à quelques années), la clé de réseau devrait être renouvelée après une certaine durée d'utilisation.

La génération d'une nouvelle clé par les membres actuels du réseau nous paraît difficile. Nous avons en effet montré qu'une clé RSA pouvait être générée de manière distribuée par quelques dizaines de pairs ; la génération par plusieurs centaines de pairs ou de groupes de fragmentation ne nous semble pas réalisable actuellement. Il serait en revanche possible de créer un nouveau comité de quelques dizaines de membres, éventuellement le même que lors de l'initialisation du réseau, et de générer une nouvelle paire de clés RSA au sein de ce comité : la distribution des fragments de cette nouvelle clé serait alors réalisée lorsque les anciens pairs rejoindraient ce nouveau réseau. Les données signées contenues dans l'ancien réseau pair-à-pair devraient également être migrées et signées avec la nouvelle clé, selon un processus qui reste à déterminer.

Dans le cadre de l'autorité de certification distribuée, nous pouvons cependant nuancer le besoin de renouveler la clé de réseau. La durée de vie d'une clé est en effet calculée à partir des deux facteurs suivants : le temps nécessaire à un attaquant pour calculer la clé privée et la probabilité d'exposition de la clé privée.

Le premier facteur – le temps nécessaire à un attaquant pour calculer la clé privée – peut être amélioré arbitrairement en choisissant une clé de plus grande taille. Le second facteur – la probabilité d'exposition de la clé privée – est le plus souvent le facteur limitant car, quelle que soit la taille de la clé, un attaquant a toujours une probabilité non nulle d'accéder au fichier contenant la clé, stocké sur un serveur. Dans notre contexte d'autorité de certification distribuée, aucun pair ne connaît la clé secrète du réseau et un attaquant ne peut donc pas l'obtenir par intrusion sur un serveur : la durée de vie de la clé peut être améliorée en choisissant une clé de plus grande taille.

Délégation de signature

Dans un réseau pair-à-pair, la majorité des pairs honnêtes ne sont pas connectés de manière permanente. Dans un système de partage de fichiers, les utilisateurs se connectent le temps de télécharger le fichier souhaité ; dans un système de voix sur IP, les utilisateurs ne sont connectés que lorsqu'ils désirent être joignables. À l'opposé, les attaquants, qui souhaitent représenter une fraction du réseau la plus élevée possible, sont toujours connectés. Ainsi, si le système proposé résiste à 20% de pairs connectés attaquants, ce nombre de pairs attaquants correspond à moins de 20% du nombre total d'utilisateurs du système. Il est donc souhaitable que les pairs honnêtes soient également représentés de manière permanente dans le réseau.

Pour permettre cette représentation permanente de tous les pairs, les pairs honnêtes pourraient, lors de leur déconnexion, déléguer leur rôle de signature à une personne de confiance : ainsi, le pair remplaçant devrait prendre des décisions similaires au pair qu'il représente. Il serait possible de réutiliser les liens de confiance établis pour SybilGuard.

Il y aurait en définitive un pourcentage plus faible de pairs à convaincre pour obtenir un certificat et l'impact sur la sécurité devrait être étudié plus précisément. Les pairs connectés subiraient également un surcoût de communication et de calcul qu'il faudrait évaluer.

Remplacement de SybilGuard

Dans ce mémoire, nous avons proposé une protection contre l'attaque sybile couplant SybilGuard à l'autorité de certification distribuée. SybilGuard est utilisé par chaque pair de l'autorité distribuée pour décider de sa coopération à l'admission d'un nouveau membre : SybilGuard est en fait utilisé comme un oracle local, permettant d'évaluer le caractère sybil d'un nouveau pair.

SybilGuard pourrait ainsi être remplacé par un autre oracle de décision, par exemple une proposition plus récente telle que SybilLimit [YGKX08] ou SybilInfer [DM09] qui, tout en testant le même type de ressources sociales, sont plus fiables. Le couplage proposé dans ce mémoire améliorant la fiabilité de SybilGuard, il serait intéressant d'évaluer s'il augmente également la fiabilité de SybilLimit ou de SybilInfer.

Simulations dynamiques

Les évaluations présentées dans ce mémoire sont, pour la plupart, réalisées à un instant précis de la vie du réseau. Or la sécurité globale du système dépend à la fois de l'honnêteté des pairs connectés et du comportement de connexion des pairs (durée de connexion, délai avant reconnexion). Il serait donc intéressant d'évaluer le système et ses performances dans un réseau représentant les motifs de connexion attendus, qui dépendent du type d'application.

Au delà des lois probabilistes régissant ces comportements, il serait intéressant de réutiliser des études telles que [SENB07b], qui analyse le comportement des pairs dans le système de partage de fichiers Kad. À notre connaissance, il n'existe pas d'étude de ce type pour des systèmes de fichiers distribués ou réseaux de voix sur IP pair-à-pair, notamment du fait de leur diffusion beaucoup plus modeste.

Finalisation de l'implémentation

L'implémentation réalisée a été principalement utilisée pour évaluer le coût de la signature distribuée et pour vérifier le comportement des deux applications proposées (couplage avec SybilGuard et service de nommage sécurisé pour de la voix sur IP). Un certain nombre de points restent donc à implémenter, notamment l'opération de rafraîchissement et les communications sécurisées :

- l'opération de rafraîchissement utilisant les arbres de fragmentation, telle que décrite dans ce mémoire, n'a pas encore été implémentée. Cette opération, nécessaire à la sécurité de l'autorité de certification distribuée, ne l'est cependant pas à son fonctionnement. Nous avons ainsi pu vérifier le comportement de notre implémentation même sans cette opération ;
- l'implémentation actuelle utilise des communications en clair au lieu de communications authentifiées et chiffrées. Lors de l'admission dans le réseau, chaque pair détecté comme honnête par SybilGuard obtient bien son certificat d'appartenance au réseau, signé par la clé de réseau ; en revanche, ce certificat n'est pas encore utilisé par la couche réseau d'Overlay Weaver (la bibliothèque pair-à-pair que nous utilisons) et le contrôle d'accès à l'*overlay* pair-à-pair n'est donc pas réellement réalisé. L'ajout de communications SSL à la couche réseau d'Overlay Weaver permettra de limiter les communications aux seuls pairs admis et certifiés.

Bibliographie

[ABC+02] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger Wattenhofer. FARSITE : Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th ACM Symposium on Operating System Design and Implementation (OSDI)*, pages 1–14, New York, 2002. ACM Press.

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In Moti Yung, editor, *Proceedings of the 22nd Annual International Cryptology Conference*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432, Santa Barbara, California, 2002. Springer.

[AEHF08] Bernhard Amann, Benedikt Elser, Yaser Houri, and Thomas Fuhrmann. IgorFs : A distributed P2P file system. In Klaus Wehrle, Wolfgang Kellerer, Sandeep K. Singhal, and Ralf Steinmetz, editors, *Proceedings of the 8th IEEE International Conference on Peer-to-Peer Computing (P2P)*, Aachen, Germany, 2008. IEEE Computer Society.

[AF06] David P. Anderson and Gilles Fedak. The computational and storage potential of volunteer computing. In *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 73–80. IEEE Computer Society, 2006.

[AH00] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, 5(10), October 2000.

[AKD07] Agapios Avramidis, Panayiotis Kotzanikolaou, and Christos Douligeris. Chord-PKI : Embedding a public key infrastructure into the chord overlay network. In Javier Lopez, Pierangela Samarati, and Josep L. Ferrer, editors, *Proceedings of the 4th European Public Key Infrastructure Workshop (EuroPKI)*, volume 4582 of *Lecture Notes in Computer Science*, pages 354–361, Palma de Mallorca, Spain, 2007. Springer.

[AS04] Baruch Awerbuch and Christian Scheideler. Group spreading : A protocol for provably secure distributed name service. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3142 of *Lec-*

ture Notes in Computer Science, pages 183–195, Turku, Finland, 2004. Springer.

[AS06] Baruch Awerbuch and Christian Scheideler. Towards a scalable and robust DHT. In Phillip B. Gibbons and Uzi Vishkin, editors, *Proceedings of the 18th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 318–327, Cambridge, Massachusetts, USA, 2006. ACM.

[Bau08] Ingmar Baumgart. P2PNS : A secure distributed name service for P2PSIP. In *Proceedings of the 6th IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 480–485, Hong Kong, 2008. IEEE Computer Society.

[BF97] Boneh and Franklin. Efficient generation of shared RSA keys. In Burton S. Kaliski, Jr., editor, *Proceedings of the 17th Annual International Cryptology Conference (CRYPTO)*, volume 1294 of *Lecture Notes in Computer Science*, pages 425–439, Santa Barbara, California, USA, 1997. Springer.

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. In Richard E. Merwin, Jacqueline T. Zanca, and Merlin. Smith, editors, *Proceedings of the AFIPS National Computer Conference*, volume 48 of *AFIPS Conference proceedings*, pages 313–317. AFIPS Press, 1979.

[BLJ05] David A. Bryan, Bruce B. Lowekamp, and Cullen Jennings. SOSIMPLE : A serverless, standards-based, P2P SIP communication system. In *Proceedings of the International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA)*, pages 42–49. IEEE Computer Society, 2005.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7) :422–426, jul 1970.

[BLP05] Shane Balfe, Amit D. Lakhani, and Kenneth G. Paterson. Trusted computing : Providing security for peer-to-peer networks. In Germano Caronni, Nathalie Weiler, Marcel Waldvogel, and Nahid Shahmehri, editors, *Proceedings of the 5th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 117–124, Konstanz, Germany, 2005. IEEE Computer Society.

[Bor06] Nikita Borisov. Computational puzzles as sybil defenses. In Alberto Montresor, Adam Wierzbicki, and Nahid Shahmehri, editors, *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 171–176, Cambridge, United Kingdom, 2006. IEEE Computer Society.

[Boy89] Colin Boyd. Digital multisignatures. In H. Baker and F.Piper, editors, *Cryptography and Coding*, pages 241–246. Oxford University Press, 1989.

[BPS05] Jean-Michel Busca, Fabio Picconi, and Pierre Sens. Pastis : A highly-scalable multi-user peer-to-peer file system. In José C. Cunha and Pedro D. Medeiros, editors, *Proceedings of 11th International Euro-Par Conference (Euro-Par)*, volume 3648 of *Lecture Notes in Computer Science*, pages 1173–1182, Lisbon, Portugal, 2005. Springer.

[BRK07] Bobby Bhattacharjee, Rodrigo Rodrigues, and Petr Kouznetsov. Secure lookup without (constrained) flooding. In *Proceedings of the 1st Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS)*, Lisbon, Portugal, 2007.

[BS06] Salman Baset and Henning Schulzrinne. An analysis of the skype peer-to-peer internel telephony protocol. In *Proceedings of the 25th IEEE Conference on Computer Communications (INFOCOM)*, Barcelona, Spain, 2006. IEEE Computer Society.

[CBH03] Srdjan Capkun, Levente Buttyán, and Jean-Pierre Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(1) :52–64, 2003.

[CCR05] Miguel Castro, Manuel Costa, and Antony I. T. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2005.

[CDG+02] Miguel Castro, Peter Druschel, Ayalvadi J. Ganesh, Antony I. T. Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proceedings of the 5th ACM Symposium on Operating System Design and Implementation (OSDI)*, Operating Systems Review, pages 299–314, New York, USA, 2002. ACM Press.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony I. T. Rowstron, and Atul Singh. Splitstream : high-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, volume 37, 5 of *Operating Systems Review*, pages 298–313, New York, USA, 2003. ACM Press.

[CF05] Alice Cheng and Eric Friedman. Sybilproof reputation mechanisms. In *Proceeding of the ACM SIGCOMM Workshop on Economics of Peer-to-Peer systems (P2PECON)*, pages 128–132, New York, NY, USA, 2005. ACM Press.

[CKS+06] Tyson Condie, Varun Kacholia, Sriram Sank, Joseph M. Hellerstein, and Petros Maniatis. Induced churn as shelter from routing-table poisoning. In *Proceedings of the 13th Annual Symposium on Network and Distributed System Security (NDSS)*, San Diego, California, USA, 2006. The Internet Society.

[CL09] Thierry Congos and François Lesueur. Experimenting with distributed generation of RSA keys. In *Proceedings of the 5th International Workshop on Security and Trust Management (STM)*, Electronic Notes in Theoretical Computer Science, Saint Malo, France, 2009. Elsevier.

[Cli00] Clip2. The gnutella protocol specification v0.4. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`, 2000.

[CMM02] Russ Cox, Athicha Muthitacharoen, and Robert Morris. Serving DNS using a peer-to-peer lookup service. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *First International Workshop on Peer-to-Peer Systems (IPTPS)*, volume

2429 of *Lecture Notes in Computer Science*, pages 155–165, Cambridge, Massachusets, USA, 2002. Springer.

[Coh03] Bram Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

[CRB+03] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like P2P systems scalable. In Anja Feldmann, Martina Zitterbart, Jon Crowcroft, and David Wetherall, editors, *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 407–418, Karlsruhe, Germany, 2003. ACM Press.

[CSWH00] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet : A distributed anonymous information storage and retrieval system. In Hannes Federrath, editor, *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66. Springer, 2000.

[Dav96] Don Davis. Compliance defects in public key cryptography. In *Proceedings of the 6th USENIX Security Symposium*, pages 171–178, San Jose, California, USA, 1996. Usenix.

[Des87] Yvo Desmedt. Society and group oriented cryptography : A new concept. In Carl Pomerance, editor, *Proceedings of Advances in Cryptology (CRYPTO)*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127. Springer-Verlag, 1987.

[Des97] Yvo Desmedt. Some recent research aspects of threshold cryptography. In Eiji Okamoto, George I. Davida, and Masahiro Mambo, editors, *Proceedings of the 1st International Workshop on Information Security (ISW)*, volume 1396 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 1997.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Giles Brassard, editor, *Proceedings of Advances in Cryptology (CRYPTO)*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, California, USA, 1989. International Association for Cryptologic Research, Springer-Verlag.

[DF92] Yvo Desmedt and Yair Frankel. Shared generation of authenticators and signatures. In Joan Feigenbaum, editor, *Proceedings of Advances in Cryptology (CRYPTO)*, volume 576 of *Lecture Notes in Computer Science*, pages 457–469. International Association for Cryptologic Research, Springer-Verlag, 1992.

[DFN+08] C.R. Davis, J. Fernandez, S. Neville, J.-M. Robert, and J. McHugh. Sybil attacks as a mitigation strategy against the storm botnet. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software (Malware)*, pages 32–40, Fairfax, Virginia, USA, 2008. IEEE Computer Society.

[DH06] Jochen Dinger and Hannes Hartenstein. Defending the sybil attack in P2P networks : Taxonomy, challenges, and a proposal for self-registration. In *Proceedings of the 1st International Conference*

on Availability, Reliability and Security (ARES), pages 756–763. IEEE Computer Society, 2006.

[DJ97] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, University of Wisconsin, 1997.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David R. Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In Greg Ganger, editor, *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, volume 35, 5, pages 202–215, Chateau Lake Louise, Banff, Canada, 2001. ACM Press.

[DLLKA05] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek, and Ross Anderson. Sybil-resistant DHT routing. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 305–318, Milan, Italy, 2005. Springer.

[DM09] George Danezis and Prateek Mittal. Sybilinfer : Detecting sybil nodes using social networks. In *Proceedings of the 16th Annual Symposium on Network and Distributed System Security*, San Diego, California, USA, 2009. The Internet Society.

[Dou02] John R. Douceur. The sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260, Cambridge, Massachusetts, USA, 2002. Springer.

[DR01] Peter Druschel and Antony I. T. Rowstron. PAST : A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS)*, pages 75–80, Schloss Elmau, Germany, 2001. IEEE Computer Society.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31(4) :469–472, 1985.

[ES00] C. Ellison and B. Schneier. Ten risks of PKI : What you're not being told about public-key infrastructure. *Computer Security Journal*, 16(1) :1–7, 2000.

[FD92] Y. Frankel and Y. Desmedt. Parallel reliable threshold multisignature. Technical Report TR-92-04-02, Department of EE & CS, University of Wisconsin-Milwaukee, Milwaukee, WI, USA, 1992.

[FGMY97a] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Optimal-resilience proactive public-key cryptosystems. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 384–393, Miami Beach, Florida, USA, 1997. IEEE Computer Society.

[FGMY97b] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive RSA. In Burton S. Kaliski Jr., editor, *Proceedings of the 17th Annual International Cryptology Conference*

(CRYPTO), volume 1294 of *Lecture Notes in Computer Science*, pages 440–454, Santa Barbara, California, USA, 1997. Springer.

[FGY96] Yair Frankel, Peter Gemmell, and Moti Yung. Witness-based cryptographic program checking and robust function sharing. In *Proceedings of The 28th Annual ACM Symposium On The Theory Of Computing (STOC)*, pages 499–508, Philadelphia, Pennsylvania, USA, 1996. ACM Press.

[FMY98] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 663–672, Dallas, Texas, USA, 1998. ACM Press.

[Fra89] Yair Frankel. A practical protocol for large group oriented networks. In J.-J. Quisquater and J. Vandewalle, editors, *Proceedings of Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, volume 434 of *Lecture Notes in Computer Science*, pages 56–61, Houthalen, Belgium, 1989. Springer-Verlag.

[FSY05] Amos Fiat, Jared Saia, and Maxwell Young. Making chord robust to byzantine attacks. In Gerth Stølting Brodal and Stefano Leonardi, editors, *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*, volume 3669 of *Lecture Notes in Computer Science*, pages 803–814. Springer, 2005.

[GHH+06] Rachid Guerraoui, Sidath B. Handurukande, Kévin Huguenin, Anne-Marie Kermarrec, Fabrice Le Fessant, and Etienne Riviere. Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. In Alberto Montresor, Adam Wierzbicki, and Nahid Shahmehri, editors, *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 12–22, Cambridge, United Kingdom, 2006. IEEE Computer Society.

[GJKR96a] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. In Neal Koblitz, editor, *Proceedings of the 16th Annual International Cryptology Conference (CRYPTO)*, volume 1109 of *Lecture Notes in Computer Science*, pages 157–172, Santa Barbara, California, USA, 1996. Springer.

[GJKR96b] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. In Ueli Maurer, editor, *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371, Saragossa, Spain, 1996. International Association for Cryptologic Research, Springer.

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, volume 1599 of *Lecture Notes*

in Computer Science, pages 295–310. International Association for Cryptologic Research, Springer, 1999.

[GSAA04] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot : Content-based publish/subscribe over P2P networks. In Hans-Arno Jacobsen, editor, *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware)*, volume 3231 of *Lecture Notes in Computer Science*, pages 254–273, Toronto, Canada, 2004. Springer.

[Har94] L. Harn. Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *Computers and Digital Techniques, IEEE Proceedings*, 141(5) :307–313, 1994.

[HB06] Cyrus Harvesf and Douglas M. Blough. The effect of replica placement on routing robustness in distributed hash tables. In Alberto Montresor, Adam Wierzbicki, and Nahid Shahmehri, editors, *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 57–66, Cambridge, United Kingdom, 2006. IEEE Computer Society.

[HBMS04] Oliver Heckmann, Axel Bock, Andreas Mauthe, and Ralf Steinmetz. The eDonkey file-sharing network. In Peter Dadam and Manfred Reichert, editors, *Proceedings of the Workshop on Algorithms and Protocols for Efficient Peer-to-Peer Applications (Informatik)*, volume 51 of *LNI*, pages 224–228. GI, 2004.

[HJ03] Anthony Harrington and Christian Jensen. Cryptographic access control in a distributed file system. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 158–168, Como, Italy, 2003. ACM Press.

[HS05] David Hausheer and Burkhard Stiller. Peermint : Decentralized and secure accounting for peer-to-peer applications. In Raouf Boutaba, Kevin C. Almeroth, Ramón Puigjaner, Sherman X. Shen, and James P. Black, editors, *Proceedings of the 4th International IFIP-TC6 Networking Conference*, volume 3462 of *Lecture Notes in Computer Science*, pages 40–52, Waterloo, Canada, 2005. Springer.

[JMJV] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim simulator. `http ://peersim.sf.net`.

[JSS04] William K. Josephson, Emin Gün Sirer, and Fred B. Schneider. Peer-to-peer authentication with a distributed single sign-on service. In Geoffrey M. Voelker and Scott Shenker, editors, *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 3 of *Lecture Notes in Computer Science*, pages 250–258. Springer, 2004.

[JSY04] Stanislaw Jarecki, Nitesh Saxena, and Jeong Hyun Yi. An attack on the proactive RSA signature scheme in the URSA ad hoc network access control protocol. In Sanjeev Setia and Vipin Swarup, editors, *Proceedings of the 2nd ACM Workshop on Security of ad hoc and Sensor Networks (SASN)*, pages 1–9, Washington, DC, USA, 2004. ACM Press.

[Kle00] Jon Kleinberg. The small-world phenomenon : an algorithmic perspective. In *Proceedings of the 32nd annual ACM Symposium on Theory of Computing (STOC)*, pages 163–170. ACM Press, 2000.

[KMT03] Yongdae Kim, Daniele Mazzochi, and Gene Tsudik. Admission control in peer groups. In *Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA)*, pages 131–139. IEEE Computer Society, 2003.

[KZL+01] Jiejun Kong, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang. Providing robust and ubiquitous security support for mobile ad hoc networks. In *Proceedings of the 9th IEEE International Conference on Network Protocols (ICNP)*, pages 251–260. IEEE Computer Society, 2001.

[LKR04] Jian Liang, Rakesh Kumar, and Keith W. Ross. Understanding KaZaA. `http://cis.poly.edu/~ross/papers/UnderstandingKaZaA.pdf`, 2004.

[LKZ+04] Haiyun Luo, Jiejun Kong, Petros Zerfos, Songwu Lu, and Lixia Zhang. URSA : ubiquitous and robust access control for mobile ad hoc networks. *IEEE/ACM Transactions on Networking*, 12(6) :1049–1063, 2004.

[LMV07a] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong. Contrôle d'accès distribué à un réseau pair-à-pair. In *Proceedings of SAR-SSI*, Annecy, France, 2007.

[LMV07b] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong. Gestion distribuée d'identités résistante à la sybil attack pour un réseau pair-à-pair. In *Proceedings of SAR-SSI*, Annecy, France, 2007.

[LMV08a] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong. Detecting and excluding misbehaving nodes in a P2P network. In *Proceedings of the 8th International Conference on Innovative Internet Community Systems (I2CS)*, Schoelcher, Martinique, 2008.

[LMV08b] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong. A distributed certification system for structured P2P networks. In David Hausheer and Jürgen Schönwälder, editors, *Proceedings of the 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, volume 5127 of *Lecture Notes in Computer Science*, pages 40–52, Bremen, Germany, 2008. Springer.

[LMV08c] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong. A sybil-resistant admission control coupling SybilGuard with distributed certification. In *Proceedings of the 4th International Workshop on Collaborative Peer-to-Peer Systems (COPS)*, Rome, Italy, 2008. IEEE Computer Society.

[LMV08d] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong. A sybilproof distributed identity management for P2P networks. In *Proceedings of the 13th IEEE Symposium on Computers and Communications (ISCC)*, Marrakech, Morocco, 2008. IEEE Computer Society.

[LMV09] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong. An efficient distributed PKI for structured P2P networks. In *Proceedings of the 9th International Conference on Peer-to-Peer Computing (P2P)*, Seattle, Washington, USA, 2009. IEEE Computer Society.

[LSM06] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A survey of solutions to the sybil attack. Technical Report 2006-052, University of Massachusetts Amherst, 2006.

[LSW06] Thomas Locher, Stefan Schmid, and Roger Wattenhofer. eQuus : A provably robust and locality-aware peer-to-peer system. In Alberto Montresor, Adam Wierzbicki, and Nahid Shahmehri, editors, *Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 3–11, Cambridge, United Kingdom, 2006. IEEE Computer Society.

[MBKM06] Ruggero Morselli, Bobby Bhattacharjee, Jonathan Katz, and Michael Marsh. Keychains : A decentralized public-key infrastructure. Technical report, University of Maryland, 2006.

[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology (CRYPTO)*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Santa Barbara, California, USA, 1987. International Association for Cryptologic Research, Springer-Verlag.

[MGGM04] Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina. DHT routing using social links. In Geoffrey M. Voelker and Scott Shenker, editors, *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 3279 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2004.

[Mil67] S. Milgram. The small world problem. *Psychology Today*, 1(1) :60–67, 1967.

[MKKW00] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmet Witchel. Separating key management from file system security. *ACM SIGOPS Operating Systems Review*, 34(2) :19–20, 2000.

[MM02] Petar Maymounkov and David Mazières. Kademlia : A peer-to-peer information system based on the XOR metric. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65, Cambridge, Massachusetts, USA, 2002. Springer.

[MWB99] Michael Malkin, Thomas D. Wu, and Dan Boneh. Experimenting with shared generation of RSA keys. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, pages 43–56, San Diego, California, USA, 1999. IEEE Computer Society.

[NNK+05] Liben D. Nowell, J. Novak, R. Kumar, P. Raghavan, and A. Tomkins. Geographic routing in social networks. *Proceedings of the National Academy of Sciences*, 102(33) :11623–11628, 2005.

[Ped91] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In Donald W. Davies, editor, *Pro-*

ceedings of the International Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT), volume 547 of *Lecture Notes in Computer Science*, pages 522–526, Brighton, UK, 1991. Springer-Verlag.

[Rab98] Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *Proceedings of the 18th Annual International Cryptology Conference (CRYPTO)*, volume 1462 of *Lecture Notes in Computer Science*, pages 89–104, Santa Barbara, California, USA, 1998. Springer.

[RD01] Antony I. T. Rowstron and Peter Druschel. Pastry : scalable, decentralized object location and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, volume 2218 of *Lecture Notes in Computer Science*, pages 329–350, Heidelberg, Germany, 2001. Springer.

[REMP07] Hosam Rowaihy, William Enck, Patrick McDaniel, and Tom La Porta. Limiting sybil attacks in structured P2P networks. In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*, pages 2596–2600, Anchorage, AK, 2007. IEEE Computer Society.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In Roch Guerin, editor, *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, volume 31, 4 of *Computer Communication Review*, pages 161–172, San Diego, California, USA, 2001. ACM Press.

[RLS02] Rodrigo Rodrigues, Barbara Liskov, and Liuba Shrira. The design of a robust peer-to-peer system. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop (EW10)*, pages 117–124, Saint Emilion, France, 2002. ACM Press.

[RS98] Ronald L. Rivest and Robert D. Silverman. Are 'strong' primes need for RSA ? Technical report, RSA Data Security, Inc., pub-RSA :adr, dec 1998.

[RS07] Ravi Rao and Sandeep K. Singhal. P2P-IM : A P2P presence system for the internet. In Manfred Hauswirth, Adam Wierzbicki, Klaus Wehrle, Alberto Montresor, and Nahid Shahmehri, editors, *Proceedings of the 7th IEEE International Conference on Peer-to-Peer Computing (P2P)*, pages 233–234, Galway, Ireland, 2007. IEEE Computer Society.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signature and public-key cryptosystems. *Communication of the ACM*, 21(2) :120–126, 1978.

[RSA99] RSA Laboratories. *PKCS #5 v2.0 : Password-Based Cryptography Standard*. RSA Data Security, Inc., 1999.

[RSA02] RSA Laboratories. *PKCS #1 v2.1 : RSA Cryptography Standard*. RSA Data Security, Inc., 2002.

[Sch05] Christian Scheideler. How to spread adversarial nodes ? : rotate ! In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 704–713, Baltimore, MD, USA, 2005. ACM Press.

[Sch07] David Andrew Schultz. Mobile proactive secret sharing. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2007.

[SCS90] Service Central de la Sécurité des Systèmes d'information SCSSI. Glossaire de la sécurité des systèmes d'information. Technical Report CNTI/CN 27, N23, SCSSI, octobre 1990.

[SDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely (extended summary). In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 522–533, Montréal, Québec, Canada, 1994. ACM Press.

[SENB07a] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. Exploiting KAD : possible uses and misuses. *ACM SIGCOMM Computer Communication Review*, 37(5) :65–70, 2007.

[SENB07b] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A global view of KAD. In Constantine Dovrolis and Matthew Roughan, editors, *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 117–122, San Diego, California, USA, 2007. ACM Press.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11) :612–613, 1979.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Proceedings of Advances in Cryptology (EUROCRYPT)*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220, Brugge, Belgium, 2000. Springer.

[SLL08] David Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing (brief announcement). In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, page 458, Toronto, Canada, 2008. ACM Press.

[SM02] Emil Sit and Robert Morris. Security considerations for peer-to-peer distributed hash tables. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *Lecture Notes in Computer Science*, pages 261–269, Cambridge, Massachusetts, USA, 2002. Springer.

[SMK+01] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. In Roch Guerin, editor, *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, volume 31, 4 of *Computer Communication Review*, pages 149–160, San Diego, California, USA, 2001. ACM Press.

[SNDW06] Atul Singh, Tsuen-Wan Ngan, Peter Druschel, and Dan S. Wallach. Eclipse attacks on overlay networks : Threats and defenses. In *Proceedings of the 25th IEEE Conference on Computer Communications (INFOCOM)*, pages 1–12, Barcelona, Spain, 2006. IEEE Computer Society.

[STS08] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay weaver : An overlay construction toolkit. *Computer Communications*, 31(2) :402–412, 2008. `http ://overlayweaver.sourceforge.net`.

[STY03a] Nitesh Saxena, Gene Tsudik, and Jeong H. Yi. Admission control in peer-to-peer : Design and performance evaluation. In *Proceedings of the 1st ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, pages 104–113, Fairfax, Virginia, USA, 2003. ACM Press.

[STY03b] Nitesh Saxena, Gene Tsudik, and Jeong H. Yi. Experimenting with admission control in P2P. In *Proceedings of the International Workshop on Advanced Developments in System and Software Security (WADIS)*, Taipei, Taiwan, 2003.

[TXM03] Chunqiang Tang, Zhichen Xu, and Mallik Mahalingam. psearch : information retrieval in structured overlays. *ACM SIGCOMM Computer Communication Review*, 33(1) :89–94, 2003.

[Wöl05] Thomas Wölfl. Public-key-infrastructure based on a peer-to-peer network. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS)*. IEEE Computer Society, 2005.

[WS99] Rebecca N. Wright and Sara Spalding. Experimental performance of shared RSA modulus generation. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 983–984, Baltimore, Maryland, USA, 1999. Society for Industrial and Applied Mathematics.

[WWW02] Theodore M. Wong, Chenxi Wang, and Jeannette M. Wing. Verifiable secret redistribution for archive system. In *Proceedings of the 1st International IEEE Security in Storage Workshop (SISW)*, pages 94–106. IEEE Computer Society, 2002.

[YGKX08] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. SybilLimit : A near-optimal social network defense against sybil attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 3–17. IEEE Computer Society, 2008.

[YKGF06a] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. SybilGuard : defending against sybil attacks via social networks. In Luigi Rizzo, Tom Anderson, and Nick McKeown, editors, *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 267–278, Pisa, Italy, 2006. ACM Press.

[YKGF06b] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. SybilGuard : defending against sybil attacks via social

networks. Technical Report IRP-TR-06-01, Intel Research Pittsburgh, 2006. Also available at `http://www.comp.nus.edu.sg/~yuhf/sybilguard-tr.pdf`.

[Ylö96] Tatu Ylönen. SSH - secure login connections over the internet. In *Proceedings of the 6th USENIX Security Symposium (SSYM)*, pages 37–42, San Jose, California, 1996. USENIX Association.

[Zim95] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

[ZJC06] Phil Zimmermann, Alan Johnston, and Jon Callas. ZRTP : Extensions to RTP for Diffie-Hellman key agreement for SRTP. `http://zfoneproject.com/zrtp_ietf.html`, oct 2006.

[ZSvR02] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA : A secure distributed online certification authority. *ACM Transactions on Computer Systems*, 20(4) :329–368, 2002.

[ZSvR05] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. APSS : Proactive secret sharing in asynchronous systems. *ACM Transactions on Information and System Security*, 8(3) :259–286, 2005.

VU :
Le Directeur de Thèse

VU :
Le Responsable de l'École Doctorale

VU pour autorisation de soutenance
Rennes, le

Le Président de l'Université de Rennes 1

Monsieur Guy Cathelineau

VU après soutenance pour autorisation de publication :
Le Président du Jury,