Université Joseph Fourier – Grenoble 1

Master Informatique et Mathématiques 2^{nde} année, spécialité Systèmes et Logiciels

Rapport de stage

Non-observabilité des communications interactives

Nicolas Bernard
<n.bernard@lafraze.net>

Stage réalisé sous la direction de

Yves Denneulin
<Yves.Denneulin@imag.fr>
INPG – Laboratoire ID-Imag

Franck Leprévost
<franck.leprevost@cu.lu>
Université du Luxembourg

Juin 2004

Remerciements: C'est un plaisir pour moi de remercier mes deux directeurs de stage, Yves Denneulin, sur Grenoble, dont j'ai fait la connaissance cette année mais que j'ai tout de suite trouvé très sympathique, et Franck Leprévost, à Luxembourg, que je connaissais déjà d'un enseignement précédent, pour leur encadrement, leurs conseils amicaux, et un tas d'autres choses. De manière plus générale, je remercie également les gens du laboratoire ID-IMAG à Grenoble ainsi que les différentes personnes que j'ai eu l'occasion de rencontrer à Luxembourg, et plus particulièrement Jean-Claude Asselborn pour son accueil chaleureux à mon arrivée à Luxembourg, et Sébastien Varrette, qui entre dans les deux catégories précédentes (Grenoblois *et* Luxembourgeois), avec qui j'ai eu le plaisir de partager mon bureau à Luxembourg.
Je voudrais également remercier Olivier Aumage, pour certaines discussions sur la Bonne Façon de programmer, qui m'ont influencé dans la phase d'implémentation.
Et finalement, une mention spéciale aux relecteurs du présent rapport, ils se reconnaîtront.

Colophon: le présent rapport a été réalisé en utilisant le système de mise en forme LaTeX. Les figures ont été réalisées avec le logiciel *xfig*.

Table des matières

In the once-upon-a-time days of the First Age of Magic, the prudent sorcerer regarded his own true name as his most valuated possession, but also the greatest threat to his continued good health, for — the stories go — once an enemy, even a weak unskilled enemy, learned the sorcerer's true name, then routine and widely known spells could destroy or enslave even the most powerful. As time passed, and we graduated to the Age of Reason and thence to the first and second industrial revolution, such notions were discredited. Now it seems that the Wheel has turned full circle (even if there never really was a First Age) and we are back to worrying about true names again (...).

— Vernor Vinge, *True Names* [20]

Introduction : contexte et définitions

Nous présentons dans le présent rapport nos réflexions sur les techniques à utiliser pour rendre des communications interactives (sur un réseau de type IP public tel que l'Internet) non-observables afin de préserver l'intimité des utilisateurs, ainsi qu'un canevas implémentant une partie des idées découlant de ces réflexions qui, s'il est encore assez sommaire, n'en est pas moins déjà utilisable.

Avant d'entrer dans le vif du sujet, commençons par rappeler les bases indispensables de cryptologie :

Rappels de cryptologie

On pourrait dire que la cryptologie est la science des codes secrets [1]. Elle regroupe la cryptographie, l'art de faire de tels codes, et la cryptanalyse, qui consiste au contraire à les casser, c'est-à-dire généralement à obtenir le message déchiffré à partir uniquement du message chiffré (sans la clef).

On appelle message en clair, ou plus simplement *clair* le message que l'on veut transmettre à notre correspondant. On le note généralement M. C dénote le message chiffré, ou *chiffré*.

Il existe principalement deux branches de la cryptographie : à clef secrète et à clef publique.

Cryptographie à clef secrète

La cryptographie à clef secrète (ou *symétrique*) existe depuis plusieurs millénaires. On a ici un algorithme de chiffrement E, un algorithme de

1. Le mot "code" devant être entendu ici avec un sens large, et non dans son sens plus restreint qui le distingue d'un *chiffre*.

déchiffrement D. Le chiffré est obtenu à partir du clair de la manière suivante :

$$C = E(M,k)$$

où k est la *clef*. Cette clef doit rester secrête. Si un bon algorithme est utilisé, il ne doit pas être possible de déchiffrer le message sans cette clef en temps raisonnable.

A partir du chiffré, on peut récupérer le message en clair par l'opération suivante :

$$M = D(C,k)$$

La cryptographie à clef secrète permet donc de transmettre des informations de manière sûre par un réseau : le message C peut tomber en de mauvaises mains sans crainte du moment que la clef k reste secrète. Et c'est là le problème : si on envoie C à quelqu'un, il doit aussi connaître k pour pouvoir récupérer M ; et k ne peut bien sûr pas être envoyée par le réseau...

Ce problème pourrait paraître insoluble, mais ce n'est pas le cas, la cryptographie à clef publique permet en effet de le résoudre.

Cryptographie à clef publique

Contrairement à la cryptographie à clef secrète, la cryptographie à clef publique (ou *asymétrique*) est récente : elle date du milieu des années 1970 (10 ans plus tôt pour les services secrets britanniques !).

Ici, chaque utilisateur possède une paire de clefs : une clef publique k_{pub} et une clef privée k_{priv}. La clef privée doit être jalousement gardée par son possesseur, contrairement à la clef publique qu'il diffuse largement.

Quand Alice [2] veut envoyer un message à Bob, elle commence par récupérer la clef publique de Bob, k_{pub}^{Bob}. Elle chiffre le message d'une manière en apparence — tant que l'on ne regarde pas le fonctionnement interne de E et D — très similaire à ce que l'on a vu précédemment :

$$C = E(M,k_{pub}^{Bob})$$

Néanmoins, et c'est là toute la beauté de la chose, l'algorithme est prévu pour qu'il soit maintenant impossible de déchiffrer C sans connaître k_{priv}^{Bob}, la clef privée de Bob! Alice elle-même ne peut plus déchiffrer C !

Bob, lui, en recevant le message, pourra effectuer

$$M = D(C,k_{priv}^{Bob})$$

2. Les cryptographes utilisent un certain nombre de noms standards. L'annexe B présente succinctement ceux que nous faisons intervenir dans le présent document.

et ainsi déchiffrer le message d'Alice.

Ajoutons finalement que si Alice chiffre le message avec sa clef privée, tous les possesseurs de sa clef publique pourront le déchiffrer. Quel intérêt ? Comme seule Alice a pu chiffrer ainsi le message, cela correspond à une signature de sa part. C'est une *signature numérique*.

L'algorithme de chiffrement asymétrique le plus connu est sans doute RSA, dont nous reparlerons par la suite.

La cryptographie à clef publique peut sembler bien supérieure à la cryptographie à clef secrète, du fait des avantages qu'elle offre. Elle a cependant de gros inconvénients, le principal étant qu'elle est beaucoup plus lente (Le chiffre de mille fois plus lente est souvent avancé. . .). Aussi ces deux types de cryptographie sont souvent associés en des systèmes mixtes, où les données sont chiffrées avec un algorithme et une clef de session (qui ne sert donc qu'une fois) symétrique, et la clef de session elle-même est chiffrée avec un algorithme asymétrique et la clef publique du destinataire.

Le lecteur est invité à se reporter à [17] pour plus de renseignements sur la cryptographie. Le chapitre 5 de [1] pourra également être utile. [9] donne un intéressant recul sur l'histoire (et le rôle dans l'Histoire !) de la cryptographie.

Limites des protocoles de "sécurisation" actuels

Bien qu'il existe un certain nombre de protocoles pour "sécuriser" les communications via l'Internet (par exemple SSL et IPsec), un certain nombre d'informations ne sont absolument pas dissimulées à un éventuel observateur.

En effet, ces protocoles ont pour but, en utilisant les techniques cryptographiques résumées dans la section précédente, d'une part de rendre inutilisables les informations transmises via le réseau pour ledit observateur en les lui rendant incompréhensibles, d'autre part d'identifier et d'authentifier un utilisateur de manière fiable, afin que des données confidentielles ne soient pas transmises à un attaquant se faisant passer pour quelqu'un d'autre. Par contre, dans ces protocoles "sécurisés", un observateur peut toujours déterminer quelles sont les parties qui communiquent, pendant combien de temps elles communiquent, et avoir une idée plus ou moins précise de la quantité de données échangées. Ces protocoles sont observables.

Cela semble anodin, mais peut cependant avoir des répercussions d'ampleur non négligeable. Par exemple, prenons la consultation d'un site web statique (i.e. dont les pages ne changent pas et sont les mêmes pour chaque

visiteur) : Alice visite donc le site (statique) de Bob en utilisant le protocole `https` (c'est-à-dire le protocole du web "sécurisé" en le faisant passer dans SSL). Eve qui écoute le trafic sait qu'Alice consulte ce site, par contre elle ne peut voir quelles pages elle y consulte. Cependant, on peut imaginer qu'elle enregistre chaque requête (elle peut les distinguer par les pauses qui les séparent) et mesure la quantité de données échangées pour chacune. Ensuite, elle consulte elle-même le site de Bob et en télécharge (en utilisant également **https**) chaque page en procédant à des mesures similaires. En établissant des correspondances entre les deux jeux de mesures, elle a de bonnes chances de retracer le parcours d'Alice qui utilisait pourtant un protocole "sûr" pour consulter ce site !

Anonymat, pseudonymat, réputation et non-observabilité

Dans ce contexte, l'**anonymat** consiste en ce que le destinataire d'un message ne puisse déterminer son émetteur[3]. Un réel anonymat implique donc la non-traçabilité (*i.e.* : l'impossibilité de remonter jusqu'à l'expéditeur réel) d'un message reçu.

Parallèlement, le **pseudonymat** consiste en ce qu'un utilisateur, qui possède un ou plusieurs pseudonymes, communique avec d'autres de ces pseudonymes, sans savoir qui sont les utilisateurs "réels" qui leur correspondent. On peut considérer les adresses emails comme un exemple possible de tels pseudonymes, étant donné qu'une telle adresse ne reflète pas forcément l'identité réelle de son possesseur. Néanmoins, étant donné que le champ `From:` des emails est très facile à usurper, on ne peut généralement pas savoir si un message email vient bien du pseudonyme qui semble l'avoir envoyé. C'est pourquoi, la définition communément admise, dans le contexte qui nous intéresse, est celle de [3], "Un *pseudonyme* numérique est une clef publique utilisée pour vérifier les signatures faites par le porteur anonyme de la clef privée correspondante", auquel on peut associer un moyen de joindre ledit porteur.

Nous introduisons maintenant la notion de **nyme**. Cryptographiquement parlant, c'est un pseudonyme, néanmoins, comme dans le cas d'une adresse

3. L'anonymat est généralement à *sens unique*, l'expéditeur devant savoir à qui il envoie le message. Il peut néanmoins exister des cas d'anonymat du récepteur : par exemple la publication d'un article usenet peut être considérée ainsi puisque l'on ne peut savoir *qui* exactement va le lire. Si l'article en question a en plus été posté de manière anonyme, on a alors de l'anonymat *bi-directionnel*.

mail, il peut y avoir une correspondance publique avec une certaine personne ou non.

En effet, un nyme peut être associé de manière connue à une entité physique ou morale : c'est ce à quoi l'on essaie généralement de parvenir via des modèles hiérarchiques comme les *infrastructures à clef publique* (PKI) ou via des modèles distribués comme le *Web of Trust* de PGP. Le but d'une telle bijection est en fait d'associer une **réputation** (celle de la personne réelle, que ce soit "c'est quelqu'un de bien" ou "c'est le chef") au nyme. Néanmoins, il est tout à fait possible d'avoir des nymes qui n'aient pas de telles correspondances établies de manière publique : dans ce cas, leur réputation ne dépendra que de leurs "actes" (messages envoyés, etc.). Il est donc important que l'on ne puisse établir la relation nyme-utilisateur si ce dernier ne le désire pas.

A l'inverse, il peut également être permis à un utilisateur de changer de nyme pour chaque message qu'il envoie : dans ce cas on retrouve l'anonymat.

La **non-observabilité** consiste simplement, d'une part, à ce que pour une communication donnée, à laquelle on ne participe pas, on ne puisse déterminer aucune des personnes entre lesquelles elle se déroule ni son contenu et, d'autre part, à ce que pour un utilisateur donné on ne puisse déterminer ni quand il communique, ni avec qui il communique (et bien sûr que le contenu des conversations reste secret dans le cas où ce ne serait pas impliqué par les hypothèses précédentes).

Bénéfices et maléfices de la non-observabilité

On peut se demander "à quoi sert réellement d'avoir des communications non-observables". En fait, le débat n'est pas nouveau et est très similaire à celui sur la cryptographie[4] : d'une part, cela sert à garantir la vie privée d'un utilisateur, d'autre part des abus sont possibles. Notons toutefois que, dans le cas de la cryptographie, les prévisions alarmistes des défenseurs d'une "cryptographie d'état" ne se sont pas vérifiées après la libéralisation qui a eu lieu dans les années 1990 : ainsi, un récent rapport [11] indique qu'aux États-Unis les écoutes téléphoniques fédérales n'ont rencontré aucun cas de communication chiffrée.

4. Ce débat a fait rage au début des années 1990 aux Etats-Unis où le gouvernement a tenté d'imposer aux utilisateurs de moyens cryptographiques de déposer une copie de leurs clefs auprès d'un organisme agréé par l'état.

Utilité

Les emplois “bénéfiques” de la non-observabilité sont nombreux, tant dans le cadre de l'intimité des personnes...

- pour le particulier, dans la vie de tous les jours, cela peut lui permettre d'utiliser le Web sans que son fournisseur d'accès puisse déterminer ce qu'il consulte. C'est particulièrement intéressant dans le cas où le fournisseur d'accès et la personne en question se connaissent, ce qui peut éveiller la curiosité du premier (cas de fournisseurs d'accès locaux, connexions à internet partagées, etc.) ;
- dans le même ordre d'idées, on peut penser au secret de la relation médecin / patient : dans certains cas, le simple fait de consulter un site web sur telle maladie particulière peut laisser supposer que vous ou un de vos proches en êtes atteint, ce qui dans certains pays est un motif suffisant pour un licenciement... L'utilisation de nymes non-observables peut aussi permettre à différentes personnes atteintes d'une maladie de discuter sans révéler leur véritable identité ;
- on peut également songer aux avocats qui peuvent avoir besoin d'entretenir des rapports confidentiels que la police devrait ignorer ;
- de même les banques utilisent couramment des nymes : généralement ce sont des nymes dont l'association avec une entité physique ou morale est bien connue, mais ce n'est pas obligatoire (cas des comptes numérotés). Dans ce dernier cas, la non-observabilité du nyme est importante ;
- on peut également penser aux personnes qui sont susceptibles d'être surveillées à des fins politiques : même dans une démocratie de telles écoutes peuvent exister. L'affaire dite des “écoutes téléphoniques de l'Élysée” le montre.
- etc.

... que dans celui de la liberté d'expression :

- un autre aspect intéressant est que l'utilisation d'un nyme peut parfois permettre à des personnes qui ne seraient normalement pas écoutées (en raison de préjugés divers, qu'ils soient liés à l'âge, à la couleur ou aux croyances) de faire entendre leur opinion ;
- l'on peut également songer aux cas des pays où la liberté d'expression est restreinte : un tel système peut permettre à des “dissidents politiques” de publier des informations sans risquer d'être arrêtés.

Les quatre cavaliers de l'Infocalypse

A ces aspects "bienfaisants" listés précédemment, il faut néanmoins ajouter un certain nombre de choses "maléfiques" qu'un tel système est susceptible d'encourager. Celles-ci sont parfois appelées plaisamment les *Quatre Cavaliers de l'Infocalypse* car vues comme des prétextes utilisables par un gouvernement pour transformer Internet en une zone ultra-surveillée.

- Les terroristes : ils pourraient utiliser un tel système pour communiquer en échappant à la surveillance de la police ;
- les pédophiles : idem ;
- cela permettrait aussi la mise en place d'un système de paris sur la date de mort de célébrités, avec la conséquence logique : il suffit de tuer la personne pour gagner. Cela fait qu'il suffirait donc en fait de "parier" une somme suffisante sur quelqu'un pour engager un tueur... A ce jour, le seul exemple d'un tel système qui nous soit connu [8] avait été mis en place par... le département de la défense américain !
- etc. Il n'y a pas exactement quatre cavaliers, d'autant plus que selon les personnes, certains des exemples classés ici comme "bons" peuvent devenir "mauvais" et inversement.

Ce bref résumé des positions est bien sûr incomplet, voire caricatural. Le lecteur intéressé pourra se reporter à [10], qui regroupe et met en rapport des textes des différents bords, permettant ainsi à chacun de se faire sa propre opinion sur le sujet.

Notons simplement que la dernière position, celle où l'on confie ses clefs à l'État nous semble avoir une faille fondamentale : l'hypothèse implicite est que l'État et ses représentants sont "bons", ce qui, même dans une démocratie, semble impossible à garantir. L'histoire du XX$^{\text{eme}}$ siècle l'a prouvé à maintes reprises [5]...

5. Sans compter qu'une interdiction des moyens de cryptographie incassables par l'État aurait toutes les chances de conduire au scénario où seuls les citoyens "honnêtes" utiliseraient cette cryptographie "faible", tandis que les criminels utiliseraient de toute façon une cryptographie forte. Il y a là une parenté avec les lois interdisant les armes : les braqueurs de banques ne semblent pas en manquer...

Chapitre 1

Un rapide état de l'art

> Morality is always the product of terror; its chains and strait-waistcoats are fashioned by those who dare not trust others, because they dare not trust themselves, to walk in liberty.
>
> — Aldous Huxley

Si des protocoles satisfaisants, basés sur les idées de Chaum [3], existent depuis une quinzaine d'années pour les communications non interactives (protocoles *cypherpunk* [4] puis *mixmaster* [12] pour les emails par exemple), il n'en est pas ainsi dans le cadre des communications interactives. Nous allons rapidement passer en revue ce qui existe.

Il est préalablement intéressant d'introduire une analogie avec le monde réel : on peut en effet comparer les communications non interactives à un échange épistolaire : une personne écrit une lettre à une autre, l'envoie, le destinataire la reçoit, la lit, puis, éventuellement, au bout d'un certain temps, y répond. Au contraire, les communications interactives pourraient être comparées à une conversation : une personne dit quelque chose (beaucoup plus bref que le contenu d'une lettre), puis, aussitôt, l'autre y répond, et ainsi de suite.

Ces deux types de communications existent sur Internet : ainsi la diffusion des emails, la propagation des messages usenet, etc. peuvent prendre quelques minutes ou quelques heures, tandis que la consultation du web, les "chats", les protocoles de connexion à distance (telnet, ssh, ...) ou les services de téléphonie (VoIP) tombent manifestement dans la catégorie des communications interactives.

1.1 Remailers Anonymes

Les remailers anonymes sont des systèmes permettant d'envoyer des emails ou des messages usenet de manière non seulement anonyme vis-à-vis de leurs destinataires, mais également, pour les plus récents, de manière non-observable.

1.1.1 Type 0

L'un des premiers remailers (1993) (type 0) était un système finlandais, `penet`, opéré par Julf Helsingius, qui permettait tout simplement à ses utilisateurs d'envoyer des mails en indiquant comme expéditeur une adresse sur cette machine. Le destinataire pouvait répondre à cette adresse et les messages étaient alors transmis à l'expéditeur original. Cela impliquait néanmoins la conservation d'une table de correspondance dans cette machine entre les nymes des utilisateurs et leur adresse email réelle... L'histoire finit comme on peut le supposer : en 1995 une entité (en l'occurrence l'Eglise de Scientologie) n'a pas apprécié certaines des informations à son sujet divulguées par l'un des utilisateurs de ce remailer, a fait un procès à son opérateur, et la police[1] a obligé celui-ci à révéler l'adresse des personnes incriminées.

1.1.2 Type 1

Cela a montré, s'il le fallait, la nécessité que l'anonymat ne dépende pas d'un unique point. Cependant, parallèlement, certaines des idées de l'article de Chaum précédemment évoqué avaient été mises en pratique, donnant les remailers dits *cypherpunks*, ou type 1.

Un tel protocole est basé sur le reroutage des messages : l'expéditeur envoie un message à un premier de ces routeurs (appelés remailers) ; celui-ci déchiffre le message (qui était chiffré avec sa clef publique) et découvre un autre message ainsi qu'une adresse. Il renvoie alors le message à cette adresse, qui est généralement celle d'un autre remailer, qui fera de même. Finalement, après un certain nombre de passages par différents remailers l'adresse déchiffrée sera celle du vrai destinataire du message.

Il y a néanmoins des attaques possibles : si un tel système donne un bon anonymat (il faut pouvoir remonter toute la chaîne pour parvenir à l'expéditeur d'un message), un adversaire puissant, qui serait capable de surveiller les liens réseaux des remailers, pourrait casser le système simplement en observant les messages entrer puis ressortir quelques instants plus tard de

1. La police, pas la justice...

$$
\begin{aligned}
\text{Alice} \rightarrow R_1 &: \left\{ R_2, \left\{ R_3, \left\{ \ldots, \left\{ Bob, \{M\}_{pk_B} \right\}_{pk_{R_n}} \ldots \right\}_{pk_{R_{\ldots}}} \right\}_{pk_{R_2}} \right\}_{pk_{R_1}} \\
R_1 \rightarrow R_2 &: \left\{ R_3, \left\{ \ldots, \left\{ Bob, \{M\}_{pk_B} \right\}_{pk_{R_n}} \ldots \right\}_{pk_{R_{\ldots}}} \right\}_{pk_{R_2}} \\
R_2 \rightarrow R_3 &: \left\{ \ldots, \left\{ Bob, \{M\}_{pk_B} \right\}_{pk_{R_n}} \ldots \right\}_{pk_{R_{\ldots}}} \\
&\vdots \\
R_{n-1} \rightarrow R_n &: \left\{ Bob, \{M\}_{pk_B} \right\}_{pk_{R_n}} \\
R_n \rightarrow Bob &: \{M\}_{pk_B}
\end{aligned}
$$

FIG. 1.1 – *Alice envoie un message à Bob à travers une chaîne de remailers.*

chaque remailer. Bien que le message ait changé (une couche de chiffrement a été enlevée) l'observateur est encore capable de le reconnaître (par sa taille, par le moment auquel il ressort si le trafic est faible, etc.). Ceci n'est que la plus simple des attaques possibles, et il en existe un certain nombre d'autres dans lesquelles l'attaquant peut être passif comme ici, mais aussi actif (insertion de messages, etc.). Le lecteur intéressé pourra se reporter à [4] qui en résume quelques autres.

1.1.3 Type 2

Pour pallier à ces attaques a commencé, en 1996, le développement du protocole *mixmaster*. Les changements les plus notables sont l'utilisation d'un temps d'attente à chaque remailer et le fait qu'un padding soit utilisé pour que tous les messages aient une taille similaire. On notera le clin d'oeil adressé par le nom du protocole à l'article de Chaum, dans lequel les remailers sont appelé des *mix*.

Chaque routeur ne renvoie pas immédiatement les messages qu'il reçoit : il attend d'en avoir reçu un certain nombre pour les envoyer alors dans un ordre différent et aléatoire. Comme tous les messages ont la même taille (un padding est utilisé) et que le message entrant ne ressemble pas au message sortant (une "couche" de chiffrement a été enlevée), un observateur ne peut donc pas déterminer la correspondance entre les messages entrants et les messages sortants. Si on ajoute le fait qu'un message passe généralement

par une chaîne de remailers, on voit que l'observateur n'a aucun moyen de déterminer où il aboutit. En fait, il ne peut même pas déterminer *quand* un utilisateur envoie réellement des messages car la possibilité d'envoyer de faux messages (des messages "vides" qui se perdent dans le réseau de remailers) existe.

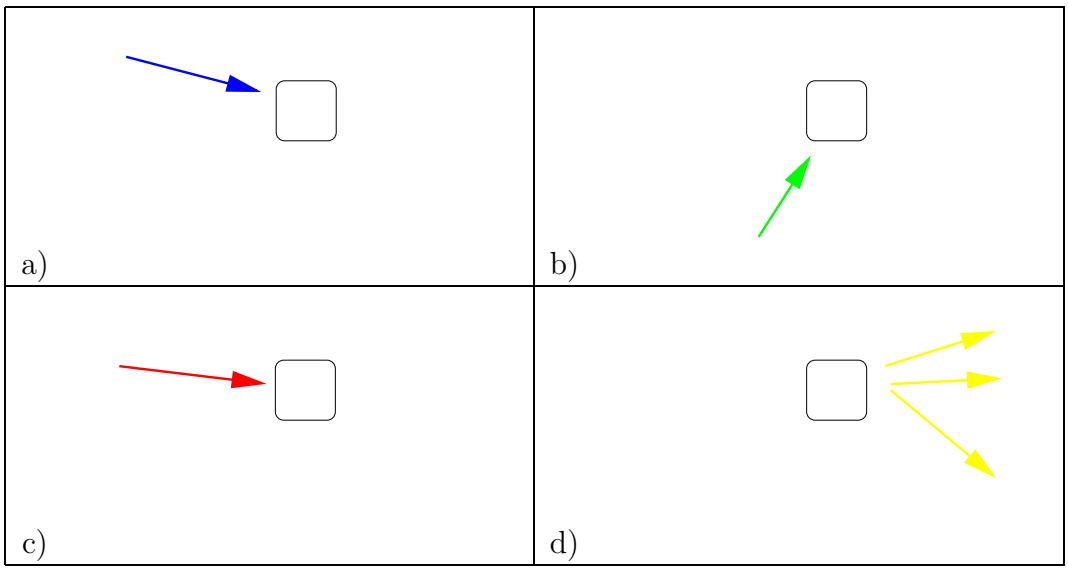


FIG. 1.2 – *Un noeud mixmaster reçoit un certain nombre de messages (a,b,c) puis les renvoie (d) tous plus ou moins simultanément (en tout cas dans un ordre indépendant de l'ordre d'arrivée) vers le noeud suivant après avoir enlevé une "couche" de chiffrement.*

L'un des points essentiels de ce système est le temps d'attente sur chaque remailer, où l'on attend la présence d'un nombre suffisant d'autres messages. Cela fait qu'un message envoyé en utilisant mixmaster peut facilement mettre quelques dizaines d'heures avant d'atteindre sa destination. Et ces attentes sont bien sûr impossibles pour des communications interactives...

La version deux de *mixmaster* est actuellement en cours de standardisation par l'Internet Engineering Task Force (IETF). Un *Internet-draft* est disponible [12].

1.1.4 Type 3

Le défaut des types 1 et 2 par rapport au remailer de Helsingius (et qui explique le succès de celui-ci alors même qu'un réseau de remailers de type 1 était déjà déployé) est qu'ils ne permettent pas au destinataire de répondre.

Le projet *mixminion*, actuellement en développement actif, devrait pallier à ce problème avec ce qui sera le type 3 de remailer.

Il faut cependant noter qu'il existe un moyen d'avoir des échanges anonymes dans les deux sens (*i.e.* : avec une personne dont on ne connaît pas l'adresse mail) avec les types 1 et 2 de remailers : il suffit de poster les messages anonymes dans un groupe usenet convenu, par exemple `alt.anonymous.messages`, avec un sujet indiquant le destinataire. Avec l'utilisation de remailers de type 2, un tel schéma est inobservable si le destinataire prend soin de télécharger tous les messages du forum en question et pas uniquement ceux qui lui sont destinés.

1.2 Onion-Routing

En ce qui concerne les communications interactives, les choses sont moins simples : déjà la communication doit être bi-directionnelle, ensuite, elle doit être établie entre les deux parties qui communiquent (contrairement au cas des remailers où une connexion est faite vers le premier de la chaîne, le message est envoyé, la connexion fermée, puis le remailer itère le processus vers le suivant de la chaîne).

Pour ce qui est de l'anonymat, un système similaire à celui des *mix-net* des remailers, appelé ici *onion-routing*[2] [15], est généralement utilisé, à ceci près que l'on ne transmet pas par leur intermédiaire *un* message, mais un certain nombre de paquets qui sont chiffrés individuellement. Il s'agit en fait de tunnels réseaux chiffrés : l'émetteur établit un tunnel vers une première machine, puis, en passant par ce tunnel en crée un autre vers une seconde machine, et ainsi de suite jusqu'au destinataire.

Un tel système offre ce que nous appelerons de l'*anonymat à sens unique* : le destinataire de la communication ne peut déterminer qui l'appelle, mais l'appelant doit connaitre l'adresse réseau du destinataire, ce qui revient généralement à connaître son identité. Un tel système est vulnérable à des attaques telles que celles évoquées précédemment face à un adversaire puissant, qui peut alors déterminer qui appelle qui.

Il existe plusieurs systèmes de ce type décrit dans la littérature, reposant sur différentes architectures de réseaux sous-jacentes : ainsi, par exemple, *Tarzan* [6] se situe au niveau IP et utilise une architecture pair-à-pair tandis que *Tor* [5] utilise une architecture plus classique de noeuds "fiables" (mais passant mal à l'échelle, les noeuds ayant des communications formant un graphe complet) et se trouve au niveau de TCP.

2. L'analogie avec des oignons étant que chaque paquet transmis par un tel système est chiffré de multiples fois et que chaque routeur "pèle" une couche de chiffrement...

1.2.1 Tarzan

Tarzan est l'un des modèles interactifs utilisant le plus de protections contre l'analyse de trafic. Néanmoins quelques-uns des moyens utilisés (l'utilisation de *mimiques* invariantes avec certains noeuds par exemple) nous semblent sujets à caution. D'autre part, Tarzan ne résoud pas le problème de l'anonymat dans les deux sens.

1.2.2 Tor

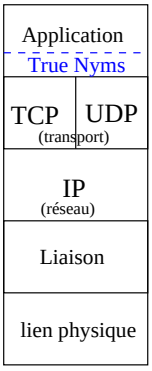
Le but principal de Tor est d'offrir de l'anonymat pour des connexions TCP. Il offre en particulier une interface SOCKS, ce qui permet à un certain nombre d'applications de l'utiliser via un paramétrage en *proxy*, donc sans modifications. Il n'utilise cependant aucune protection contre l'analyse de trafic autre que des paquets de taille identique. Il propose également des points de rendez-vous qui sont établis d'une manière élégante, un nyme établissant une communication avec un noeud qu'il choisit puis diffusant sur une table de hachage distribuée l'information qu'il faut envoyer les messages pour établir une communication avec lui au noeud en question. Ce système ne fournit néanmoins pas la non-observabilité et il nous semble qu'un attaquant peut découvrir le nyme d'un utilisateur particulier ; en effet, étant donné que les utilisateurs ne font pas partie du réseau de reroutage, l'absence de faux trafic peut permettre d'envoyer une demande d'établissement de communication à chaque utilisateur et de voir quand celui que l'on surveille reçoit des paquets...

Notre réflexion, décrite au chapitre suivant, a porté sur les moyens d'obtenir un système permettant des communications entre nymes qui soient non-observables, tout en ayant une faible latence une fois la communication établie. Le temps d'établissement de la communication, lui, peut être relativement long (De telles contraintes se retrouvent par exemple dans les applications de téléphonie par Internet (VoIP)).

Chapitre 2

Non-observabilité pour les communications interactives

> Electronic communication will be the fabric of tomorrow's society (...). By codifying the Governement's power to spy invisibly on these contacts, we take a giant step toward a world in which privacy belongs only to the wealthy, the powerful, and perhaps, the criminals.
>
> — Whitfield Diffie



Notre schéma est basé sur une architecture pair-à-pair, à la manière de Tarzan par exemple. Néanmoins, alors que Tarzan se situe "entre" les couches *réseau* et *transport* du modèle TCP/IP, nous avons préféré nous placer plus haut, entre les couches *transport* et *application*, comme le fait par exemple une bibliothèque de chiffrement des communications standard telle qu'OpenSSL.

L'idée du système est simple : les deux participants se connectent à un noeud déterminé, appelé *point de rendez-vous*, en utilisant l'onion-routing. Le point de rendez-vous transmet tout ce qui arrive sur l'une des communications à l'autre, et *vice versa*. Étant donné que chacun des participants utilise une communication bénéficiant de l'anonymat à sens unique vers ce point de rendez-vous, ce dernier ne peut déterminer leur identité (ni le nyme, ni l'identité réelle).

L'intérêt d'utiliser une architecture pair-à-pair est double : d'une part cela permet d'utiliser les ressources des utilisateurs pour la fourniture du service lui-même, ce qui est utile étant donné que les communications que l'on fait passer peuvent utiliser, si elle sont nombreuses, beaucoup de bande passante.

D'autre part, l'utilisateur étant intégré au réseau de routeurs, il est difficile pour un observateur de séparer les communications dans lesquelles un noeud est partie prenante de celles qu'il se contente de router. L'inconvénient est que l'on ne peut supposer les noeuds fiables. Nous reviendrons sur ce problème plus bas.

Différents moyens détaillés plus bas sont utilisés pour déjouer les attaques par analyse ou insertion de trafic et s'approcher de la non-observabilité.

2.1 Etablissement de la communication

Soient Alice et Bob deux nymes. Alice possède la clef publique de Bob. Alice désire entrer en communication avec Bob. Pour cela:

- Elle crée un message contenant une clef de session $\mathcal{K}_S$, et une liste de couples (noeud, numéro de session). Nous appelons *noeuds de rencontre* les noeuds de cette liste.
- Optionnel : elle signe le message avec sa clef privée, pour que Bob puisse savoir qui l'appelle.
- Elle chiffre ce message avec la clef publique de Bob.
- Elle envoie ce message chiffré à un serveur via une communication anonyme à sens unique à l'un des noeuds du réseau qui se chargera de le diffuser aux noeuds voisins qui en feront autant, et ainsi de suite sur tout le réseau.

Tous les utilisateurs reçoivent donc le message, mais seul Bob peut le déchiffrer. Alice et Bob établissent tous les deux une communication à *anonymat fort à sens unique* vers chacun des noeuds de rendez-vous. Ils fournissent alors le numéro de session à chacun de ces noeuds. Les numéros correspondant, le noeud établit la connexion (ie: envoie les messages de l'un à l'autre).

On note que si ce système est non-observable, il est très coûteux en ressources : en bande passante d'une part, le message étant diffusé à tout le réseau, mais surtout en temps de calcul, toutes les machines tentant de déchiffrer tous les messages. C'est la principale limite à l'agrandissement d'un tel réseau, ainsi qu'une de ses faiblesses, cela laissant la place à une attaque de type déni de service.

Parmi les solutions envisageables, on peut penser à un découpage du réseau en cellules, la diffusion se faisant uniquement dans une cellule donnée, l'utilisateur indiquant au réseau dans quelle cellule il se trouve, par exemple par un mécanisme basé sur une table de hachage distribuée comme c'est le cas dans Tor (notons que Tor n'utilise que ce mécanisme, ce qui affaiblirait le

système avec nos hypothèses sur l'attaquant). L'utilisation de la cryptographie basée sur les courbes elliptiques pourrait également permettre de gagner un ordre de grandeur dans la taille de chaque cellule.

2.2 Une fois la communication établie

2.2.1 Dissimulation

Pour réduire l'observabilité, nous proposons d'une part de rendre les différentes communications indistinguables et, d'autre part, de "noyer" chaque communication dans une sorte de bruit de fond, constitué d'autres communications transitant sur le réseau et de trafic leurre.

Homogénéité des communications

Supposons pour commencer que tout le trafic sur notre réseau est constitué de vraies communications : nous voulons empêcher quelqu'un observant un routeur de pouvoir distinguer les communications (c'est-à-dire, s'il y a quatre flux, de les associer par paire en pouvant dire "ce qui entre sur ce flux ressort sur celui-ci et inversement").

Dans ce but, la première chose à faire est de rendre les paquets eux-mêmes indiscernables. Ils doivent donc tous avoir la même taille. Comme chaque routeur enlève (ou ajoute, selon le sens dans lequel le paquet circule) une couche de chiffrement, il n'y a alors pas de moyen (en ne regardant que le paquet lui-même) de faire le lien entre un paquet entrant et un paquet sortant.

Pour rendre ensuite les flux de paquets indistinguables, il est nécessaire d'une part qu'ils transmettent les paquets à une fréquence fixée. Le routeur, pour sa part, ne doit pas retransmettre immédiatement un paquet qu'il reçoit, sous peine de trahir le lien entre les flux entrants et sortants ; il doit au contraire faire les émissions selon la fréquence établie, dans un ordre indépendant de l'ordre d'arrivée des paquets.

Le problème est alors d'établir la taille et la fréquence des paquets : une latence faible demande de petits paquets envoyés souvent, tandis que l'efficacité pousse plutôt à utiliser de grands paquets envoyés à une fréquence plus faible, étant donnée que chaque paquet nécessite un certain nombre d'en-têtes, plus, le cas échéant, un mécanisme de contrôle d'intégrité...
A titre d'anecdote, pour illustrer ce problème, rappelons que lors de la mise au point du protocole ATM, qui utilise des en-têtes de 5 octets (c'est-à-dire beaucoup plus petites que dans notre cas avec IP), les informaticiens voulaient des paquets (pour ATM on parle plutôt de cellules) d'au moins

128 octets pour des raisons d'efficacité, tandis que les gens des télécoms ne voulaient pas plus de 32 octets[1]...

Asynchronisation des connexions

Quelques paragraphes plus haut, nous parlions d'une synchronisation entre les paquets, maintenant nous parlons d'asynchroniser les connexions... Il n'y a néanmoins pas de contradiction : nous entendons par là que si Eve, qui surveille notre routeur, voit disparaître simultanément deux des flux, elle pourra conclure que ces deux flux se correspondaient (Notez en passant que si Eve n'espionnait que ce noeud dans l'espoir de remonter au précédent pour découvrir finalement l'origine de la communication, elle a des chances d'être coincée : elle connaît en effet le noeud suivant, mais, ne l'ayant pas surveillé elle ne peut aller plus loin. Cependant, l'observateur puissant de notre modèle est capable de surveiller de nombreux noeuds simultanément...).

Pour parer à ce problème, l'idée que nous proposons est la suivante : lorsqu'une connexion n'est plus utilisée, au lieu de la fermer immédiatement, on la met dans un *pool* de connexions "pouvant être fermées". Lorsqu'il a besoin de bande passante, par exemple pour l'établissement d'une nouvelle communication, ou même aléatoirement, le routeur peut prendre une communication dans ce pool et la fermer.

Dynamicité

Afin d'empêcher un attaquant observant un noeud de pouvoir baser son analyse sur une grande quantité de données, mais aussi surtout pour augmenter les chances que plusieurs communications entrantes (vraies ou fausses, voir plus bas) arrivent à des temps rapprochés sur un routeur, ce qui permet de créer leurs connexions de sortie sur des intervalles de temps proches, dans un ordre aléatoire, nous proposons une grande dynamicité dans les routes utilisées pour une communication.

Des études complémentaires sont nécessaires, mais des changements au bout d'un temps aléatoire d'une durée moyenne de quelques dizaines ou quelques centaines de secondes nous semblent adaptés.

On peut également imaginer, en complément, non pas des changements de l'intégralité de la route, mais seulement d'une partie, à des intervalles plus rapprochés.

1. Afin de loger tout le monde à la même enseigne, le standard a été fixé à 48 octets...

Trafic leurre

Nous utilisons du trafic leurre à plusieurs échelles :

- Echelle des paquets : lorsque l'application qui utilise la communication ne fournit pas une quantité suffisante de données, ou lorsqu'un routeur ne reçoit pas de paquet sur un flux dans l'intervalle de temps entre deux de ses envois (que cela résulte d'une congestion du réseau ou d'une tentative de Mallory, pour voir s'il manque aussi un paquet en sortie du routeur, ce qui lui permettrait de savoir quelle est la connexion de sortie qui correspond à celle où il a enlevé un paquet), de faux paquets sont créés aléatoirement. Ils sont indistinguables des données chiffrées, aussi vont-ils se propager jusqu'au bout de la communication ; c'est l'hôte final qui les jettera voyant qu'ils ne contiennent que du charabia...
- Echelle des connexions : l'idée est d'empêcher Eve de pouvoir déterminer, lorsqu'une nouvelle connexion arrive sur un noeud, quelle connexion de sortie lui correspond. Pour ce faire, le noeud en question peut, en fonction de sa charge, créer un certain nombre de connexions leurres vers des noeuds choisis au hasard (qui eux-mêmes auront une certaine probabilité, du coup, de créer de nouvelles communications également, etc.). Si tous les noeuds en font autant, et de manière combinée avec la dynamicité proposée plus haut, il y a une certaine probabilité qu'une autre communication entre sur le même noeud dans un intervalle de temps donné. Si cet intervalle de temps est suffisamment bref, les créations de routes pour cette nouvelle connexion et pour celle qui nous intéresse peuvent être faites simultanément, gênant d'autant plus l'observateur. Il serait donc intéressant de calculer la probabilité d'une telle arrivée, en fonction du temps, de la manière dont on choisit de créer des connexions leurres et de la dynamicité mentionnée plus haut, puis d'inverser la formule pour choisir les paramètres en fonction de la probabilité que l'on désire avoir.
- Echelle des communications : ici aussi, l'idée est d'introduire du bruit sur le réseau. On imagine que, de manière aléatoire, un noeud du réseau va créer une route, puis la modifier, comme si une réelle communication se déroulait. Cela crée du trafic comme dans le cas des connexions et empêche un observateur de déterminer quand le noeud communique réellement.

2.2.2 Latence

Le problème de l'*onion-routing* est que chaque noeud sur une route ajoute un peu plus de latence. Bien que ce fait semble incontournable, nous avons

différentes idées pour améliorer cette latence. En effet, pour certains types de communications comme la téléphonie via Internet, une très faible latence est nécessaire.

Redondance des routes

L'un des points de notre architecture pair-à-pair est que les noeuds doivent être considérés comme non fiables (que cela implique ou non une intention malicieuse). Un des noeuds par lesquels passe une communication peut très bien disparaître brusquement. Aussi, pour diminuer ce risque, l'idée est qu'une communication utilise plusieurs routes différentes simultanément.

Cela a d'autres conséquences : d'une part il est possible que le fait qu'il y ait plusieurs routes établies simultanément diminue la non-observabilité offerte par notre système. Il faudrait étudier dans quelle mesure exactement. Cependant la redondance a d'autres avantages ; en particulier dans les cas où la latence est importante, il est possible d'émettre les paquets sur les différentes routes et de prendre à l'arrivée, à chaque fois, le paquet arrivé le premier. A l'inverse, dans les cas où la latence n'est pas cruciale, on peut émettre des données différentes sur les routes et ainsi augmenter le débit qui autrement est limité par les contraintes énoncées plus haut concernant la non-distinguabilité des connexions.

Autoriser la perte de certains paquets

La plupart des protocoles nécessitant une très faible latence acceptent très bien la perte de certains paquets, mais pas celle d'autres (généralement beaucoup moins nombreux) nécessaires à des fins de contrôle de la connexion ; c'est pourquoi ils utilisent à la fois une communication UDP (non fiable, pour les données à transmettre rapidement) et une communication TCP (fiable, pour les données de contrôle). On peut citer comme exemple la plupart des protocoles de VoIP, mais sans s'y limiter : le système d'affichage distribué Chromium fonctionne également ainsi.

Le problème d'une communication fiable via TCP est que si un paquet est perdu, il faut attendre que la perte soit détectée et que le paquet soit retransmis. Pendant ce temps, l'application qui utilise cette connexion ne reçoit rien et est obligée d'attendre. C'est pourquoi ces communications ne sont pas adaptées à la transmission de la voix par exemple.

Il ne nous est pas possible de séparer les données et les informations de contrôle sur une communication non-observable (sauf à augmenter démesurément le trafic), nous ne pouvons donc avoir qu'un seul type de communication. Une solution à ce problème serait de créer un protocole au-dessus d'UDP

(donc rapide) qui permette à l'émetteur de spécifier pour chaque message si les données qu'il contient doivent absolument arriver ou non. Il y aurait ainsi un contrôle des pertes de bout en bout et non entre chaque noeud.

Routage

Une autre possibilité, qui nécessite toutefois de prudentes études préalables, serait l'introduction d'une méthode de routage pour que la création des routes ne soit pas aléatoire mais suive plus ou moins la topologie des liens du réseau sous-jacent. Il faut néanmoins faire attention ici étant donné qu'un noeud malicieux pourrait mentir dans le but de faire passer le trafic par ses complices.

Les différentes méthodes décrites ci-dessus devraient nous offrir un système présentant de bonnes propriétés de non-observabilité, au détriment de l'efficacité en termes de rapport données utiles / occupation de bande passante. Notons en passant que la non-observabilité nous donne implicitement la mobilité. Le chapitre suivant décrit l'architecture du début d'implémentation que nous avons effectué.

Chapitre 3

Praxis

> Good ideas are not adopted automatically. They must be driven into practice with courageous impatience.
>
> — Amiral Hyman Rickover

Nous avons implémenté ces idées par, d'une part, un démon Unix qui gère le réseau et, d'autre par, une bibliothèque de fonctions utilisables par les clients pour utiliser le réseau.

Le nom actuellement donné pour identifier cette implémentation est *True Nyms*[1]. Elle fait actuellement un peu plus de 5500 lignes de code C. Ce code source est disponible sur le cd-rom joint.

3.1 Bibliothèque de fonctions

Il y a encore relativement peu de choses à dire sur la bibliothèque de fonctions, l'API qu'elle offre n'étant pas encore totalement stabilisée. Parmi les fonctions existantes et qui devraient rester (leurs paramètres, eux, changeront probablement, aussi ne sont-ils pas indiqués ici) notons qu'il y a une séparation entre les fonctions fournissant simplement de l'anonymat, et les fonctions concernant les nymes, avec, à chaque fois, une fonction permettant d'établir une communication (`anon_connect` et `nyms_connect`), une fonction permettant d'attendre une communication (`anon_listen` et `nyms_listen`), et des fonctions permettant les opérations de lecture / écriture (`anon_read`, `nyms_read`,`anon_write` et `nyms_write`). Le lecteur intéressé pourra se reporter aux fichiers `anon.h` et `nyms.h`.

Actuellement, un nym est constitué d'une clef RSA. Ils peuvent être stockés au format PEM (et créés avec `openssl genrsa`).

1. Jusqu'à ce qu'un meilleur soit éventuellement trouvé...

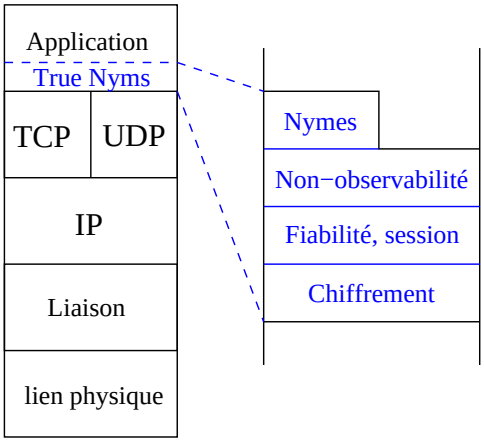


FIG. 3.1 – *Intégration de* True Nyms *dans le modèle TCP/IP.*

Ces fonctions se connectent au démon local qui tourne en arrière-plan et l'utilisent pour insérer les communications sur le réseau.

3.2 Le démon *nymsd*

Au lancement du démon, plusieurs processus sont créés : le processus principal est chargé d'attendre les communications entrantes ; un second processus est chargé de la partie concernant la synchronisation des écritures, tandis qu'un troisième gère les messages d'établissement de communication et ce que l'on pourrait qualifier "d'infrastructure" du réseau pair-à-pair utilisé pour leur diffusion.

3.2.1 Gestion des communications

Le démon consiste donc en un processus principal qui attend les communications entrantes. De manière classique ce processus fait un `fork(2)` lorsqu'une nouvelle communication arrive, le nouveau processus gérant celle-ci. Selon ce qui est demandé par l'hôte distant, ce processus peut soit, dans certains cas (demande d'authentification, etc), tout faire lui-même, soit faire appel à des ressources externes : la liste des hôtes, établie par le processus de gestion des messages d'établissement de session qui se trouve en mémoire partagée, lorqu'on lui demande une telle liste, ledit processus de gestion des messages d'établissement lorsqu'il s'agit d'émettre un nouveau de ces messages, ou le processus `writer` lorsqu'une nouvelle connexion sortante est créée.

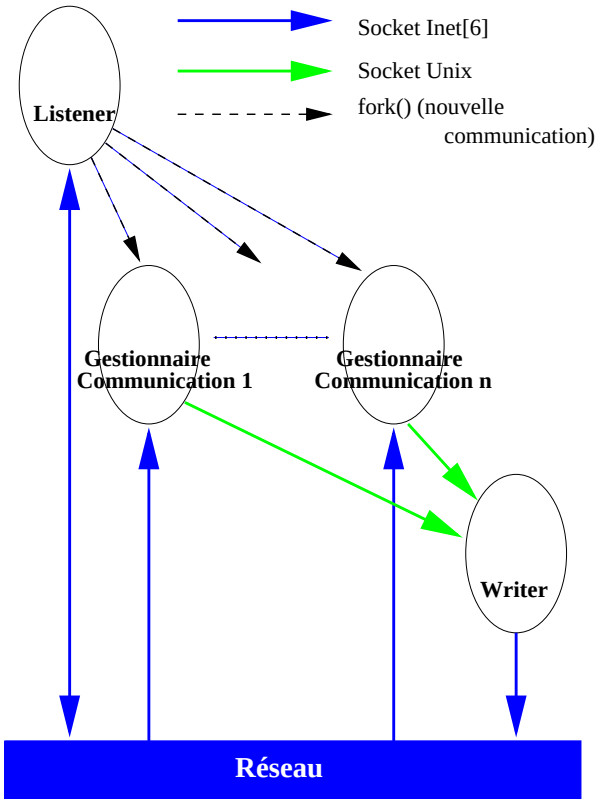


FIG. 3.2 – *Chemin des communications dans* nymsd.

Synchronisation des écritures

Un point qui nous semble intéressant sur un plan technique dans notre implémentation est celui de la synchronisation entre les processus concernant les écritures sur le réseau : afin d'éviter une attaque par analyse de trafic entrant et sortant triviale sur un noeud, celles-ci doivent en effet d'une part, avoir la même fréquence (émission de message leurre s'il n'y a pas de données), d'autre part elles ne doivent pas refléter l'ordre d'arrivée des messages. Si ce premier point est facile à résoudre – il suffirait que chaque processus gérant une communication écrive avec une certaine fréquence qui serait la même pour tous les processus – le second l'est moins : en effet une solution naïve telle celle que nous venons de décrire ferait que lorsqu'une nouvelle connexion serait créée, il serait facile de savoir quelle est la *vraie* communication de sortie (on suppose ici que le routeur crée de fausses communications pour dissimuler cette information) simplement en regardant quelle est l'émission qui se fait tant de temps après l'initiation de cette nouvelle connexion.

Nous avons choisi de résoudre ce problème en chargeant un unique processus de faire toutes les écritures : ainsi, les écritures sont synchronisées (ou plus exactement séquentielles avec une fréquence fixée, ce qui est suffisant), et on change aléatoirement l'ordre dans lequel elles se font à la réception d'une nouvelle connexion, empêchant ainsi de déterminer la vraie connexion de sortie correspondante.

Le problème d'une telle architecture est que l'on a donc un processus `writer` qui doit connaître les sockets sur lesquelles il écrit, alors que ces sockets sont créées par un autre processus après la création de `writer` lui-même... Il faut donc transmettre une socket entre deux processus. Il existe une solution peu connue à ce problème, qui gagnerait à l'être car elle pourrait simplifier un grand nombre de programmes existants : il est possible de transmettre un descripteur de fichier (ou, comme ici, de socket) via les messages de contrôle des sockets du domaine Unix. Le lecteur intéressé pourra se référer aux pages man linux `socket(7)` et `cmsg(3)`, ainsi qu'à [18] concernant le fonctionnement de ce système au niveau du noyau[2].

3.2.2 Gestion des messages d'établissement de session

Le processus `caller` se charge des parties consistant à connaître le réseau d'hôtes et de la gestion des messages d'établissement de session.

La connaissance du réseau se fait principalement en demandant au démarrage à des hôtes connus (fichier de configuration) une liste de noeuds participants au réseau pair-à-pair. Le processus se connecte alors à quelques-uns de ces noeuds et leur demande de lui transmettre tous les messages d'établissement de session qu'ils reçoivent. Lui-même retransmet les messages qu'il reçoit (soit par un des noeuds auquel il s'est connecté soit par un client qui lui a demandé un service similaire par la suite) aux autres noeuds.

Lorsqu'un client vient s'enregistrer, il est ajouté à la liste des hôtes connus (qui est en mémoire partagée pour que les autres processus constituant le démon puissent l'utiliser), qui pourra être utilisée soit pour établir des communications leurres soit transmise (partiellement) à des machines qui en font la demande.

Dans la version actuelle, les messages d'établissement de communication sont composés d'une partie qui contient les indications sur les points de rendez-vous, chiffrée avec Blowfish, et d'une partie contenant la clef Blowfish permettant de déchiffrer la précédente, chiffrée avec RSA. On notera ici

2. En fait c'est un mécanisme similaire à la notion de *capacité* que l'on croise parfois en sécurité informatique : les fichiers (ou sockets dans notre cas) ouverts sont enregistrés dans le noyau du système d'exploitation. On transmet en fait une autorisation d'y accéder à l'autre processus.

que l'utilisation naîve de RSA est vulnérable à des attaques (RSA est un homomorphisme multiplicatif), aussi utilisons-nous le padding PKCS-OAEP tel que défini dans [16] et qui est succinctement rappelé dans l'annexe A.

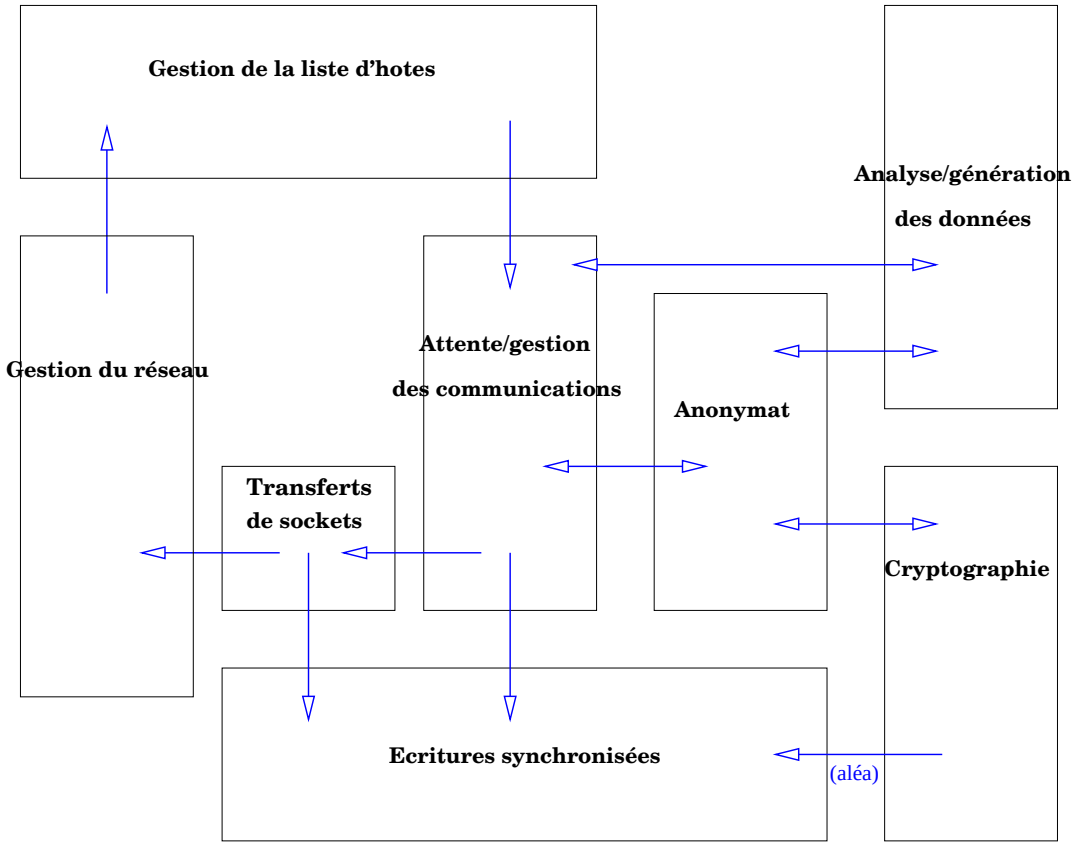


FIG. 3.3 – *Principales parties de* nymsd *et sens des données dans les échanges internes.*

3.3 Trivia

Sûreté et sécurité: au cours du développement de ce programme, une attention particulière a été portée à la sécurité : en effet, comme il s'agit d'un démon réseau, il doit résister à des utilisateurs malicieux tentant de prendre le contrôle du système sur lequel il tourne. [21] nous a donné quelques précieux conseils. Le démon peut être facilement installé dans une cage `chroot` pour améliorer la sécurité du système.

Concernant la sécurité du protocole lui-même, [1] nous a permis d'éviter

de nombreux écueils que l'on peut rencontrer en utilisant des primitives cryptographiques, comme celui de l'utilisation de RSA pour les messages d'établissement de session.

Il faut toutefois noter que le programme est résolument expérimental et qu'il y a de nombreux points – dont nous sommes conscient – qui devront être complétés ou réécrits (certaines choses ayant été laissées en attente pour une question de temps) pour avoir une sécurité correcte.

Modularité et extensibilité : comme nous venons de le dire, notre implémentation étant résolument expérimentale, nous avons tenté de garder une grande modularité afin de pouvoir, le cas échéant, en modifier une partie de manière aussi transparente que possible pour le reste du programme. Pour ce faire, nous nous sommes basé sur une "philosophie" consistant à utiliser la souplesse du langage C pour programmer d'une manière s'approchant de la programmation orientée objets (voire de la programmation orientée composants!) là où cela nous semblait nécessaire.Ces idées s'inspirent en partie de [14] et surtout de [2].

Dans le même ordre d'idées, nous avons essayé de garder quand c'était possible un certain niveau d'abstraction vis-à-vis des dépendances du programme ; ainsi nous utilisons pour la partie cryptographique la bibliothèque OpenSSL [13, 19], mais nous avons fait en sorte que son remplacement (par exemple par la Cryptlib de Peter Gutmann, ou la récente libgcrypt) soit aisé en regroupant les appels à ses fonctions dans un fichier wrapper.

Gestion du code source : l'outil *GNU-Arch* a été utilisé pour la gestion des révisions code source. `Doxygen` a servi pour introduire des commentaires aisément extractibles, que ce soit pour documenter l'API interne ou pour se souvenir de détails restant à écrire.

Portabilité : notre démon fonctionne pour l'instant sur Linux. Nous ne l'avons pas encore testé sur d'autres systèmes Unix, mais, étant donné que nous avons tenté de respecter le standard Posix, il ne devrait pas y avoir de problème particulier. Notons que nous avons utilisé le standard ISO C99, il faut donc un compilateur compatible, tel que gcc version 3.

Conclusion et perspectives

> We can only see a short distance ahead, but we can see plenty there that needs to be done.
>
> — Alan M. Turing, Computing Machinery and Intelligence, *Mind*, 1950.

Nous avons donc un certain nombre de propositions pour obtenir un système qui permette des communications bénéficiant d'une faible latence non-observables, c'est-à-dire pour lesquelles il n'est pas possible de déterminer si un utilisateur communique ou non, ni avec qui, au détriment toutefois de l'efficacité en termes de bande passante.

Nous avons débuté une implémentation de ces idées, qui devrait permettre, via des tests, de les trier selon leur adéquation à notre problème et de trouver des paramètres donnant les meilleurs résultats dans le cas qui nous intéresse.

Comme nous l'avons indiqué dans le texte, un certain nombre de ces idées peuvent être développées. D'autres problèmes nécessitent une réflexion approfondie, notamment celui de la résistance aux attaques de type déni de service. Nous espérons donc poursuivre ce travail en thèse.

Annexe A

PKCS-OAEP

Nous indiquons simplement ici la manière dont se fait l'encodage selon le padding PKCS-OAEP, tel que défini dans [16], document auquel le lecteur désirant plus de détails est renvoyé. Une fois un message ainsi encodé, il peut être chiffré par RSA. L'utilisation de RSA sans un tel encodage préalable peut être vulnérable face à des *attaques à chiffrés choisis*.

Dans la version actuelle de *True Nyms*, ces opérations sont déléguées à la bibliothèque `OpenSSL`.

Définitions:

- $Hash$ est une fonction de hachage, généralement SHA-1. $hLen$ est la longueur de l'empreinte générée par cette fonction (20 octets dans le cas de SHA-1) ;
- MGF une fonction de génération de masques (la seule fonction actuellement définie dans le standard, MGF1, est basée sur le hachage multiple de la graine) ;
- k est la longueur **en octets** du module RSA. Dans la version actuelle de *True Nyms*, qui utilise des clefs RSA de 1024 bits, on a $k = 128$;
- M est le message à encoder (et qui sera chiffré ensuite...), de longueur $mLen$. On a la condition $mLen \leq k - 2hLen - 2$;
- L est une étiquette que l'on peut optionnellement associer avec le message ;

Algorithme d'encodage :

1. Si l'étiquette L n'est pas fournie, considérons qu'il s'agit de la chaîne vide ; soit $lHash = Hash(L)$ une chaîne d'octets de longueur $hLen$.
2. Soit PS une chaîne d'octets consistant de $k - mLen - 2hLen - 2$ octets nuls. (La longueur de PS peut être nulle.)

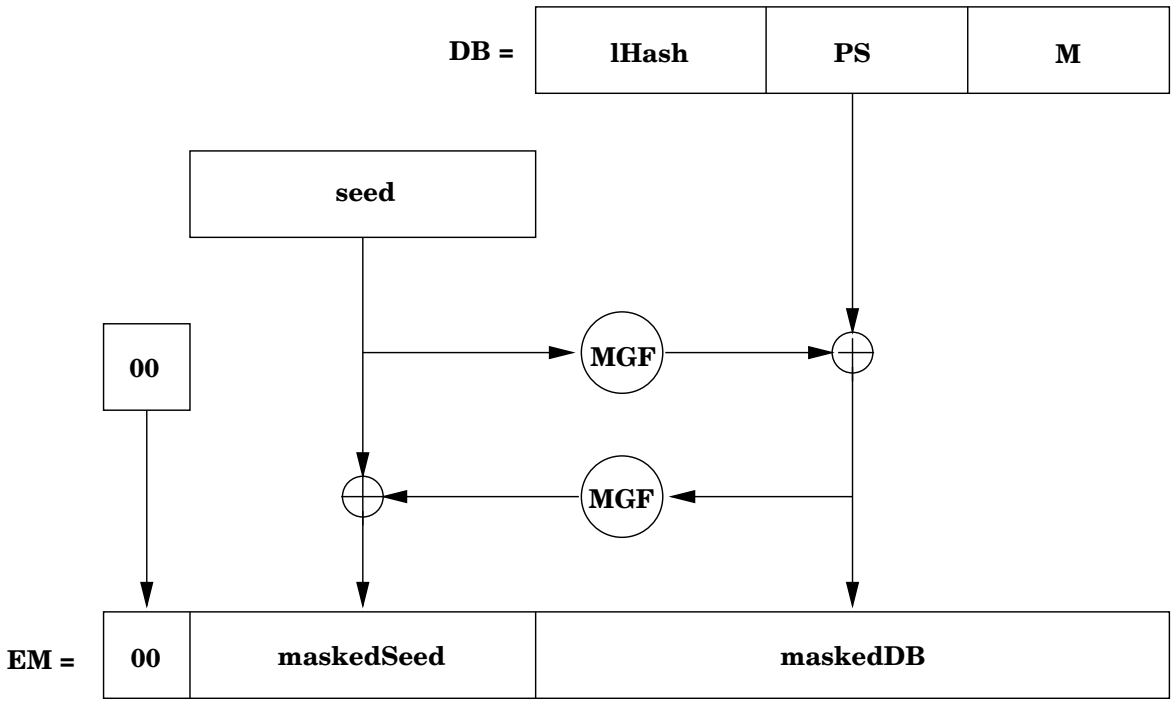


FIG. A.1 – *Encodage d'un bloc de données M selon le padding* pkcs-oaep.

3. Concaténons $lHash$, PS, un octet valant un, et le message M, formant ainsi un bloc de données DB de longueur $k - hLen - 1$ octets :

$$DB \;=\; lHash \;||\; PS \;||\; 0x01 \;||\; M$$

4. Soit une chaîne d'octets aléatoire *seed* de longueur $hLen$.
5. Soit $dbMask = MGF(seed,\, k - hLen - 1)$.
6. Soit $maskedDB = DB \;\oplus\; dbMask$.
7. Soit $seedMask = MGF(maskedDB,\, hLen)$.
8. Soit $maskedSeed = seed \;\oplus\; seedMask$.
9. Concaténons un octet nul, avec $maskedSeed$ et $maskedDB$ pour former un message encodé EM de k octets :

$$EM \;=\; 0x00 \;||\; maskedSeed \;||\; maskedDB$$

Annexe B

Noms standards

Nous rappelons brièvement ici "qui" sont les différents noms standards en cryptographie que nous utilisons dans le texte :

Alice est traditionnellement une utilisatrice du système. Elle désire simplement communiquer avec un autre utilisateur, généralement Bob. Sauf mention spéciale, Alice agit de manière non malicieuse.

Bob est un autre utilisateur, très similaire à Alice. Il est parfois appelé Bernard dans les ouvrages français.

Eve est l'espionne : elle peut écouter le réseau pour obtenir des informations. Contrairement à Mallory, Eve est uniquement passive[1], elle n'essaie pas d'envoyer de faux messages pour tenter de casser le système. Les attaques d'Eve sont difficiles à détecter, aussi un protocole sûr doit-il se prémunir contre elles. Elle est parfois appelée Estelle dans les ouvrages français.

Mallory parfois appelé également Mallet, est un attaquant actif. Il peut prétendre être quelqu'un d'autre, ajouter ou supprimer des messages, couper une connexion, etc. Il est donc beaucoup plus puissant qu'Eve au niveau des attaques qu'il peut faire. Notons que Mallory peut être parallèlement un utilisateur légitime du système qu'il essaie de casser.

1. Vis-à-vis de la cible ! Eve peut néanmoins déployer d'importantes ressources personnelles ailleurs si celles-ci peuvent lui permettre d'obtenir des informations à partir de ce qu'elle a vu passer.

Bibliographie

[1] Ross Anderson. *Security Engineering*. Wiley Computer Publishing. John Wiley & Sons, Inc, 2001.

[2] Olivier Aumage. Correspondance privée. Echange d'emails sur le thème de Ibis, PM2, Java, la philosophie Unix et les avantages et inconvénients de la programmation orientée objet, mars 2003.

[3] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, Février 1981.

[4] Lance Cottrell. Mixmaster & remailer attacks. `http://www.obscura.com/loki/remailer/remailer-essay.html`.

[5] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor : The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004. A paraître.

[6] Michael J. Freedman and Robert Morris. Tarzan : A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM, 2002.

[7] James Frenkel, editor. *True Names and the opening of the cyberspace frontier*. Tor, 2001.

[8] Ken Guggenheim. Pentagon abandons terrorism betting plan. Dépèche Associated Press, Juillet 2003.

[9] David Kahn. *The Codebreakers*. Scribner, troisième edition, 1996.

[10] Peter Ludlow, editor. *Crypto Anarchy, Cyberstates, and Pirate Utopias*. Digital Communications. MIT Press, 2001.

[11] 2003 wiretap report. Technical report, Administrative Office of the United States Courts, Leonidas Ralph Mecham (dir), mai 2004. `http://www.uscourts.gov/wiretap03/contents.html`.

[12] Ulf Moeller and Len Sassaman Lance Cottrell, Peter Palfrader. Mixmaster protocol version 2. Internet-draft, Mai 2004. Accessible à `http://www.ietf.org/internet-drafts/`

`draft-sassaman-mixmaster-01.txt` jusqu'en novembre 2004 ou à la parution d'une révision.

[13] The OpenSSL Project. Site web. `http://www.openssl.org`

[14] Eric Steven Raymond. *The Art or Unix Programming*. Addison-Wesley, 2003.

[15] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, Mai 1998.

[16] RSA Laboratories. *PKCS #1 v2.1: RSA Cryptography Standard*. RSA Security, 2002. La RFC 3447, écrite par Jonsson et Kaliski et daté de février 2003 semble être exactement le même document.

[17] Bruce Schneier. *Cryptographie appliquée*. International Thompson Publishing, deuxième edition, 1996.

[18] W. Richard Stevens. *TCP/IP Illustrated*, volume 3. Addison-Wesley, 1996.

[19] John Viega, Matt Messier, and Pravir Chandra. *Network Security with OpenSSL*. O'Reilly and Associates, 2002.

[20] Vernor Vinge. *True Names*. In Frenkel [7], 1981.

[21] David A. Wheeler. *Secure Programming for Linux and Unix HOWTO*, 3.010 edition, Mars 2003. `http://www.dwheeler.com/secure-programs`.