Contrôle d'accès distribué à un réseau pair-à-pair

François Lesueur, Ludovic Mé et Valérie Viet Triem Tong
(prénom.nom@supelec.fr)

Supélec, Équipe SSIR (EA 4039), Avenue de la Boulaie - CS 47601, F-35576 Cesson-Sévigné cedex

Les systèmes pair-à-pair sont en plein essor. De nouvelles problématiques de sécurité apparaissent, le bon comportement de chaque nœud participant au réseau étant nécessaire pour le fonctionnement du réseau. Dans cet article, nous présentons un système de contrôle d'accès distribué à un réseau pair-à-pair, permettant de restreindre le réseau à un ensemble de nœuds ayant un comportement sain. Ce système repose sur des décisions prises par un pourcentage des nœuds présents et non par un nombre fixe de participants, ce qui est nécessaire dans le cadre des réseaux pair-à-pair dont la taille varie fortement au cours du temps.

Mots-clés: P2P, Autorisation d'accès, Certification distribuée

1 Introduction

Les systèmes Pair-à-Pair sont en plein essor depuis maintenant plusieurs années car ils permettent de concevoir des systèmes de très grande taille à forte disponibilité, tout cela à faible coût. En effet, d'une part les réseaux Pair-à-Pair sont constitués de nombreux nœuds, munis de mécanismes de réplication qui permettent une haute disponibilité, d'autre part ces nœuds sont des PC standards. Ces systèmes sont d'abord connus pour leur application au partage de fichiers (Gnutella [Cli00], Kazaa [LKR04], eDonkey [HBMS04]), mais ils permettent également de mettre en œuvre des systèmes de fichiers distribués (Past [DR01]), des applications de type *publish/subscribe* (Meghdoot [GSAA04]), de la multidiffusion (SplitStream [CDK+03]) ou encore de la voix sur IP (P2PSIP [BLJ05]).

Nous nous intéressons ici à la sécurité des réseaux pair-à-pair. Dans de tels réseaux, les propriétés de sécurité qui peuvent être recherchées (selon les besoins applicatifs) sont l'intégrité des ressources, la confidentialité des échanges entre pairs, la confidentialité des ressources globales du réseau (fichiers pour les systèmes de fichiers, abonnements pour le *publish/subscribe*...), l'authentification des pairs et l'anonymat des pairs. Il existe également un compromis entre l'authentification précise des pairs et l'anonymat total, qui pourrait être de savoir reconnaître qu'un membre a été admis dans un certain groupe sans pour autant l'identifier précisément.

Pour assurer l'intégrité des ressources dans un système de partage de fichiers, il suffit d'identifier les fichiers f par $h(f)$, h étant une fonction de hachage [Cli00, LKR04, HBMS04, CDG+02]. Le problème de l'intégrité des autres types de ressources reste un problème ouvert [CDG+02].

La confidentialité des échanges entre deux pairs, ou plus généralement entre n pairs, peut être assurée par la négociation entre les pairs concernés d'une clé de groupe [CSWH00, Was04]. À notre connaissance, il n'existe pas, à l'heure actuelle, de gestion de la confidentialité des ressources globales du réseau (fichiers, abonnement).

Une manière simple d'authentifier précisément des pairs peut être de lier un pair à sa clé publique. Néanmoins, cette approche n'est utilisable que dans de petits réseaux (jusqu'à une centaine d'utilisateurs). *Waste* [Was04] propose par exemple une approche reposant sur la connaissance des clés publiques de tous les membres du réseau, permettant de créer des réseaux pair-à-pair privés entre des membres se connaissant tous. Cette approche repose sur le fait que les membres se connaissent deux à deux et n'est donc pas applicable à de grands réseaux.

De plus, dans ces grands réseaux, nous pensons qu'il est plus pertinent de pouvoir appliquer un profil à un pair plutôt que de l'identifier précisément. Par exemple, dans un système de partage de fichiers, il est intéressant de pouvoir différencier les nœuds égoïstes des nœuds altruistes, pour ensuite favoriser les noeuds altruistes. Dans ce cas, l'identité précise des pairs n'est pas pertinente.

Les réseaux pair-à-pair sont des réseaux collaboratifs, dans lesquels chacun des participants est responsable d'une partie du bon fonctionnement global. Il apparaît donc que pour résoudre les problèmes de sécurité posés, il faut restreindre l'impact des nœuds malveillants sur le système. Suivant ce principe, notre contribution est ici un mécanisme permettant de contrôler l'accès au réseau pair-à-pair de manière distribuée. Ainsi, il est possible de restreindre l'accès au réseau à un ensemble de nœuds ayant un comportement sain. Ce mécanisme repose sur des décisions prises par un pourcentage constant des membres du réseau, quel que soit sa taille, et non sur un nombre fixe de participants comme proposé dans [KZL+01, STY03].

Le reste de cet article est organisé comme suit. Dans une première partie, nous décrivons les réseaux pair-à-pair structurés utilisés ici. Ensuite, dans une deuxième partie, nous présentons l'état de l'art. Ensuite, dans une troisième partie, nous expliquons notre proposition de contrôle d'accès distribué à un réseau pair-à-pair. Dans une quatrième partie, nous décrivons la certification distribuée d'un nœud. Dans une cinquième partie, nous présentons les opérations de maintenance mises en place. Enfin, nous étudions les résultats de simulations de ce système.

2 Les réseaux pair-à-pair structurés

Les réseaux Pair-à-Pair reposent sur une architecture entièrement décentralisée, où chaque participant a un rôle équivalent. L'objectif est d'éviter les points critiques et d'augmenter la disponibilité du système en agrégeant les ressources de tous les participants, tout en utilisant des algorithmes qui passent à l'échelle. Les nœuds sont supposés volatils et peu fiables : un nœud peut rejoindre ou quitter le réseau sans prévenir. Il existe deux grandes familles de réseaux pair-à-pair : les réseaux non-structurés (par exemple Gnutella [Cli00]) et les réseaux structurés (par exemple Pastry [RD01], CAN [RFH+01], Chord [SMK+01], Tapestry [ZKJ01]). Les réseaux non-structurés sont plutôt adaptés à des recherches complexes (portant sur le contenu du fichier) ou à des applications de diffusions, alors que les réseaux structurés sont limités à des recherches simples. En contrepartie, les réseaux structurés sont beaucoup plus performants en terme de coût de communication [CCR04]. Nous avons choisi dans cet article de présenter des mécanismes pour les réseaux pair-à-pair structurés. Néanmoins, nos mécanismes pourraient également s'appliquer aux réseaux non-structurés.

Les réseaux structurés reposent sur des algorithmes de routage *proactifs*, c'est-à-dire que pour toute requête il existe une route déjà définie dans les tables de routage.

Ces réseaux implémentent une *DHT (Distributed Hash Table)*. La DHT représente un espace virtuel, l'espace des clés $\mathcal{K}eyIds$. Chaque nœud (PC) est repéré par un identifiant unique $nodeId \in \mathcal{K}eyIds$; de la même façon, chaque *ressource* (fichier, abonnement, ...) est identifiée de manière unique par un identifiant de clé key, $key \in \mathcal{K}eyIds$ (dans le cas d'un fichier, la clé est souvent l'empreinte SHA1 de ce fichier). Les nœuds et les ressources partagent donc le même espace d'identifiants, espace fini mais supposé suffisamment grand (couramment de 2^{160} éléments en raison de la taille des empreintes SHA1). Pour accéder à une ressource, un nœud doit connaître la clé de cette ressource. Ce nœud utilise alors la DHT, qui implémente de façon distribuée une fonction liant la clé à la ressource.

L'espace $\mathcal{K}eyIds$, contenant à la fois les identifiants de nœuds ($nodeId$) et les identifiants de ressource (key), forme un réseau logique appelé *overlay*. Il est muni d'une mesure sur les identifiants permettant de calculer des distances dans l'*overlay*. Chaque nœud (A) est responsable de la zone de cet espace autour de lui (Z_A). Les nœuds souhaitant insérer des données dans la zone Z_A doivent envoyer ces données à A et les requêtes sur Z_A sont servies par A. Sachant que les nœuds sont volatils et peu fiables, chaque ressource est dupliquée sur n autres nœuds *répliques* (fonction de réplication gérée par l'*overlay*), qui seront alors capable de remplacer A si il quitte l'*overlay*, en prenant le contrôle de Z_A.

La figure 1 donne une représentation rectangulaire et une représentation circulaire d'un *overlay*. Sur ces deux représentations, les nœuds sont responsables des ressources dont ils sont les plus proches. Chaque nœud possède une zone dont il est responsable et connaît les limites de cette zone.

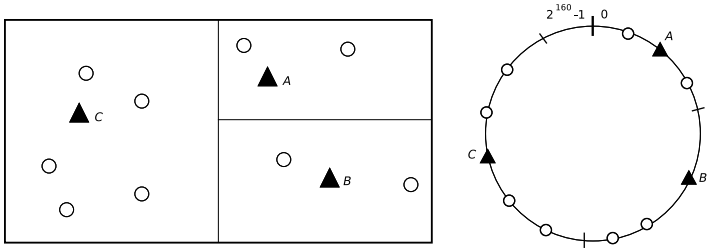


FIG. 1: Espace des identifiants d'un *overlay* structuré. À gauche, une représentation rectangulaire (CAN). À droite, une représentation sous forme d'anneau (Pastry). ▲ représente un nœud (PC) et ○ une ressource (fichier)

Le routage s'effectue de proche en proche. Chaque nœud dispose d'une table de routage, contenant ses voisins ainsi que d'autres nœuds répartis dans l'espace virtuel. Cette table de routage est mise à jour en arrière-plan, en fonction des modifications du réseau. Pour accéder à une ressource R, le nœud demandeur A envoie sa requête au nœud B contenu dans sa table ayant l'identifiant le plus *proche* de R. Si B ne possède pas R, il redirige la requête de la même façon. Par convergence de l'algorithme de routage, le message arrive au nœud X qui est responsable de la zone contenant R, en un nombre de sauts de l'ordre de $log(\mathcal{N})$, $\mathcal{N}$ étant la taille du réseau.

Les réseaux Pair-à-Pair sont des réseaux dynamiques, des nœuds pouvant être insérés ou retirés durant leur fonctionnement. Pour s'insérer, un nœud a besoin d'un *nodeId*, d'être connu des autres nœuds et de posséder une table de routage minimale.

Généralement, le *nodeId* est tiré aléatoirement par le nœud (avec une résolution de conflit éventuelle). Cela permet de générer une distribution aléatoire des nœuds dans l'espace de la DHT, sans nécessiter de point central.

Ensuite, le nœud A souhaitant s'insérer a besoin de connaître un nœud B appartenant déjà au réseau. B sert de point d'entrée à A, lui permettant de s'insérer au bon endroit dans la DHT et de se faire connaître de ses voisins. À ce point, A est accessible car ses voisins peuvent lui faire suivre les messages qui lui sont destinés.

Il ne reste plus à A qu'à se construire une table de routage minimale. Pour cela, A se base sur les tables de routage des nœuds qu'il connaît (B et les voisins de A). Il complète ensuite sa table en arrière plan, en utilisant l'algorithme de maintenance de table de routage. A est ainsi au moins connu de ses voisins et peut router des messages vers tout le réseau.

Lorsqu'un nœud A s'aperçoit qu'un de ses voisins B' est défaillant, il devient responsable des ressources de B'. De même, lorsqu'un nœud A s'aperçoit qu'un nœud B' appartenant à sa table de routage n'est plus dans le réseau, il supprime l'entrée de B' dans sa table et utilise une autre route possible.

3 État de l'art du contrôle d'accès aux réseaux pair-à-pair

Dans une première partie, nous présentons des travaux dans le domaine du contrôle d'accès à des réseaux pair-à-pair. Ensuite, dans une deuxième partie, nous décrivons la cryptographie à seuil, permettant de matérialiser cryptographiquement des accords entre plusieurs entités. Cette cryptographie sert de base à notre proposition, qui sera détaillée dans les sections suivantes.

3.1 Authentification de l'appartenance à un groupe

La première solution proposée pour permettre l'authentification des membres dans un réseau de pairs est l'établissement d'une clé de groupe. L'authentification d'un membre du groupe se fait alors en vérifiant que ce membre a bien la connaissance de la clé du groupe. Par exemple, dans [LLY06], Lee *et al.* présentent un protocole d'accord collaboratif pour les groupes pair-à-pair basé sur l'échange de clés Diffie-Hellman. Chaque membre du groupe est capable de reconstruire dynamiquement la clé du groupe et la clé est reconstruite à chaque départ ou arrivée d'un membre.

Une synthèse sur les différents travaux menés sur les protocoles d'accord de clés de groupes est présentée dans [AKNRT04]. De façon générale, l'ensemble de ces travaux permet à des nouveaux membres de

rejoindre le groupe et d'acquérir la clé du groupe. Lorsque des membres quittent le groupe, la clé est remise à jour. Le problème est qu'il n'y pas de contrôle d'accès : un nouveau membre peut rejoindre le groupe dès qu'un membre accepte de le parrainer. Les protocoles d'accord sur des clés de groupe n'ont de sens que si l'accès d'un nouveau membre à un groupe est contrôlé par le groupe ou au moins une partie du groupe.

Le contrôle d'accès distribué dans un réseau pair-à-pair est un problème très peu étudié. Kim *et al.*, les premiers, posent les bases de ce problème dans [KMT03]. Dans ce qui suit, nous reprenons l'article [KMT03] pour définir le cadre général du contrôle d'accès dans les réseaux pair-à-pair.

La première étape du protocole d'accès passe par la définition d'une politique d'accès au groupe : par exemple, la politique peut définir qu'un membre est accepté dans le groupe si il est capable de présenter un certificat signé par la clé privée du groupe. Le problème du contrôle d'accès est alors de définir comment un pair souhaitant rejoindre le groupe peut obtenir son certificat. Kim *et al.* proposent dans [KMT03] le schéma général suivant :

1. le nouveau membre M_{new} envoie une requête à des membres du groupe, cette requête est signée par M_{new} et contient un certificat de M_{new} signé par une autorité éventuellement extérieure au groupe ;
2. les membres ayant reçu la requête et acceptant l'entrée de M_{new} dans le groupe renvoient leur vote signé à M_{new}. Chaque vote doit être accompagné d'une preuve d'appartenance au groupe ;
3. M_{new} collecte les votes et les renvoie à l'autorité désignée pour le groupe ;
4. l'autorité désignée vérifie les votes et si ceux-ci sont suffisants, renvoie à M_{new} son certificat signé par la clé privée du groupe.

L'autorité désignée pour le groupe peut être soit un membre particulier du réseau (le fondateur par exemple), soit l'ensemble des membres du réseau. Nous nous intéressons ici au cas où l'autorité pour le groupe est l'ensemble des pairs. Nous distinguons deux cas : celui où le nouveau membre peut rejoindre le réseau si un nombre fixe de membres l'acceptent (*seuil fixe*), et celui où le nouveau membre doit être accepté par un pourcentage des membres du réseau (*seuil dynamique*).

Kong *et al.* proposent dans [KZL+01] un mécanisme de signature à seuil fixe totalement distribué basé sur le partage de secret de Shamir [Sha79]. Lors de la phase d'initialisation du réseau, les membres fondateurs choisissent la clé privée $< d, n >$ du groupe ainsi qu'un polynôme aléatoire $f(x)$ (privé) de degré $k-1$ tel que $f(0) = d$. Ensuite, chaque membre M_i possède un identifiant unique Id_i et dispose d'un secret partagé $P_{Id_i} = f(Id_i) \ mod \ n$. La reconstruction de d et donc la signature du groupe, ne se peut se faire qu'avec une coalition de k membres : en connaissant $f(Id_i)$ pour k valeurs différentes de i, f se reconstruit par interpolation de Lagrange. Pour qu'un membre rejoigne le groupe, il faut alors qu'il obtienne k signatures partielles de son certificat.

Le problème de cette approche est que le nombre de votes nécessaire à l'admission d'un nouveau membres doit être fixé à l'avance et ne peut pas évoluer. Le choix de k pose donc problème si le nombre de membres du réseau augmente fortement ou si, au contraire, il devient inférieur à k.

Dans [STY03], Saxena *et al.* proposent de faire varier ce seuil en utilisant [FGMY97]. Afin de savoir quand changer les seuils, un serveur est responsable de maintenir en temps réel le nombre de participants au réseau pair-à-pair. Au delà des problèmes de passage à l'échelle et de malveillance dans les algorithmes de changement de seuil, la présence d'un serveur va à l'encontre des principes du pair-à-pair et nuit à la disponibilité du réseau.

3.2 Cryptographie à seuil

La cryptographie à seuil, basée sur le partage de secret de Shamir [Sha79], permet de chiffrer des messages par la collaboration de k entités parmi n. Elle peut être utilisée afin de matérialiser cryptographiquement l'accord de k participants. La cryptographie à seuil consiste à séparer une clé privée en plusieurs fragments, puis à répartir ces fragments sur différentes entités. Nous nous basons ici sur [Des97, Rab98] pour présenter la cryptographie à seuil.

3.2.1 Fonctionnement

La cryptographie à seuil, également appelée cryptographie k-parmi-n, permet de chiffrer un message à l'aide de k fragments quelconques parmi n fragments d'une même clé. k et n sont des constantes décidées

à l'initialisation du système. La cryptographie à seuil permet de matérialiser l'accord de k membres d'un groupe contenant n personnes. Si personne ne connaît la clé entière, cela permet de plus de mieux protéger la clé des personnes malveillantes. En effet, il faut k fragments pour chiffrer un message, mais $k-i$ fragments n'apportent *aucune* information sur la clé secrète. Un intrus devra donc corrompre k fragments de la clé pour corrompre la clé complète.

Pour réaliser le chiffrement d'une donnée, une personne doit contacter k personnes possédant un fragment. Chacune de ces personnes chiffre la donnée avec son fragment et envoie le résultat au demandeur. Le demandeur multiplie tous les résultats et obtient ainsi le chiffré de sa donnée, la fonction de chiffrement retenue devant être homomorphique. Seuls les chiffrés intermédiaires sont publiés et en aucun cas les fragments de la clé : ceci assure la confidentialité de chaque fragment.

3.2.2 Problèmes soulevés

Le premier problème soulevé est l'initialisation du système. En effet, la répartition initiale des n fragments demande soit un *leader*, soit un algorithme de génération distribuée. Dans le cas de l'utilisation d'un *leader*, ce *leader* connaît alors toute la clé secrète et doit distribuer les fragments de manière sécurisée. Ce *leader* est un point faible du système, puisque la sécurité de la clé fragmentée repose sur la confiance en ce *leader*. Une solution plus robuste est l'utilisation d'algorithmes de génération distribuée, présentés notamment dans [BF97].

Un second problème réside dans le choix des valeurs k et n. En effet, ces valeurs sont fixées lors de l'initialisation du système, ce qui gène l'utilisation de la cryptographie à seuil dans le contexte de réseaux dynamiques. Des algorithmes existent cependant pour faire varier ces valeurs, tels que présentés dans [FGMY97], mais ils requièrent la participation de tous les fragments et ne sont pas robustes face à des utilisateurs malveillants.

Troisièmement, un intrus peut, au fur et à mesure du temps, obtenir k fragments différents, en récupérant un fragment à la fois. Pour éviter ce cas de figure, les fragments de clé sont régulièrement *rafraîchis* [HJKY95, Rab98]. L'idée est de mélanger les fragments, pour rendre chaque ancien fragment obsolète et incompatible avec le nouveau découpage de la clé. Pour cela, chaque personne divise aléatoirement son fragment en n parts et envoie une part à chaque personne. Chaque personne faisant de même, tout le monde reçoit n parts d'autres fragments, les somme et obtient ainsi un nouveau fragment compatible avec les autres. Cette opération doit être robuste à la présence de personnes malveillantes.

Un dernier problème est la vérification de la validité des chiffrements intermédiaires. En effet, il est facile pour une personne de falsifier un chiffrement intermédiaire, les clés publiques correspondant aux fragments n'étant pas connues (et ne devant pas l'être, car elles permettraient de casser le système cryptographique). Des mécanismes de validation ont été proposés, notamment dans [GJKR96, Rab98] pour RSA. Ces mécanismes reposent sur la présence d'un *leader*, responsable dans ce cas de distribuer des éléments de vérification en plus des fragments de la clé privée. Dans le cas où la clé est générée de manière distribuée (donc sans *leader*), ces mécanismes ne sont pas applicables.

La cryptographie à seuil présentée, permettant de chiffrer des messages par la collaboration de k entités parmi n, n'est donc pas applicable telle quelle dans le contexte des réseaux pair-à-pair. Nous présentons donc dans ce papier de nouveaux algorithmes de cryptographie à seuil adaptés au contexte pair-à-pair. Grâce à ces nouveaux algorithmes, nous proposons un mécanisme de contrôle d'accès dans les réseaux pair-à-pair structurés ouverts dans lequel l'admission se fait avec l'accord d'un pourcentage fixe des pairs, tout cela de manière entièrement distribuée.

4 Contrôle d'accès distribué au réseau

Dans cette section, nous présentons notre proposition de contrôle d'accès distribué au réseau. Son principe est de certifier collaborativement l'appartenance d'un nœud au réseau.

Chaque nœud M membre du réseau pair-à-pair possède au départ un couple de clés publique/privée (P_M, S_M) éventuellement certifié par une autorité extérieure au réseau. Le réseau est déterminé par un couple de clés (P_R, S_R), P_R étant connue de tous et S_R étant fragmentée sur l'ensemble des nœuds actuellement connectés au réseau (aucun nœud ne possède S_R en entier). Si M veut rejoindre le réseau, il doit obtenir un

certificat $Cert_M$ constitué par l'empreinte $h(P_M)$ (h étant une fonction de hachage) signé par la clé privée du réseau S_R. Cette empreinte $h(P_M)$ sera utilisé comme identifiant du nœud M. Afin de simplifier les notations, nous utiliserons plutôt Id_M à la place de $h(P_M)$. Nous montrons que la certification n'est possible que par la collaboration de k nœuds appartenant déjà au réseau. Contrairement à [KZL+01], k n'est pas fixé à l'initialisation du système mais est un seuil dynamique, correspondant à chaque instant à un certain pourcentage du nombre de membres du réseau. Enfin, afin que M soit capable à son tour de participer à l'admission de nouveaux nœuds, M doit recevoir le matériel cryptographique adéquat.

Dans un premier temps, nous décrivons notre proposition de cryptographie à seuil *adaptative*, permettant des seuils dynamiques dans le contexte des réseaux pair-à-pair, là où [FGMY97] ne le permet pas. Ensuite, dans la section 5, nous décrivons la certification des nœuds. Enfin, dans la section 6, nous expliquons les opérations de maintenance de la clé privée de réseau partagée.

4.1 Cryptographie à seuil adaptative

La cryptographie à seuil [Des97, Rab98] a pour principe de faire réaliser des chiffrements par la collaboration de plusieurs entités, le nombre d'entités nécessaires (le seuil) pour réaliser un chiffrement devant être connu lors de l'initialisation du système. Dans cette partie, nous proposons une approche permettant de faire varier ces seuils automatiquement en fonction de la taille du réseau.

Problèmes posés dans le cadre des réseaux pair-à-pair

Les réseaux pair-à-pair sont par nature très dynamiques et de taille très variable au cours du temps. Dans le cadre de ces réseaux pair-à-pair, il est donc souhaitable d'adapter dynamiquement le nombre de nœuds nécessaires au chiffrement, en fonction de la taille courante du réseau. Cette adaptation dynamique permet de répartir le pouvoir au sein du réseau, quel que soit le nombre de nœuds.

Pour cela, nous ne pouvons pas nous baser sur les algorithmes de cryptographie à seuil permettant de faire varier les seuils, présentés dans [FGMY97]. En effet, ces algorithmes demandent la collaboration et la bienveillance de tous les membres du système. Or, dans un réseau pair-à-pair, le coût pour impliquer tous les nœuds de manière synchronisée à chaque changement de taille serait beaucoup trop élevé, le nombre de nœuds étant très grand et les communications asynchrones. De plus, les réseaux pair-à-pair utilisés ici étant des réseaux ouverts, le système proposé doit tolérer la présence de nœuds malveillants.

De manière générale, les algorithmes de cryptographie à seuil demandant une participation globale (rafraîchissement, changement des seuils) doivent être modifiés afin de ne pas requérir la participation de tous les nœuds. En effet, dans les réseaux pair-à-pair, le facteur le plus important est le passage à l'échelle : un réseau se comportant bien avec un faible nombre de nœuds doit toujours bien se comporter avec un très grand nombre de nœuds. Tous les algorithmes mis en œuvre doivent pour cela être de complexité constante ou logarithmique et non polynomiale.

Solution proposée

Nous proposons ici un mécanisme de contrôle d'accès à un réseau pair-à-pair, dans lequel le contrôle est fait par un pourcentage donné des membres présents dans le réseau. Nous proposons de prouver l'appartenance d'un membre par la possession d'un certificat contenant l'identifiant du membre (qui est en fait l'empreinte de sa clé publique) et signé par la clé secrète du réseau. La sécurité du réseau repose donc sur un couple de clés publique/privée (P_R, S_R) (RSA dans notre cas), la clé publique étant connue de tous alors que la clé privée est générée de manière distribuée [BF97] entre les premiers participants au réseau et est fragmentée en plusieurs morceaux. Chaque membre connaît un fragment de la clé privée S_R mais la clé privée entière n'est connue d'aucun membre. Lorsqu'un nouveau membre souhaite rejoindre le groupe, il lui faut obtenir son certificat. Pour cela, il doit obtenir la signature de sa clé publique par chacun des fragments de la clé du groupe. Le pourcentage de nœuds présents devant accorder l'accès à un nouveau membre est défini par la politique de sécurité du système.

Nous basons la fragmentation de la clé privée sur la propriété d'homomorphie de la fonction de chiffrement RSA : si $S_R = (e,n)$ est la clé privée du réseau et $e_1, \ldots, e_f$ sont tels que $e = e_1 + \ldots + e_f$, alors $m^e[n] = m^{e_1+\cdots+e_f}[n] = (m^{e_1}[n] \times \ldots \times m^{e_f}[n])[n]$. Si la politique de sécurité du réseau fixe que l'accès d'un nouveau membre doit être contrôlé par $k\%$ des nœuds présents et si nous notons $\mathcal{N}$ la taille courante du réseau, alors la clé doit être fragmentée en $\frac{k \times \mathcal{N}}{100}$ fragments. Chaque fragment e_i est ensuite répliqué sur plusieurs membres formant un groupe appelé *groupe de fragmentation*. Il est intéressant de noter que si t est le

nombre de fragments, alors la cryptographie à seuil utilisée est, au sens strict, une cryptographie t-parmi-t. Cependant, chaque fragment étant répliqué sur plusieurs membres de sorte que chaque membre possède un fragment, il faut bien joindre t membres parmi $\mathcal{N}$ pour réaliser un chiffrement.

Du fait de la nature des réseaux pair-à-pair, dans lesquels le nombres de nœuds présents est très variable, il ne nous paraît pas réaliste de souhaiter l'accord d'*exactement k%* des membres. De plus, pour assurer la disponibilité de l'ensemble de fragments, il est souhaitable que tous les fragments soient distribués de manière équitable. Nous choissisons donc de définir dans la politique de sécurité la taille minimale $\mathcal{N}_{min}$ et la taille maximale $\mathcal{N}_{max}$ des groupes de fragmentation. La taille minimale $\mathcal{N}_{min}$ d'un groupe de fragmentation assure la disponibilité de tous les fragments. Les tailles $\mathcal{N}_{min}$ et $\mathcal{N}_{max}$ sont les bornes inférieure et supérieure des valeurs autorisées pour k, puisque k varie entre $\frac{1}{\mathcal{N}_{max}}$ et $\frac{1}{\mathcal{N}_{min}}$. Lorsqu'un groupe atteint $\mathcal{N}_{max}$, il est scindé en deux groupes. Afin d'obtenir un système stable, il faut assurer que la nouvelle taille des groupes est supérieure à $\mathcal{N}_{min}$, sinon la scission d'un groupe contenant trop de membres crée deux groupes ne contenant pas assez de membres, qui vont immédiatement devoir fusionner à nouveau. Pour empêcher cela, il faut choisir $\mathcal{N}_{min}$ et $\mathcal{N}_{max}$ de sorte que $\mathcal{N}_{max} > 2 \times \mathcal{N}_{min}$. Dans notre approche, il n'est pas nécessaire de maintenir un compte global du nombre de nœuds du réseau pour faire varier le seuil de certification, seules des observations locales à chaque groupe de fragmentation sont nécessaires. Chaque nœud doit surveiller la présence des autres membres de son groupe de fragmentation, afin de détecter quand ce nombre franchit une borne $\mathcal{N}_{min}$ ou $\mathcal{N}_{max}$.

Par exemple, sur la figure 2, la politique de sécurité définit que chaque groupe de fragmentation doit contenir au minimum $\mathcal{N}_{min} = 3$ nœuds et au maximum $\mathcal{N}_{max} = 8$ nœuds, ce qui vérifie bien $\mathcal{N}_{max} > 2 \times \mathcal{N}_{min}$. La clé est divisée en 3 fragments et sur le cas présenté le réseau est constitué de 13 pairs. L'accès d'un nouveau membre est contrôlé par $\frac{3}{13}$ des membres, soit environ 20%.

Dans la section suivante, nous présentons notre algorithme de certification distribuée. Dans la section 6, nous présentons les algorithmes nécessaires à la maintenance de la clé du réseau pair-à-pair : rafraîchissement de la clé, obtention de la clé, fusion de deux groupes de fragmentation et scission d'un groupe de fragmentation.

5 Certification distribuée d'un nœud

Comme annoncé dans la section précédente, notre algorithme se base sur un couple de clés publique/privée RSA (P_R, S_R) définissant les membres d'un réseau pair-à-pair. La clé P_R est connue de tous, elle est aussi accessible aux utilisateurs extérieurs au réseau pair-à-pair. La clé S_R n'est connue de personne mais elle est divisée en plusieurs fragments, chaque fragment étant connu de plusieurs pairs qui forment un groupe de fragmentation. Lorsqu'un nouveau membre M_{new} demande à accéder aux ressources du réseau, il doit obtenir un certificat $Cert_{new}$ signé par la clé de réseau S_R. Dans notre cas, le certificat contient uniquement l'identifiant qu'aura M_{new} une fois accepté dans le réseau pair-à-pair. Pour l'obtenir, M_{new} fait une requête contenant une demande de certificat à un nœud A *quelconque* appartenant déjà au réseau. A est ensuite chargé de faire signer la demande de certificat par exactement un membre de chaque groupe de fragmentation. M_{new} est accepté si A parvient à faire signer la demande de certificat par tous les fragments composant S_R. Dans la suite, nous présentons la manière dont les fragments de clé sont identifiés afin que les membres soient capables de faire suivre des demandes de certificats à un membre de chacun des groupes de fragmentation. Ensuite, nous détaillons l'algorithme de certification distribuée.

Dans tout ce qui suit, nous dirons qu'un nœud est *bienveillant* s'il agit en suivant les protocoles présentés ou qu'il est *malveillant* s'il agit en dehors des spécifications des protocoles.

Identification des fragments de clé

Chaque fragment de clé est identifié de manière unique, chaque identifiant *nodeId* d'un nœud du réseau pair-à-pair correspondant à un unique fragment de la clé. Les fragments de clé s'organisent dans un arbre binaire. Chaque fragment est connu de tous les nœuds ayant un identifiant de préfixe l'identifiant du fragment, c'est-à-dire que le fragment $S_{R_i} =< e_i, n >$ est connu de tous les nœuds d'identifiant $i*$. Cette distribution est illustrée par la figure 2. Chaque nœud possède donc un seul fragment (figure 2), et la clé privée du réseau S_R n'est connue de personne. La seule façon de retrouver $S_R =< e, n >$ est de posséder tous les fragments

$S_{R_i} =< e_i, n >$. Pour obtenir un certificat *Cert* signé par la clé S_R, il faut combiner les chiffrements partiels de la demande de certification D de tous les fragments $S_{R_i} =< e_i, n >$:

$$Cert = D^e[n] = D^{e_1+\ldots+e_f}[n] = D^{e_1} \times \ldots \times D^{e_f}[n]$$

Reprenons l'exemple proposé sur la figure 2. Sur cette figure, il y a trois fragments S_{R_0}, $S_{R_1}0$ et $S_{R_1}1$. Tous les nœuds dont l'identifiant débute par un 0 connaissent le fragment S_{R_0}, ceux dont l'identifiant débute par 10 connaissent $S_{R_{10}}$, les autres disposent de $S_{R_{11}}$.

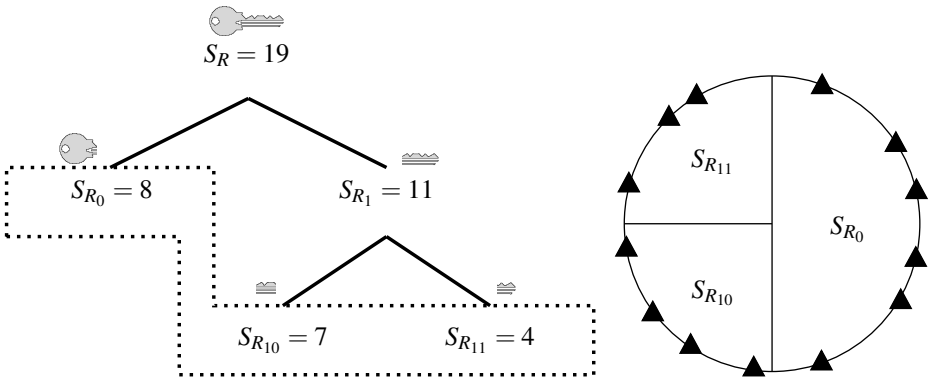


FIG. 2: À gauche, l'arbre des fragments ; à droite, la répartition des fragments dans le réseau, chaque nœud connaissant le fragment de sa zone uniquement. Les seuls fragments présents dans le réseau sont les fragments encadrés sur l'arbre : aucun nœud ne connaît S_R.

Algorithme de certification

L'algorithme de certification a pour but de faire approuver la demande d'accès d'un membre M_{new}. Pour cela, M_{new} contacte un nœud appelé *nœud initiateur* chargé d'obtenir la signature de la demande de certificat de M_{new}. Dans notre cas, le certificat d'un membre est simplement la signature de son identifiant pour le réseau par la clé secrète du réseau. La vérification de la signature se fait à l'aide de la clé publique du réseau qui est connue de tous.

Le mécanisme de certification est présenté récursivement et est détaillé par l'algorithme 1. Notre algorithme a pour entrée un identifiant de fragment de clé i et l'identifiant à certifier $Id_{M_{new}}$. Nous montrons par la suite que, dans une approche optimiste où tous les nœuds sont bienveillants, si les membres sollicités acceptent l'admission du nouveau membre, cet algorithme termine et renvoie la signature de l'identifiant par la clé S_R. Nous présentons ensuite une autre version de l'algorithme (algorithme 2) destinée à obtenir la certification en présence de nœuds malveillants.

Chaque nœud n appelé reçoit d'un autre nœud n' l'identifiant $Id_{M_{new}}$ à certifier ainsi que l'identifiant i correspondant à un fragment de clé $S_{R_i} =< e_i, n >$. Si n possède le fragment $S_{R_i} =< e_i, n >$, alors il renvoie $Id^{e_i}_{M_{new}}[n]$ à n' qui lui avait envoyé la requête. Sinon, il demande à deux nœuds quelconques dont les identifiants *nodeId* sont de la forme $i0*$ et $i1*$ d'effectuer le chiffrement de $Id_{M_{new}}$ avec les fragments de clés d'identifiants respectifs $i0*$ et $i1*$. Il multiplie ensuite les résultats obtenus et retransmet le résultat à n'. Notons que cet algorithme débute avec le nœud initiateur et l'identifiant de clé vide.

Le déroulement de cet algorithme est illustré figure 3. Nous reprenons la fragmentation de la figure 2, avec trois fragments de clés S_{R_0}, $S_{R_{11}}$ et $S_{R_{10}}$. Supposons qu'un nouveau membre M_{new} contacte le nœud A afin d'obtenir l'autorisation d'accès au réseau. A transmet à deux nœuds dont les identifiants commencent l'un par 0 et l'autre par 1, l'identifiant $Id_{M_{new}}$ à signer avec les fragments S_{R_0} et S_{R_1} (comme illustré figure 3). Le nœud dont l'identifiant commence par 0 est en mesure de calculer $Id^{e_0}_{M_{new}}[n]$ et de le renvoyer à A. Le second nœud contacté par A ne peut effectuer le chiffrement partiel demandé puisque S_{R_1} n'existe pas. Ce second nœud demande alors à deux autres nœuds d'identifiants commençant respectivement par 10 et 11 de faire les chiffrements à l'aide des fragments $S_{R_{11}}$ et $S_{R_{10}}$. Au retour de l'appel récursif, si les nœuds sont bienveillants et acceptent l'admission de M_{new} dans le groupe, A dispose du chiffrement de $Id_{M_{new}}$ par S_R.

Lorsqu'un nœud signe le certificat présenté de M_{new} avec son fragment de clé, cela signifie qu'il donne son accord pour l'insertion de B dans le réseau. Si le nœud ne souhaite pas accepter M_{new} dans le réseau, alors il lui suffit de refuser de certifier avec son fragment de clé.

Terminaison de l'algorithme de certification

Si les nœuds n'acceptent pas l'admission du nouveau membre alors ils ne répondent pas à la requête reçue et cet algorithme ne termine pas (sortie par un *timeout*). Par contre, si les membres acceptent l'admission du nouveau membre alors cet algorithme termine puisqu'il est appelé sur un argument (l'identifiant de la clé) de taille strictement croissante, bornée et atteignant sa borne. Rappelons que l'identifiant d'un fragment de clé est un préfixe des identifiants de nœuds $nodeId \in \mathcal{K}eyIds$ et que l'ensemble $\mathcal{K}eyIds$ est fini.

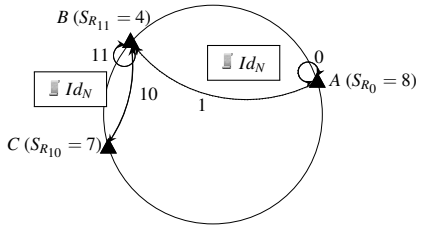
Algorithme 1 Algorithme de certification

Entrées: $Id_{M_{new}}$ (identifiant à certifier), i (identifiant de fragment)
Sorties: $\{Id_{M_{new}}\}_i$ (certificat partiel correspondant à S_{R_i})

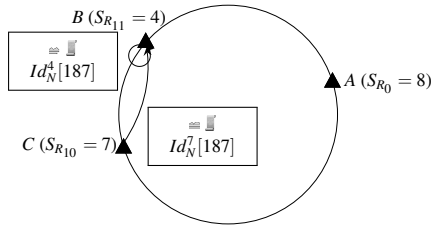
si S_{R_i} est le fragment possédé **alors**
  Retourner $\{Id_{M_{new}}\}_{S_{R_i}}$
sinon
  Retourner $cert(Id_N, i0) \times cert(Id_N, i1)$
fin si

Optimisation

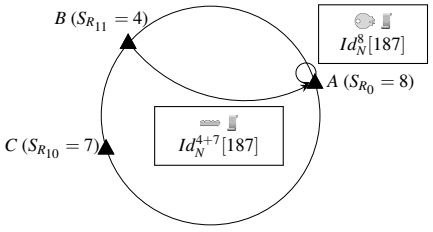
Dans l'algorithme présenté, chaque nœud n contacté fait suivre deux requêtes à deux autres nœuds n_0 et n_1 dont les identifiants on comme préfixe $i0$ et $i1$. Cependant l'identifiant du nœud n a lui-même pour préfixe $i0$ ou $i1$. Pour réduire le nombre de communications, n peut ne transmettre que la requête qui ne correspond pas à son propre préfixe. Dans ce cas, le nombre de nœuds contactés correspond exactement au nombres de fragments.



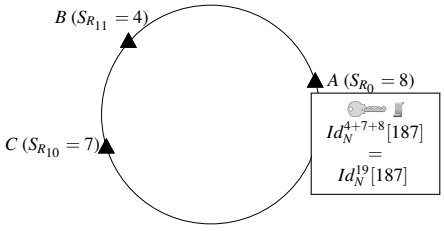
(a) A transmet l'identifiant Id_N à certifier à tous les nœuds nécessaires (un nœud par fragment)



(b) B et C réalisent leurs certificats intermédiaires, correspondant aux fragments en leur possession



(c) B multiplie les deux certificats intermédiaires et A réalise son certificat intermédiaire



(d) A combine les deux derniers certificats intermédiaires et obtient le certificat signé par la clé privée du réseau

FIG. 3: Illustration de la certification dans un réseau structuré, avec la clé privée RSA $(19, 187)$. Seuls les certificats intermédiaires circulent, les fragments de clé ne circulent pas.

Certification en présence de nœuds malveillants

Durant les opérations de cryptographie à seuil, chaque nœud impliqué a la capacité de corrompre le chiffrement global. En effet, la clé complémentaire à chaque fragment n'est pas connue et il n'est donc pas possible de vérifier les chiffrements intermédiaires. Pour pallier ce problème, la cryptographie à seuil propose des mécanismes de validation [GJKR96, Rab98], permettant de confirmer chaque chiffrement intermédiaire. Cependant, ces mécanismes ne sont pas utilisables ici. En effet, ces mécanismes demandent le calcul et l'envoi secret par un *leader* initial d'éléments de vérification ; ils ne sont donc pas utilisables dans le cas de génération distribuée de clés. Il existe donc deux menaces à cette certification : des certificats intermédiaires faussés et des multiplications biaisées.

Nous proposons de remplacer la vérification cryptographique de la cryptographie à seuil par une vérification auprès de plusieurs nœuds, en supposant que le nombre de nœuds malveillants est faible. Ceci est réaliste si ces nœuds malveillants sont régulièrement exclus du réseau. Il suffit alors d'interroger suffisamment de nœuds pour en déduire le chiffrement le plus vraisemblable, qui est le chiffrement intermédiaire *valide*.

Nous proposons de réaliser l'opération en deux temps, dans une approche optimiste. Dans un premier temps, le chiffrement est réalisé comme décrit précédemment, en supposant que tous les nœuds sont bienveillants, chaque nœud stockant les chiffrements intermédiaires utilisés localement. Puis, dans un second temps, si le chiffrement final n'est pas valide (vérification avec la clé publique de réseau), un algorithme plus robuste est appliqué. Durant cette étape, chaque nœud demandant un chiffrement le demande en fait à plusieurs nœuds distincts, en commençant par vérifier le bas de l'arbre de chiffrement (le bas de l'arbre est le moins coûteux à vérifier, puisque générant moins d'appels récursifs). Chaque chiffrement intermédiaire, qu'il soit au niveau d'une feuille ou d'un nœud de l'arbre, est ainsi vérifié. Si le chiffrement vérifié est différent du chiffrement retourné initialement, le nœud propage une interruption dans l'arbre ainsi que le nouvelle valeur pour rafraîchir le chiffrement global (en utilisant un mécanisme simple de multidiffusion arborescente). Cette interruption permet d'éviter des vérifications inutiles, au cas où la seule erreur ait été trouvée. Si le chiffrement global est toujours faux, la vérification reprend là ou elle en était. La progression de l'algorithme est ainsi assurée, les nœuds déjà vérifiés ne revérifiant pas de nouveau leur chiffrement. De plus, moyennant l'hypothèse qu'en interrogeant n nœuds sur un même chiffrement intermédiaire, le demandeur est sûr de la *validité* du chiffrement intermédiaire obtenu, la correction de l'algorithme est assurée. Cet algorithme est très proche de l'algorithme de certification (il suffit de faire plusieurs requêtes au lieu d'une et de faire passer un message d'interruption).

Algorithme 2 Algorithme de certification en présence de nœuds malveillants (présenté ici sans mécanisme d'interruption)

Entrées: Id_N (identifiant à certifier), i (identifiant de fragment)
Sorties: $\{Id_N\}_i$ (certificat partiel correspondant à S_{R_i})

si S_{R_i} est le fragment possédé **alors**
  Retourner $\{Id_N\}_{S_{R_i}}$
sinon
  $cert_{i0}[nbChecks], cert_{i1}[nbChecks]$
  pour $j = 0..nbChecks$ **faire**
    $cert_{i0}[j] = cert(Id_N, i0)$
  fin pour
  pour $k = 0..nbChecks$ **faire**
    $cert_{i1}[k] = cert(Id_N, i1)$
  fin pour
  Retourner $maxPresent(cert_{i0}) \times maxPresent(cert_{i1})$
fin si

Nous disposons donc de deux algorithmes légèrement différents. Un premier optimiste, considérant que tous les nœuds sollicités sont bienveillants et permettant de privilégier de bonnes performances ; un second sûr, garantissant le résultat obtenu aux dépends des performances.

6 Maintenance de la clé secrète fragmentée de réseau

Nous présentons d'abord le fonctionnement général de la maintenance de la clé fragmentée, puis nous décrivons précisément les quatre mécanismes mis en jeu.

Comme expliqué dans la section 4.1, pour définir le pourcentage des nœuds nécessaire pour réaliser un certificat, la politique de sécurité du système définit en fait $\mathcal{N}_{min}$ et $\mathcal{N}_{max}$. $\mathcal{N}_{min}$ correspond au nombre minimum de membres d'un groupe de fragmentation et $\mathcal{N}_{max}$ au nombre de membres maximum d'un tel groupe. Le pourcentage de membres nécessaire à un chiffrement est par conséquent compris entre $\frac{1}{\mathcal{N}_{min}}$ et $\frac{1}{\mathcal{N}_{max}}$. Lorsqu'un groupe atteint $\mathcal{N}_{max}$, il est scindé en deux groupes de taille N_{new}. Afin d'obtenir un système stable, il faut $\mathcal{N}_{new} > \mathcal{N}_{min}$, sinon, la scission d'un groupe contenant trop de membres crée deux groupes ne contenant pas assez de membres, qui vont immédiatement devoir fusionner à nouveau.

Nous distinguons quatre opérations de maintenance :

- l'*obtention* qui consiste pour un nouveau nœud à obtenir un fragment de clé du réseau afin de pouvoir participer aux opérations de certification globales ;
- la *fragmentation* qui consiste à créer deux groupes de fragmentation à partir d'un seul, quand les membres de ce groupe sont trop nombreux ;
- la *fusion* qui consiste à regrouper deux groupes de fragmentation lorsqu'ils sont trop petits ;
- le *rafraîchissement* qui consiste à modifier les fragments afin d'éviter qu'un adversaire puisse progressivement connaître toute la clé secrète du réseau.

La fusion et le rafraîchissement reposent en premier lieu sur un mécanisme commun de découverte d'un groupe possédant un autre fragment. Cette découverte ainsi que la fragmentation, la fusion et le rafraîchissement reposent ensuite sur des consensus [Kur02]. Les consensus sont des opérations coûteuses mais nécessaires, qui sont ici possibles grâce au faible nombre de nœuds impliqués (au maximum l'ensemble des membres de deux groupes de fragmentation). Nous présentons tout d'abord le principe de la découverte d'autres groupes avant de présenter les quatre opérations de maintenance.

6.1 Découverte d'un autre groupe

Soient $\mathcal{N}$ l'ensemble des nœuds du réseau et S_R la clé secrète du réseau. Dans tout ce qui suit, nous faisons l'hypothèse que dans chaque groupe de fragmentation, le nombre de nœuds malveillants est très inférieur au nombre de nœuds normaux. Les opérations de maintenance de la clé secrète fragmentée de réseau visent à garantir l'**invariant** suivant :

1. $\forall n \in \mathcal{N}$, n connaît localement la liste des membres bienveillants présents dans son groupe de fragmentation, dont éventuellement certains inactifs ;
2. $S_R = \sum_i S_{R_i}$, la clé S_R est la somme de tous les fragments possédés dans les groupes de fragmentation.

Afin de réaliser des opérations de fusion ou rafraîchissement entre deux groupes de fragmentation A et B distincts, il est nécessaire que chaque membre ait connaissance de tous les membres de son groupe et de tous les membres composants l'autre groupe. D'après l'invariant, chaque membre a déjà connaissance des membres composant son propre groupe. En effet, les membres d'un même groupe de fragmentation sont les nœuds actifs dont l'identifiant a comme préfixe l'identifiant du fragment de clé du groupe de fragmentation. Par contre, les membres n'ont pas connaissance des membres des autres groupes.

Lorsqu'un groupe de fragmentation A décide de réaliser une opération (fusion ou rafraîchissement) et qu'il a besoin de connaître les membres d'un autre groupe B, chaque nœud de A demande à un nœud de B de lui renvoyer la liste des membres du groupe B. Un consensus est alors déclenché dans B, afin de savoir si les membres acceptent ou non de réaliser l'opération. Si les membres de B acceptent, chaque nœud ayant reçu une requête renvoie la liste des membres du groupe B. Les nœuds de A ont ainsi chacun reçu la liste du groupe B. Les membres de A s'accordent ensuite sur la vision des membres de B en choisissant par consensus l'ensemble des identifiants définis comme actifs par la majorité des membres de B. Si les membres du groupe B ont accepté de réaliser l'opération, alors les membres de B ont eux aussi besoin de

connaître les membres du groupe A. En réponse à la liste des membres de B, les membres de A renvoient à leur tour la liste des membres de A.

À l'issue de ce protocole de découverte, chaque groupe A et B réalise séparément un consensus sur la liste des nœuds des deux groupes.

Propriété 1 *À l'issue du protocole de découverte d'un autre groupe, dans un réseau figé, chaque groupe a une vision complète de l'autre groupe.*

Démonstration : La démonstration se fait en utilisant un raisonnement par l'absurde : supposons que les membres du groupe B s'accordent à penser que le groupe A soit constitué des membres $\{a_0, \ldots, a_n\}$. Si il existe un membre a de A n'apparaissant pas dans $\{a_0, \ldots, a_n\}$ alors aucun nœud B n'a reçu de réponse d'un nœud de A contenant a. Soit les nœuds interrogés ne connaissent pas a, ce qui contredit l'invariant de connaissance des membres actifs ; soit ces nœuds sont malveillants, ce qui contredit l'hypothèse selon laquelle le nombre de nœuds malveillants est faible. La démonstration est la même pour le consensus au sein de A.

Nous pouvons remarquer ici que les nœuds malveillants n'ont pas d'influence sur les consensus finaux. En effet, lorsque les membres d'un groupe s'accordent par consensus sur la vision des membres de l'autre groupe, la décision se fait suivant la majorité des réponses reçues. De plus, par hypothèse, il y a plus de nœuds bienveillants que de nœuds malveillants dans un groupe de fragmentation.

6.2 Obtention d'un fragment de clé

Lorsqu'un nœud M s'insère dans le réseau (lors de sa première connexion ou suite à une déconnexion), il doit automatiquement obtenir le fragment de clé correspondant à son identifiant. Pour cela, il contacte un nombre l de nœuds actuellement responsables de ce fragment, l devant être suffisamment grand pour contenir au moins un nœud bienveillant.

Hypothèse 1 *l (nombre de nœuds contacté par un nouveau nœud) est choisi suffisamment grand pour assurer la présence d'un nœud bienveillant parmi les l.*

Chacun de ces nœuds, vérifie d'abord que le nœud M a le droit de recevoir ce fragment. Pour cela il faut vérifier deux choses. Tout d'abord que M peut présenter un certificat signé par la clé privée du groupe, la vérification est faite en utilisant la clé publique P_R. Ensuite il faut vérifier que l'identifiant contenu dans le certificat a bien comme préfixe l'identifiant du fragment de clé demandé. Une fois ces vérifications faites, les nœuds contactés par M renvoient à M le fragment de clé demandé et la liste des membres du groupe de fragmentation. Le nouveau nœud M fusionne alors les listes reçues, contacte chaque membre pour annoncer son arrivée et obtient ainsi sa propre liste des membres actifs. Le nouveau nœud est enfin en charge de vérifier que tous les fragments de clé reçus sont les mêmes.

Propriété 2 *À la fin de cet algorithme, le nouveau nœud connaît tous les membres de son groupe de fragmentation.*

Démonstration : Le nouveau nœud m contacte k nœuds, parmi lesquels il existe par hypothèse au moins un nœud bienveillant. Ce nœud bienveillant connaît tous les membres actifs de son groupe de fragmentation (première partie de l'invariant) et les transmet donc à m. m connaît ainsi, entre autres, tous les membres de son groupe de fragmentation. m élimine ensuite naturellement les membres inactifs (ou faux).

Propriété 3 *À la fin de cet algorithme, chaque nœud du groupe a inséré le nouveau nœud dans la liste des membres du groupe de fragmentation, si ce nouveau nœud est bienveillant.*

Démonstration : Par l'absurde. Supposons qu'il existe un nœud n qui n'ait pas inséré le nouveau nœud m dans sa liste. Alors m n'a pas contacté n pendant l'algorithme. Soit m a volontairement évité de contacter n, auquel cas m est malveillant et n'a donc pas besoin de figurer dans la liste de n ; soit m n'a pas reçu n dans l'une des k listes de membres reçus, ce qui contredit la propriété précédente.

La première partie de l'invariant est donc vérifiée. Le fragment reçu étant vérifié auprès de plusieurs sources, la seconde partie de l'invariant est également vérifiée.

Le choix d'une valeur de l (nombre de nœuds contactés) optimale peut être compliqué. Si l est trop petit, un nouveau nœud peut ne contacter que des nœuds malveillants, et ainsi ne pas connaître tous les membres de son groupe de fragmentation ou ne pas connaître le bon fragment de clé. À l'inverse, si l est trop grand, l'obtention d'un fragment entraîne un trafic inutile. Une possibilité est de reporter le trafic non réalisé à l'insertion, en réalisant régulièrement des requêtes *via* l'*overlay* pour trouver un autre membre et vérifier que ce membre est déjà connu.

6.3 Fragmentation d'un fragment

La politique de sécurité du système définit une valeur $\mathcal{N}_{max}$, en rapport avec le pourcentage de membres nécessaires pour réaliser une certification globale (k). $\mathcal{N}_{max}$ correspond au nombre maximum de membres d'un groupe de fragmentation, afin d'assurer qu'une certification globale requière l'accord d'au moins $k\%$ des nœuds. Lorsqu'un nœud détecte que le fragment S_{R_i} qu'il possède est répliqué sur plus de $\mathcal{N}_{max}$ autres nœuds actifs, il déclenche la fragmentation de ce fragment. L'objectif est d'obtenir deux groupes distincts, possédant $S_{R_{i0}}$ pour l'un et $S_{R_{i1}}$ pour l'autre, tels que $S_{R_{i0}} + S_{Ri1} = S_{R_i}$. Ce nœud demande alors un consensus de fragmentation, au début duquel chaque nœud place un verrou, chaque nœud pouvant alors soit proposer une valeur pour $S_{R_{i0}}$ s'il juge la fragmentation légale par rapport à la politique de sécurité, soit renvoyer un refus de fragmenter s'il juge cette opération illégale : cette alternative permet d'empêcher un nœud malveillant de déclencher une fragmentation inutile. À la fin du consensus, chaque nœud connaît son nouveau fragment et son identifiant et peut donc relâcher le verrou associé à l'opération. La première partie de l'invariant est garantie car chaque nouveau groupe est un sous-ensemble du groupe initial ; la seconde partie est garantie par $S_{R_{i0}} + S_{Ri1} = S_{R_i}$, chaque nœud connaissant S_{R_i} et $S_{R_{i0}}$ et choisissant $S_{R_{i1}} = S_{R_i} - S_{R_{i0}}$

Enfin, les nœuds ayant encore la possibilité de connaître les deux fragments résultant, chaque nouveau groupe initialise un rafraîchissement de son fragment.

6.4 Fusion des fragments

De la même façon que $\mathcal{N}_{max}$, la politique de sécurité définit une valeur $\mathcal{N}_{min}$, correspondant au nombre minimum de membres d'un groupe pour assurer la disponibilité du fragment lié. Lorsqu'un nœud détecte que le fragment $S_{R_{i0}}$ qu'il possède est répliqué sur moins de $\mathcal{N}_{min}$ autres nœuds actifs, il déclenche la fusion de ce fragment avec son fragment adjacent $S_{R_{i1}}$ (le fragment de même identifiant excepté le dernier bit). Pour cela, ce nœud commence par exécuter l'algorithme de découverte du groupe possédant $S_{R_{i1}}$, comme présenté précédemment. Chaque nœud possède ainsi la nouvelle liste des membres de son groupe. Ensuite, l'ensemble des nœuds réalise un consensus sur le nouveau fragment. Lors du premier tour, chaque nœud envoie son fragment possédé avant la fusion puis, lors des tours suivants, chaque nœud envoie la somme de son fragment possédé avant la fusion et du fragment supposé de l'autre groupe. À la fin du consensus, chaque nœud connaît le nouveau fragment et son identifiant et peut donc relâcher le verrou associé à l'opération. La première partie de l'invariant est garantie par les propriétés de l'algorithme de découverte ; la seconde partie est garantie par $S_{R_i} = S_{R_{i0}} + S_{R_{i1}}$, S_{R_i} étant le nouveau fragment possédé par ces nœuds.

6.5 Rafraîchissement des fragments

Régulièrement, chaque groupe décide de rafraîchir son fragment. Le principe est de mélanger différents fragments de clés, afin de rendre d'anciennes versions des fragments inutiles. Ce rafraîchissement peut être spontané ou déclenché par une fragmentation.

La première phase de cette opération consiste en l'algorithme de découverte d'un groupe quelconque, présenté précédemment. Ensuite, si l'algorithme s'est exécuté avec succès (chaque groupe a accepté l'opération), l'ensemble des deux groupes réalise un consensus sur une valeur aléatoire Δ. À l'issue de ce consensus, les membres de A soustraient Δ à leur fragment de clé et les membres de B ajoutent Δ à leur fragment. Si S_{R_A} et S_{R_B} sont les deux fragments initiaux et S'_{R_A} et S'_{R_B} les deux fragments finaux, nous avons bien $S_{R_A} + S_{R_B} = S'_{R_A} + S'_{R_B}$, ce qui assure que la clé de chiffrement globale est conservée (seconde partie de l'invariant). Enfin, tous les nœuds impliqués relâchent leur verrou et peuvent de nouveau participer à d'autres opérations. La première partie de l'invariant est garantie étant donnée que les groupes ne changent pas.

7 Simulations

Afin de valider nos propositions, nous avons simulé l'algorithme de certification dans l'environnement *PeerSim* [JJMV04]. *PeerSim* est un simulateur de système pair-à-pair, écrit en Java, permettant assez simplement de greffer de nouveaux algorithmes. Dans cette section, nous présentons quelques résultats liés à la certification dans un réseau structuré. Les paramètres de nos simulations sont le nombre de nœuds du réseau et le pourcentage de nœuds malicieux. Pour les algorithmes pessimistes, chaque nœud demande confirmation du chiffrement à 4 nœuds différents (ce qui donne en tout 5 résultats et assure de pouvoir trouver une majorité dans les réponses). Lors de chaque test, la certification optimiste est d'abord essayée. Ensuite, en cas d'échec (et *uniquement* en cas d'échec), l'algorithme pessimiste est déclenché. Si ce second algorithme échoue, alors la certification est un échec. En chaque point des graphiques, nous avons donc $nb_{optimiste} + nb_{pessimiste} + nb_{échec} = 100\%$.

Dans ces simulations, nous ne considérons pas le cas d'attaques sur des nœuds. Plus précisément, une attaque de type *vers* pourrait avoir lieu, permettant à une personne malveillante de prendre le contrôle d'un grand nombre de nœuds bienveillants, à l'insu des utilisateurs. Nous pensons que les moyens de se prémunir de telles attaques sont hors de ce travail. Ces simulations ont donc lieu dans un réseau où les nœuds sont bienveillants ou malveillants au seul choix de l'utilisateur qui les exécute.

Dans la figure 4, nous faisons varier le nombre de nœuds du réseau à pourcentage de nœuds malveillants constant (10%, ce qui constitue une fraction très importante, voire irréaliste, des nœuds dans le cas où l'on ne considère pas les attaques de type *vers*). Il apparaît logiquement que plus le réseau est grand, plus il est difficile de réaliser une certification sans rencontrer de nœud malveillant. Avec 500 nœuds, à peine 20% des chiffrements optimistes réussissent. Au-delà de 1500 nœuds, l'algorithme optimiste ne réussit jamais. En revanche, l'algorithme pessimiste semble résistant à cette difficulté, environ 90% des certifications se terminant avec succès.

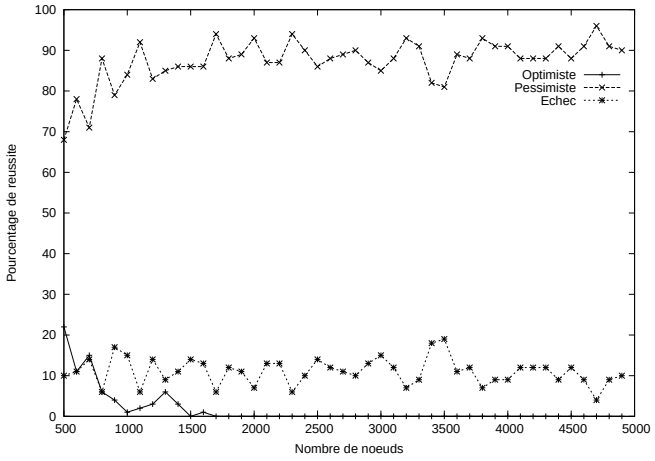


FIG. 4: Pourcentage de réussite des différents algorithmes en fonction du nombre de nœuds du réseau. Chaque expérience comporte 10% de nœuds malicieux et chaque nœud demande confirmation des chiffrements à 4 autres nœuds lors des algorithmes pessimistes.

Dans la figure 5, nous faisons varier le pourcentage de nœuds malveillants, à nombre de nœuds dans le réseau constant (2000). Nous avons choisi ce nombre de nœuds pour avoir une valeur médiane au graphe précédent. Il apparaît qu'à pourcentage de nœuds malveillants faible, l'algorithme optimiste remplit son rôle. À partir d'1% de nœuds malveillants, l'algorithme optimiste ne réussit plus que 60% du temps. À partir de 3% de nœuds malveillants, cet algorithme optimiste est très peu efficace, laissant à l'algorithme pessimiste la tâche de réaliser les certifications.

Dans les figure 6 et 7, nous illustrons la détection des nœuds malveillants au niveau du chiffrement. En effet, lors d'un chiffrement pessimiste, plusieurs nœuds sont interrogés pour un même calcul : il est

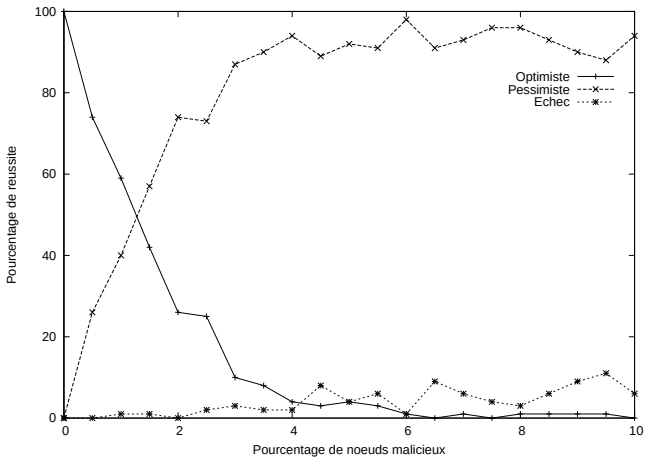


FIG. 5: Pourcentage de réussite des différents algorithmes en fonction du pourcentage de nœuds malicieux. Chaque expérience comporte 2000 nœuds et chaque nœud demande confirmation des chiffrements à 4 autres nœuds lors des algorithmes pessimistes.

donc possible, en plus de choisir le bon résultat, de détecter les nœuds malveillants. Ici, nous utilisons un algorithme simple, dans lequel chaque nœud rendant une réponse minoritaire est exclu des procédés de chiffrement suivant. Il faut néanmoins remarquer que ces graphiques présupposent certains mécanismes encore inachevés, étant donné que l'exclusion du réseau demande d'abord la constitution d'une preuve de mauvais comportement d'un nœud, preuve qui n'est pas encore formellement définie. Néanmoins, ces deux figures sont très encourageantes puisque, moyennant cette preuve, les nœuds malveillants sont très rapidement exclus. La figure 6 illustre les réussites des chiffrements successifs et fait apparaître qu'avec initialement 15% de nœuds malveillants parmi 2000 nœuds, l'algorithme optimiste de chiffrement devient très vite utilisable. Dans cette figure, l'axe des abscisses correspond au nombre de certifications tentées, que l'algorithme optimiste ait réussi ou non. Le nombre de réussites de l'algorithme optimiste atteint presque les 90% après seulement 10 certifications. De plus, il faut remarquer que plus l'algorithme optimiste réussit, moins l'algorithme pessimiste est appelé (cet algorithme pessimiste étant le seul à bannir des nœuds).

Enfin, la figure 7 montre le nombre de nœuds malveillants restant après chaque chiffrement, dans un réseau formé par 2000 nœuds. Le pourcentage de nœuds malveillants passe de 15% à 1% en l'espace de 5 certifications. À partir de 10 certifications, le nombre de nœuds malveillants devient presque négligeable, la courbe formant une hyperbole.

8 Conclusion et perspectives

Nous avons proposé un mécanisme distribué permettant le contrôle d'accès des nœuds dans un réseau pair-à-pair. Ce mécanisme est basé sur une certification distribuée des identifiants d'utilisateurs, *via* un nouveau système de cryptographie à seuil *adaptative*. Contrairement aux travaux antérieurs dans ce domaine, les seuils utilisés varient dynamiquement en fonction de la taille du réseau, afin de s'adapter aux fortes variations de taille des réseaux pair-à-pair. De plus, notre mécanisme passe à l'échelle, ce qui est nécessaire étant donnée la grande taille des réseaux pair-à-pair.

Le système présenté est cependant vulnérable à la *Sybil attack* [Dou02]. En effet, une personne malveillante peut générer un grand nombre d'identifiants de nœuds et ainsi récupérer chaque fragment de la clé privée du réseau, pour ensuite reconstituer cette clé privée intégralement. De plus, une personne malveillante peut choisir ces identifiants de nœuds et biaiser l'aléa nécessaire au bon fonctionnement du réseau. Nous proposons dans [LMT] une architecture de gestion des identifiants, permettant d'éviter qu'une personne soit en possession d'un grand nombre d'identifiants de nœuds ou puisse les choisir. Il est intéressant de noter que les réseaux pair-à-pair structurés sont eux-mêmes vulnérables à la *Sybil attack*, et que notre système n'introduit donc pas de nouvelle faiblesse dans ces réseaux.

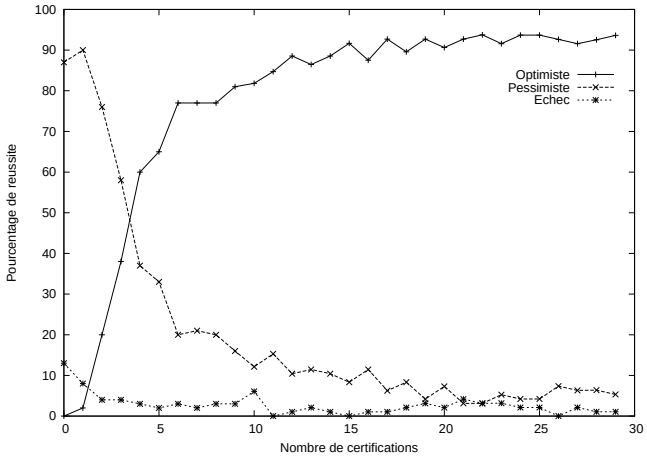


FIG. 6: Pourcentage de réussite des différents algorithmes en fonction du nombre de certifications déjà effectuées. Chaque expérience comporte 2000 nœuds, démarre avec 15% de nœuds malicieux et chaque nœud demande confirmation des chiffrements à 4 autres nœuds lors des algorithmes pessimistes.

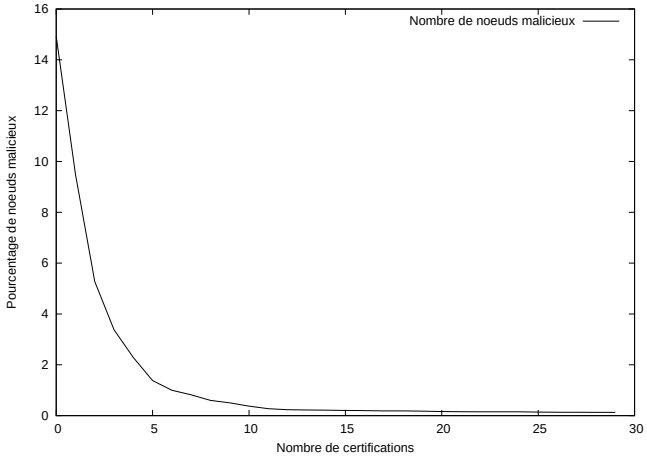


FIG. 7: Pourcentage de nœuds malicieux présents dans le réseau en fonction du nombre de certifications déjà effectuées. Chaque expérience comporte 2000 nœuds, démarre avec 15% de nœuds malicieux et chaque nœud demande confirmation des chiffrements à 4 autres nœuds lors des algorithmes pessimistes.

Le problème de la robustesse de la cryptographie à seuil en présence de nœuds malveillants n'est que partiellement résolu. En effet, il est possible qu'un grand nombre de nœuds malveillants parviennent à contrôler entièrement un fragment de la clé, rendant alors tout chiffrement impossible. Nous étudions actuellement la possibilité de maintenir plusieurs fragmentations différentes de la clé de réseau, réparties sur des groupes séparés différemment. Cette approche permettrait de basculer sur une autre fragmentation de la clé de réseau au cas où la courante deviendrait indisponible. Nous envisageons également la possibilité de maintenir des données de restauration de chaque fragment. Il sera également intéressant d'étudier comment renouveler la clé secrète du réseau.

Enfin, dans le système présenté, l'admission des nœuds est automatique. Ce travail est en fait la première partie d'un projet consistant à matérialiser l'accès au réseau (présenté ici), exclure les nœuds malveillants et empêcher ces nœuds exclus de se reconnecter. Les causes menant à l'exclusion d'un nœud ainsi que les protocoles menant à l'exclusion effective de ce membre sont encore à définir.

Références

[AKNRT04] Yair Amir, Yongdae Kim, Cristina Nita-Rotaru, and Gene Tsudik. On the performance of group key agreement protocols. *ACM Transactions on Information and System Security (TISSEC)*, 2004.

[BF97] Boneh and Franklin. Efficient generation of shared RSA keys. In *Proceedings of the 17th Annual International Cryptology Conference (CRYPTO)*, volume 1294 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[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 (AAAIDEA)*, 2005.

[CCR04] Miguel Castro, Manuel Costa, and Antony Rowstron. Peer-to-peer overlays : structured, unstructured, or both ? Technical Report MSR-TR-2004-73, Microsoft Research (MSR), 2004.

[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. ACM Press, 2002.

[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. ACM Press, 2003.

[Cli00] Clip2. The gnutella protocol specification v0.4. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`, 2000.

[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-Verlag, 2000.

[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-Verlag, 1997.

[Dou02] John R. Douceur. The sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer-Verlag, 2002.

[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. IEEE Computer Society, 2001.

[FGMY97] 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. IEEE, IEEE Computer Society, 1997.

[GJKR96] 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. Springer-Verlag, 1996.

[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. Springer-Verlag, 2004.

[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.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or : How to cope with perpetual leakage. *Lecture Notes in Computer Science*, 963 :339–352, 1995.

[JJMV04] Márk Jelasity, Gian Paolo Jesi, Alberto Montresor, and Spyros Voulgaris. PeerSim P2P Simulator. `http://peersim.sourceforge.net/`, 2004.

[KMT03] Yongdae Kim, Daniele Mazzochi, and Gene Tsudik. Admission control in peer groups. In *Proceedings of th IEEE International Symposium on Network Computing and Applications (NCA)*, pages 131–139. IEEE Computer Society, 2003.

[Kur02] Klaus Kursawe. Optimistic byzantine agreement. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 262–267. IEEE Computer Society, 2002.

[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.

[LLY06] Patrick P.C. Lee, John C. S. Lui, and David. K. Y. Yau. Distributed collaborative key agreement and authentication protocols for dynamic peer group. *IEEE/ACM Transactions on Networking*, 2006.

[LMT] François Lesueur, Ludovic Mé, and Valérie Viet Triem Tong. Gestion distribuée d'identités résistante à la sybil attack pour un réseau pair-à-pair. Soumis à SAR 2007.

[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. Springer-Verlag, 1998.

[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. Springer-Verlag, 2001.

[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 SIGCOMM Conference (SIGCOMM)*, volume 31, 4 of *Computer Communication Review*, pages 161–172. ACM Press, 2001.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), 1979.

[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 SIGCOMM Conference (SIGCOMM)*, Computer Communication Review, pages 149–160. ACM Press, 2001.

[STY03] 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)*, 2003.

[Was04] Waste. `http://waste.sourceforge.net`, 2004.

[ZKJ01] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry : an infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, University of California at Berkeley, apr 2001.