Vérification de protocoles cryptographiques: la logique à la rescousse!

Jean Goubault-Larrecq[1]

1: LSV/CNRS UMR 8643 et ENS Cachan,
61, avenue du président-Wilson,
94235 Cachan CEDEX, France
goubault@lsv.ens-cachan.fr

Résumé

La vérification de propriétés de sécurité de protocoles cryptographiques est une activité délicate et importante. Après avoir passé en revue les principes de base de cette activité, nous montrerons que la logique du premier ordre permet à la fois une formalisation adéquate des protocoles cryptographiques et de certaines de leurs propriétés les plus courantes (secret, authentication), et se prête bien à des mécanismes d'abstraction *automatique* qui en permettent une vérification automatique aisée. Le lien entre logique du premier ordre et automates d'arbre y est ici fondamental.

1. Introduction

La sécurité des systèmes d'information est aujourd'hui un domaine en pleine effervescence, voire à la mode. Parmi les différents aspects de la sécurité informatique, la cryptologie et l'étude des protocoles cryptographiques a pris une importance considérable. La cryptologie est originellement l'étude des moyens permettant de transmettre des messages secrets, et remonte à l'antiquité. Depuis, la portée des méthodes cryptologiques s'est considérablement étendue, pour inclure des préoccupations de garantie d'authenticité et d'intégrité des messages reçus par exemple. Mais aussi, comme la sécurité des messages repose sur l'usage de clés de chiffrement ou de signature, la cryptologie a dû aussi traiter des problèmes de distribution sécurisée de clés à des tiers, de révocation de clés supposées peu fiables, de récupération de clés perdues par des utilisateurs réputés honnêtes, etc.

La cryptologie est donc un domaine riche [37], mais qui s'est longtemps limité essentiellement aux applications militaires. On peut considérer que le surgissement d'intérêt pour la cryptographie dans les années 1990 et 2000 est dû au développement du commerce électronique. Ce dernier terme recouvre pour commencer les échanges chiffrés d'information qui ont lieu lorsque l'on retire de l'argent dans un distributeur de billets à l'aide d'une carte bancaire, où l'ordinateur qu'est le distributeur de billets s'engage dans un dialogue avec la carte du client et possiblement avec un serveur de la banque pour vérifier que l'utilisateur de la carte est honnête, dispose de la somme demandée sur son compte, et ne pourra pas récupérer les billets demandés sans que son compte en soit débité, notamment. Le commerce électronique recouvre aussi, par exemple, les paiements par carte à l'aide de terminaux portables, dans les supermarchés ou les restaurants, où l'utilisateur doit confirmer son identité en entrant son code PIN à quatre chiffres. Toutes ces applications demandent des garanties de sécurité élevées, portant sur des propriétés de secret et d'authenticité comme ci-dessus, mais aussi de nombreuses autres propriétés, parmi lesquelles, la non-duplication (des factures, par exemple, dans l'intérêt du client), de non-révocation (des commandes, dans l'intérêt du commerçant), d'anonymat (dans le cadre des micro-paiements, où l'on souhaite offrir au client l'équivalent électronique du billet de banque, qui ne porte pas le nom de son utilisateur, contrairement au chèque par exemple), etc. La cryptologie aurait cependant pu rester un domaine relativement peu connu du grand public sans l'Internet, le World-Wide Web et en particulier le protocole SSL introduit par Netscape et son successeur TLS [16]. En 2000, il est possible de faire des achats depuis un browser Web, qui négocie un protocole garantissant un certain niveau de secret et d'authentification avec le serveur d'un commerçant, et l'utilise ensuite pour chiffrer les négociations de produits et de tarifs conduites par l'utilisateur du browser avec le commerçant.

Or il se trouve que si, depuis les années 1980, on dispose d'algorithmes cryptographiques suffisamment sûrs, il n'en est pas de même, loin s'en faut, des protocoles cryptographiques. Nous le montrons au travers d'un exemple en section 2, et y illustrons les particularités de la sécurité cryptographique. Nous montrons ensuite comment on peut modéliser et soit vérifier des protocoles cryptographiques, soit détecter des attaques, en section 3. Nous serons ici relativement partiaux, et nous contenterons de présenter une variante d'une approche due à Dolev et Yao, qui a le mérite d'être simple, de se prêter à l'automatisation, et de se concentrer sur les problèmes de sécurité principaux. Nous présenterons cependant rapidement en section 3.4 quelques autres approches intéressantes, notamment pour leur rigueur et leur précision, comme celles fondées sur les algèbres de processus et notamment le spi-calcul, ou l'étude des protocoles au travers de méthodes de théorie de la complexité probabiliste. Nous concluons en section 4.

2. Un exemple de protocole cryptographique

2.1. Premières tentatives

Tentons d'imaginer comment permettre à deux principaux (ordinateurs, cartes à puce, éventuellement aussi êtres humains) A et B d'obtenir une clé K_{ab} qu'eux seuls connaissent, et qui leur permettra de chiffrer les messages qu'ils désirent échanger par la suite. Une solution simple est de faire créer K_{ab} par A, qui l'envoie en clair à B. Ceci a cependant de nombreux défauts. D'abord, la clé K_{ab} ne peut pas être considérée secrète : il est facile pour un intrus I d'espionner la ligne de communication entre A et B et de récupérer K_{ab}. Ensuite, la clé K_{ab} ne peut pas non plus être considérée comme authentique par B : il est aussi facile pour I de couper la ligne de communication entre A et B et de la rediriger de I vers B ; toute clé K_I envoyée par I à B sera non authentique.

Supposons donc que A et B ont déjà en commun une clé K_{long}, dite à long terme, et partagée uniquement entre eux deux. A peut alors créer la clé K_{ab} comme précédemment, mais au lieu de l'envoyer en clair, il la chiffre d'abord avec la clé K_{long}, au moyen d'un algorithme de chiffrement sûr (quel qu'il soit), obtenant ainsi le message chiffré $\{K_{ab}\}_{K_{long}}$. A envoie ensuite $\{K_{ab}\}_{K_{long}}$ à B. Si l'algorithme de chiffrement est suffisamment sûr, la seule façon pour un intrus de récupérer K_{ab} à partir de $\{K_{ab}\}_{K_{long}}$ est de connaître la clé K_{long} pour déchiffrer $\{K_{ab}\}_{K_{long}}$. Si K_{long} est maintenue secrète par A et B, l'intrus ne pourra donc pas récupérer la valeur de K_{ab} par l'espionnage seul des lignes de communication. Si de plus le chiffrement inclut suffisamment de redondance dans le message chiffré pour que le déchiffrement de $\{K_{ab}\}_{K_{long}}$ à l'aide d'une clé autre que K_{long} échoue, l'intrus ne pourra pas non plus passer un message de la forme $\{K_I\}_{K_{long}}$ à B, espérant que B utilisera une clé K_I qu'il aurait concoctée spécialement pour que B l'utilise sans le savoir à la place de la clé légitime K_{ab}. En effet, l'intrus ne sait pas chiffrer avec K_{long}, ne disposant pas de cette dernière.

Si ce protocole — car c'en est un, malgré sa simplicité — semble donc offrir de meilleures garanties de sécurité, grâce à l'utilisation de moyens de chiffrement, il ne semble par contre pas très utile. Le lecteur attentif aura sans doute remarqué qu'en principe ce protocole ne résout rien : pour obtenir une clé K_{ab} entre A et B, il faut déjà avoir une clé K_{long} entre A et B. En réalité, K_{long} est typiquement une clé extrêmement sûre, par exemple une clé à 1024 bits à utiliser avec l'algorithme RSA [37]. Mais cet avantage a un prix : chiffrer et déchiffrer avec K_{long} peut prendre de l'ordre de 1000 fois plus longtemps qu'avec un algorithme comme DES avec des clés de longueur effective 56 bits [37]. On préfère donc utiliser pendant peu de temps des clés relativement courtes, et renouveler ces clés régulièrement en utilisant la clé plus sûre K_{long} par un protocole comme celui ci-dessus. C'est pourquoi K_{long} est appelée une clé à long terme et K_{ab} est une clé de session. La distribution de la clé K_{long} elle-même peut être effectuée à l'aide du protocole de Diffie-Hellman [17], par exemple, qui repose sur des considérations de théorie des nombres.

En fait, ce protocole souffre de problèmes plus graves. Même s'il semble être sûr grâce à l'utilisation de méthodes de chiffrement, il n'en est rien. Rappelons qu'une clé de session est bien moins sûre qu'une clé à long terme comme K_{long}. Il est donc prudent de supposer qu'une clé de session K_0 utilisée il y a suffisamment longtemps a pu rentrer en la possession d'un intrus, que ce soit parce que l'intrus a eu tout le temps de trouver la clé K_0 à partir des différents messages chiffrés $\{M\}_{K_0}$ échangés entre A et B, par exemple par exploration brute de l'espace de toutes les clés possibles ; ou bien plus simplement parce que A ou B a de façon plus ou moins involontaire diffusé K_0 en clair après utilisation. L'intrus, qui avait d'autre part récupéré le message $\{K_0\}_{K_{long}}$ par espionnage des lignes de communication, peut maintenant passer $\{K_0\}_{K_{long}}$ à B, sachant que B va prendre la vieille clé K_0 pour la clé attendue K_{ab}. Tout message M que B chiffrera avec cette clé sera envoyé sous la forme $\{M\}_{K_0}$ et non $\{M\}_{K_{ab}}$, ce qui permettra à l'intrus de les déchiffrer et de récupérer M. Une telle attaque est appelée attaque par rejeu, car pour tromper B l'intrus a rejoué un message, $\{K_0\}_{K_{long}}$, d'une ancienne session du même protocole entre A et B.

Pour empêcher les attaques par rejeu, on complique le protocole pour inclure des garanties de fraîcheur dans les messages, autrement dit des données additionnelles qui permettent aux principaux de vérifier que les messages reçus sont récents, et donc pas des versions rejouées d'anciens messages. Une solution est d'apparier les messages avec une estampille temporelle T : A lit l'heure T à partir d'une horloge locale, et envoie le couple K_{ab}, T chiffré avec K_{long}, soit $\{K_{ab}, T\}_{K_{long}}$ à B, et B vérifie que le champ T du couple déchiffré est une heure suffisamment récente. Les estampilles temporelles sont d'un maniement délicat, la notion de "suffisamment récent" étant relativement vague pour commencer ; d'autre part pour que le mécanisme fonctionne, il faut encore que les horloges des différents principaux soient synchronisées, et ce par un protocole lui-même sécurisé, pour éviter qu'un intrus n'attaque le système en désynchronisant les horloges dans le but d'effectuer de nouveau des attaques par rejeu.

Une solution plus simple est l'utilisation de nonces. B, qui maintenant se méfie des attaques par rejeu, va initier la session et tirant un nombre aléatoire N_b (le *nonce*), et va l'envoyer en clair à A. A va ensuite créer la clé K_{ab}, et envoyer le message $\{K_{ab}, N_b\}_{K_{long}}$ à B. À réception de ce message, B va vérifier que le message se déchiffre correctement avec K_{long}, puis qu'il obtient ainsi un couple dont la deuxième composante est exactement le nonce N_b. Si tout s'est bien passé, B conclut que K_{ab} est une bonne clé récente pour communiquer avec A. En effet, avec très forte probabilité (et surtout si N_b est un nombre sur une grande largeur de bits, disons 128), le nombre N_b n'a jamais circulé que ce soit en clair ou chiffré sur les lignes de communication : N_b est une valeur fraîche. Si l'intrus avait pu construire $\{K_{ab}, N_b\}_{K_{long}}$, il n'avait donc pas pu le construire avant que N_b ne soit inventé par B, et la vérification par B que N_b est bien la deuxième composante du couple chiffré, sous l'hypothèse que la clé à long terme K_{long} est toujours inconnue de l'intrus, permet donc d'assurer que la paire K_{ab}, N_b, et donc la clé K_{ab} est bien récente et ne vient pas d'un message rejoué. On note en abrégé ce protocole :

1.	B $\longrightarrow$ A : N_b
2.	A $\longrightarrow$ B : $\{K_{ab}, N_b\}_{K_{long}}$

Pour autant, ce protocole est-il sûr ? Rien ne permet de l'affirmer. En fait, l'intrus pourrait jouer le rôle de A tout du long, et créer lui-même la clé K_{ab}. Les protocoles utilisés en réalité sont plus complexes, et utilisent en général un troisième principal de confiance. Ici, la solution classique est de faire intervenir un serveur de clés S, dont le rôle est de fabriquer une clé K_{ab} à la demande pour A et B, et de la leur communiquer dans une série de messages chiffrés avec des clés à long terme K_{as} et K_{bs} respectivement partagées entre A et S d'une part, entre B et S d'autre part.

2.2. Un protocole cryptographique réaliste

Comme exemple de cette famille plus réaliste de protocoles impliquant un tiers de confiance S, considérons le protocole suivant, dû à Needham et Schroeder [32]. Le but est que seuls A et B (et S, en toute rigueur) connaissent K_{ab} à la fin du protocole. Voici ce que font A, B et S dans ce protocole (voir aussi la figure 1).

1. A $\longrightarrow$ S : A, B, N_a
2. S $\longrightarrow$ A : $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. A $\longrightarrow$ B : $\{K_{ab}, A\}_{K_{bs}}$
4. B $\longrightarrow$ A : $\{N_b\}_{K_{ab}}$
5. A $\longrightarrow$ B : $\{N_b + 1\}_{K_{ab}}$

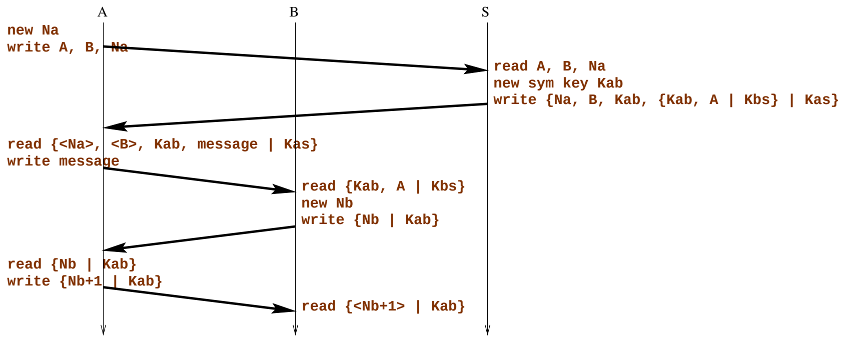

FIG. 1 – Le protocole de Needham-Schroeder à clés secrètes

En clair, voici comment se déroule une session du protocole. D'abord (message 1 ci-dessus, commandes `new Na` et `write A, B, Na` en haut à gauche de la figure 1, qui sont écrites dans le langage de l'outil CPV [23]), A envoie à S un message composé de sa propre identité, que l'on note A, de l'identité B de B, et d'un nonce N_a. (La création de ce dernier est effectuée par l'instruction `new Na` de la figure 1.) Les identités sont des identificateurs des principaux, par exemple des numéros de téléphone, des numéros de compte ou de carte bancaire, ou des URL. Ici, les identités de A et de B serviront au serveur S à retrouver les clés à long terme K_{as} et K_{bs} à l'étape 2. S les retrouve typiquement dans une base de données, à partir des identités de A et de B. Il est tentant de croire que les identités des principaux pourraient servir à authentifier les messages, mais ce n'est pas suffisant en général : il est facile de retrouver l'identité de n'importe qui, et n'importe quel intrus peut alors l'utiliser pour se faire passer pour n'importe qui.

Lorsque le message de l'étape 1 arrive à son destinataire S (`read A, B, Na` en figure 1), S renvoie le message $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$ à A. C'est un message un peu plus compliqué, qui a pour premier but d'envoyer la clé K_{ab} nouvellement créée (`new sym key Kab` en figure 1) à A et à B sous pli confidentiel. Pour ceci, S fabrique la paire consistant en K_{ab} et A, qu'il chiffre ensuite avec la clé K_{bs}, obtenant le message qu'on a noté $\{K_{ab}, A\}_{K_{bs}}$. Bien que ce message fasse partie d'un message que S envoie à A, A ne pourra pas accéder à K_{ab} en le déchiffrant, car seuls B et S connaissent la clé K_{bs} par hypothèse ; en fait, le message $\{K_{ab}, A\}_{K_{bs}}$ ne sert qu'à être récupéré par A pour le renvoyer à B à l'étape 3. C'est à l'étape 3 que B déchiffrera ce message et en retirera la valeur de K_{ab}. En attendant, S apparie ce message $\{K_{ab}, A\}_{K_{bs}}$ avec le nonce N_a envoyé par A, l'identité de B, et de nouveau la clé K_{ab}. (Le chiffrement d'un message M par la clé K, usuellement noté $\{M\}_K$, est `{M | K}` dans la notation de l'outil CPV [23].)

Revenons à l'étape 2 du protocole. A reçoit maintenant le message $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$, il utilise sa copie de la clé K_{as} pour le déchiffrer et ainsi récupérer le nonce N_a, la valeur de l'identité de B, la clé K_{ab} nouvellement créée par S, ainsi qu'un message qui est censé être $\{K_{ab}, A\}_{K_{bs}}$ et que A enverra à B à l'étape suivante. La figure 1 est plus précise quant à ce que A fait à cette étape (`read {<Na>, <B>, Kab, message | Kas}`) : A vérifie que la première composante du message déchiffré est exactement (la notation `<.>`) le nonce N_a que A avait créé en étape 1. Cette vérification assure A que le message $\{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$ est frais, et donc que la clé K_{ab} est elle-même fraîche.

A envoie ensuite à l'étape 3 le sous-message (`message` dans la figure 1) qu'il n'avait pas pu déchiffrer, et qui est censé être $\{K_{ab}, A\}_{K_{bs}}$. B le reçoit, obtient l'identité de A et la clé K_{ab}. Il s'engage ensuite dans deux étapes de protocole supplémentaires, 4. et 5., pour s'assurer qu'il possède la même clé K_{ab} que A. Pour ceci, B crée un nouveau nonce N_b, qu'il envoie chiffré par la clé K_{ab} à A. Si A est d'accord sur la valeur de K_{ab}, il pourra déchiffrer $\{N_b\}_{K_{ab}}$, fabriquer $N_b + 1$, le rechiffrer avec K_{ab} et le renvoyer à B. Noter qu'on pourrait modifier le protocole de sorte que A renvoie n'importe quel autre message fabriqué à partir de N_b et différent de N_b. Il ne peut pas renvoyer $\{N_b\}_{K_{ab}}$, sinon l'intrus pourrait intercepter le message 4 et le réinjecter directement à B.

Ce protocole est-il sûr ? La clé K_{ab}, que ce soit celle reçue par A ou celle reçue par B, est-elle véritablement connue uniquement de A, B, et S ? Les clés K_{ab} de A et B sont-elles les mêmes ? A peut-il avoir confiance en le fait que seul B (à part S) connaît K_{ab} ? Ce sont quelques-unes des questions que l'on peut se poser.

En fait, ce protocole n'est pas sûr. On peut montrer qu'il existe des attaques où les clés K_{ab} reçues par A et B sont différentes. Celle de A est toujours secrète (connue uniquement de A et S), mais celle de B peut provenir d'un rejeu. L'attaque typique est celle de la figure 2.

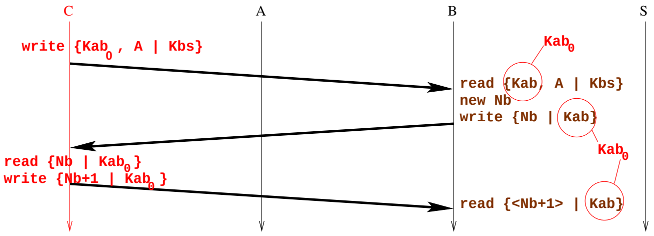

FIG. 2 – Une attaque contre le protocole de Needham-Schroeder à clés secrètes

Elle consiste pour l'intrus C à rejouer un message 3. d'une ancienne session, disons $\{K_{ab0}, A\}_{K_{bs}}$, suffisamment ancienne pour que C ait pu prendre connaissance de l'ancienne clé de session K_{ab0}. Il initie directement une session avec B à l'étape 3 (noter que B n'intervient pas avant cette étape), qui demande confirmation en créant le nonce N_b, et en envoyant le message $\{N_b\}_{K_{ab0}}$ à A. C déroute toutes les communications depuis B, et déchiffre lui-même ce dernier message, calcule $N_b + 1$ et le rechiffre avec K_{ab0}. Ainsi B dispose d'une vieille clé au lieu d'une fraîche. De plus cette clé est connue de l'intrus.

Le protocole à clés symétriques de Needham-Schroeder présenté ci-dessus est caractéristique des protocoles cryptographiques. Bien qu'il ait été publié, et ainsi soumis au regard critique de la communauté des chercheurs du domaine, il contient une faille de sécurité qui n'a été découverte que quelques années plus tard. Les failles sont parfois extrêmement subtiles. Dans le cas du protocole d'Otway-Rees [33], l'attaque [28] exige qu'un même principal joue deux rôles différents (ceux de A et de B) au cours d'une même session. Dans ce dernier cas, il a fallu six ans pour découvrir l'existence d'une attaque, mais il faut parfois plus longtemps. Un cas extrême est le protocole de Needham-Schroeder à clés publiques [32] (à ne pas confondre avec celui de la section 2.2), dont l'attaque a été découverte dix-sept ans plus tard [27].

En tout état de cause, ces quelques exemples ont permis à la communauté scientifique de prendre conscience de la nécessité de moyens systématiques de recherche d'attaques, ou a contrario de preuve formelle de sécurité des protocoles cryptographiques.

3. Le modèle de Dolev-Yao et ses applications

3.1. Les principes fondamentaux

Un modèle simple servant de fondement à l'étude des protocoles cryptographiques est dû à Dolev et Yao [18]. Il connaît de nombreuses variantes et a servi de support à de nombreuses méthodes de vérification [29, 8, 9, 10, 35, 34, 15, 6, 24, 20, 31, 23, 19].

En premier lieu, il consiste à simplifier le modèle de communication entre principaux, en prenant en compte la quasi-omnipotence des intrus. Nul n'est besoin de préciser le modèle de communication (files, réseaux avec ou sans panne, avec ou sans duplication ou perte ou réordonnancement de messages, etc.) entre principaux : l'intrus pouvant espionner et dérouter toutes les lignes de communication, toute communication d'un message M entre deux principaux peut être simulée par un envoi de M à l'intrus, suivi d'un envoi de M par l'intrus au destinataire souhaité. Comme l'intrus peut simuler n'importe quel envoi et réception de message, on simplifie le modèle de communication en supposant que tout message envoyé est envoyé à l'intrus, et tout message reçu l'est de l'intrus.

En second lieu, le modèle de Dolev et Yao suppose que les messages envoyés et reçus ne sont pas ni des nombres ni des suites de bits, mais des éléments d'une algèbre de termes, éventuellement modulo une théorie équationnelle. Les messages sont engendrés par une grammaire de la forme :

$$M ::= D|K|pair(M,M)|p_1(M)|p_2(M)|encrypt(M,K)|decrypt(M,K)$$

où D parcourt un ensemble de données de base (entiers, réels, etc.), K parcourt un ensemble de clés, $pair(M_1, M_2)$ est le couple formé de M_1 et de M_2, $p_1(M)$ récupère la première composante du couple M, $p_2(M)$ récupère la deuxième composante, $encrypt(M,K)$ chiffre le message M avec la clé K, tandis que $decrypt(M,K)$ déchiffre le message chiffré M à l'aide de la clé K. Les messages sont à comprendre modulo la théorie équationnelle :

$$\begin{array}{rcl} p_1(pair(M_1, M_2)) & = & M_1 \\ p_2(pair(M_1, M_2)) & = & M_2 \\ decrypt(encrypt(M,K),K) & = & M \end{array}$$

En particulier, toute tentative de déchiffrer $\{M\}_K$ (codé ici par $encrypt(M,K)$) en utilisant une clé K' différente de K fournit un terme $decrypt(encrypt(M,K),K')$ qui est prouvablement différent de M dans la théorie équationnelle ci-dessus — plus précisément, qui est différent de M dans le modèle initial de cette théorie équationnelle.

Troisièmement et finalement, le modèle de Dolev et Yao représente l'intrus comme un système déductif, qui représente tous les messages qu'un intrus peut fabriquer par un ensemble de règles, qui définissent un prédicat $E \models M$ ("à partir de l'ensemble de messages E, l'intrus peut fabriquer le message M") :

1. $E, M \models M$: de tout ensemble E de messages augmenté d'un message M, l'intrus peut inférer M (rejeu simple) ;
2. Si $E \models M_1$ et $E \models M_2$, alors $E \models pair(M_1, M_2)$: l'intrus peut fabriquer tous les couples de messages formés de messages qu'il peut fabriquer ;
3. Si $E \models M$ alors $E \models p_1(M)$ et $E \models p_2(M)$: l'intrus peut extraire les composantes des couples ;
4. Si $E \models M$ et $E \models K$ alors $E \models encrypt(M,K)$: l'intrus peut effectuer tous les chiffrements qu'il souhaite ;
5. Si $E \models M$ et $E \models K$ alors $E \models decrypt(M,K)$: l'intrus peut tenter de déchiffrer tout message avec toute clé qu'il sait fabriquer ;
6. Si $E \models M$ et $M = N$ dans la théorie équationnelle ci-dessus, alors $E \models N$.

L'intrus peut produire des messages en utilisant ces règles en aussi grand nombre et aussi souvent que nécessaire. Par contre, l'intrus ne peut pas produire de message d'aucune autre façon. Notamment, l'intrus ne peut pas produire M à partir de $encrypt(M,K)$, à moins de ne pouvoir produire la clé K ou de ne pouvoir produire M sans partir de $encrypt(M,K)$.

3.2. Modélisation en logique du premier ordre

Sous ces hypothèses, il est maintenant raisonnablement facile de représenter une ou plusieurs sessions d'un protocole cryptographique, ainsi que les hypothèses initiales (dans l'exemple de la section 2.2, le fait que K_{as} et K_{bs} sont initialement secrètes, c'est-à-dire impossibles à produire par l'intrus) et les propriétés à prouver. À titre d'exemple, montrons comment le protocole de Needham-Schroeder à clés symétriques de la section 2.2 peut être décrit à l'aide d'une formule. Nous utilisons une notation à la Prolog pour bien montrer l'analogie avec l'écriture d'un programme : la notation `A :- B1, ., Bn` est la clause "`B1` et ... et `Bn` impliquent `A`". Nous conviendrons aussi que les identificateurs commençant par une minuscule sont des constantes, les variables commençant par des majuscules, comme en Prolog.

En premier, on va coder les comportements des principaux A, B, et S. Pour ceci, on représente chaque principal comme un automate, dont l'état est un n-uplet comportant un champ décrivant à quelle étape du protocole le principal se trouve (un compteur de programme), et autant de champs que nécessaires pour donner des valeurs aux variables locales. Par exemple, A maintiendra un champ pour son identité A, un pour celle de B, un pour la valeur du nonce N_a, un pour la clé K_{as}, un pour la clé K_{ab}, un pour le message `message` (que nous noterons `M`), et un pour le nonce N_b. Le prédicat

```
Pa ((S, A, B, Na, Kas, Kab, Nb), E, (S', A', B', N'a, K'as, K'ab, N'b), E')
```

exprimera que A peut passer de l'état

```
(S, A, B, Na, Kas, Kab, Nb)
```

à l'état

```
(S', A', B', N'a, K'as, K'ab, N'b)
```

alors que l'intrus disposait de tous les messages dans l'ensemble `E` et dispose après cette action de tous les messages dans l'ensemble `E'`. On écrira alors le rôle de A (la suite des actions qu'il exécute, cf. la colonne de gauche de la figure 1) par exemple comme suit :

```
% A émet le message 1,
% et passe de l'état "avant-1" à l'état "entre-1-et-2":
Pa((avant-1, A, B, _, Kas, Kab, M, Nb), E,
    (entre-1-et-2, A, B, Na, Kas, Kab, M, Nb), E') :-
    % new Na est simulé en stockant dans $N_a$ une constante fraîche:
    Na = nonce-A-1,
    % write A, B, Na ajoute la liste [A, B, Na]
    % aux connaissances E de l'intrus:
    E' = add (pair (A, pair (B, pair (Na, nil))), E).

% A reçoit le message {Na, B, Kab, {Kab, A}Kbs}Kas,
% et passe de "entre-1-et-2" à "entre-2-et-3":
Pa((entre-1-et-2, A, B, Na, Kas, _, _, Nb), E,
    (entre-2-et-3, A, B, Na, Kas, Kab, M, Nb), E') :-
    % read {<Na>, <B>, Kab, message | Kas} se décompose comme suit:
    knows(E, M1),       % lecture du message reçu M1,
    decrypt(M1, Kas) = %  déchiffrement et lecture des composantes de M1:
           pair (N'a, pair (B', pair (Kab, pair (M, nil)))),
    N'a = Na,           % vérification du nonce Na,
    B' = B,             % vérification de l'identité de B,
    E' = E.             % l'intrus n'apprend  rien durant cette étape.
```

```
% A émet le message M, et passe de "entre-2-et-3" à "entre-3-et-4":
Pa ((entre-2-et-3, A, B, Na, Kas, Kab, M, Nb), E,
    (entre-3-et-4, A, B, Na, Kas, Kab, M, Nb), E') :-
   E' = add (M, E).   % write message

% A reçoit le message {Nb}Kab,
% et passe de "entre-3-et-4" à "entre-4-et-5":
Pa ((entre-3-et-4, A, B, Na, Kas, Kab, M, _), E,
    (entre-4-et-5, A, B, Na, Kas, Kab, M, Nb), E') :-
                      % read {Nb | Kab}:
    knows(E, M2),     % lecture du message reçu M2,
    decrypt(M2, Kas) = Nb,  % et récupération du nonce $N_b$,
    E' = E.

% A envoie {Nb+1}Kab et passe de "entre-4-et-5" à "après-5":
Pa ((entre-4-et-5, A, B, Na, Kas, Kab, M, _), E,
    (après-5, A,  B, Na, Kas, Kab, M, Nb), E') :-
    E' = add (encrypt (Nb+1, Kab), E).    % write {Nb+1 | Kab}
```

où `knows` est un prédicat binaire codant la relation $\mapsto$, qu'il est facile de définir en Prolog (exercice : retranscrire les clauses 1.–6. de la section 3.1 en Prolog).

On écrit de même une série de clauses définissant le prédicat `Pb` caractérisant les actions possibles de B, dont l'état instantané est de la forme

```
(S, Kab, A, Nb, Kbs)
```

et le prédicat `Ps` caractérisant les actions possibles de S, dont l'état instantané est de la forme

```
(S, A, B, Na, Kab, Kas, Kbs)
```

On peut maintenant décrire la composition parallèle des trois principaux A, B, et S. Une configuration globale est un quadruplet `(Ea, Eb, Es, E)` comportant un état `Ea` de A (un septuplet `(S, A, B, Na, Kas, Kab, Nb)`), un état `Eb` de B (un quintuplet `(S, Kab, A, Nb, Kbs)`), un état `Es` de S (un septuplet `(S, A, B, Na, Kab, Kas, Kbs)`), et un ensemble de messages `E` récupérés par l'intrus. Définissons un prédicat `P` décrivant les actions possibles de la composition parallèle de A, B, et S :

```
P ((Ea, Eb, Es, E), (E'a, E'b, E's, E'))
```

décrit toutes les transitions possibles de l'état global `(Ea, Eb, Es, E)` vers l'état global `(E'a, E'b, E's, E')`, par entrelacement entre les actions de A, de B, et de S :

```
P((Ea, Eb, Es, E), (E'a, E'b, E's, E')) :-
   Pa  (Ea,  E,  E'a,  E'), Eb=E'b, Es=E's.

P((Ea, Eb, Es, E), (E'a, E'b, E's, E')) :-
   Ea=E'a, Pb(Eb,  E,  E'b, E'), Es=E's.

P((Ea, Eb, Es, E), (E'a, E'b, E's, E')) :-
   Ea=E'a,  Eb=E'b,  Ps(Es, E, E's, E').
```

Une exécution est une suite éventuellement vide de transitions, et `exec` peut être définie comme la clôture réflexive transitive de `P`, de sorte que

```
exec((Ea, Eb, Es, E), (E'a, E'b, E's, E'))
```

soit vrai si le système global peut évoluer de `(Ea, Eb, Es, E)` vers l'état `(E'a, E'b, E's, E')` en un nombre quelconque d'étapes :

```
exec((Ea, Eb, Es, E), (Ea, Eb, Es, E)).          % pas d'action.

exec((Ea, Eb, Es, E), (E''a, E''b, E''s, E'')) :- % une  transition,
                                                  % suivie d'une exécution:
     P((Ea, Eb, Es, E), (E'a, E'b, E's,  E')),
     exec((E'a,  E'b, E's, E'), (E''a, E''b, E''s, E'')).
```

Notons $Exec$ la conjonction de toutes ces clauses, préalablement universellement quantifiées. On a ainsi entièrement spécifié toutes les exécutions possibles du protocole cryptographique, incluant non seulement les exécutions normales, comme celle de la figure 1, mais aussi toutes les exécutions pathologiques, comme celle de la figure 2.

Notons $Intrus$ la modélisation du prédicat `knows` ($\mapstochar\!\rightarrow$) telle que décrite dans les points 1.-6. à la section 3.1, retranscrite en logique du premier ordre.

Vérifier que la valeur de K_{ab} que A possède à la fin du protocole est secrète, par exemple, en supposant que K_{as} et K_{bs} sont initialement secrètes revient à prouver la formule :

$$
\begin{array}{l}
\left(\begin{array}{l}
Exec \wedge Intrus \wedge \forall E_{init}, E_{a\ init}, E_{b\ init}, E_{s\ init} \cdot \\
\quad (\forall K_{as} \cdot E_{a\ init} = (\texttt{avant-1}, \_, \_, \_, K_{as}, \_, \_, \_) \Rightarrow E_{init} \not\mapsto K_{as}) \\
\wedge\, (\forall K_{bs} \cdot E_{b\ init} = (\texttt{avant-1}, \_, \_, \_, K_{bs}) \Rightarrow E_{init} \not\mapsto K_{bs}) \\
\wedge (\forall K_{as}, K_{bs}, K_{ab} \cdot E_{s\ init} = (\texttt{avant-1}, \_, \_, \_, K_{ab}, K_{as}, K_{bs}) \\
\qquad\qquad E_{init} \not\mapsto K_{as} \wedge E_{init} \not\mapsto K_{bs} \wedge E_{init} \not\mapsto K_{ab})
\end{array}\right) \\
\Rightarrow \quad \forall E_{fin}, E_{a\ fin}, E_{b\ fin}, E_{s\ fin}, K_{ab} \cdot \\
\qquad \texttt{exec}((E_{a\ init}, E_{b\ init}, E_{s\ init}, E_{init}), (E_{a\ fin}, E_{b\ fin}, E_{s\ fin}, E_{fin})) \\
\qquad \wedge E_{a\ fin} = (\texttt{après-5}, \_, \_, \_, \_, K_{ab}, \_) \\
\qquad\quad \Rightarrow E_{fin} \not\mapsto K_{ab}
\end{array}
$$

En clair, si les clés K_{as}, K_{bs}, K_{ab} ne sont pas connues dans l'état initial de connaissances E_{init} de l'intrus, alors pour tout état final où A est au compteur `après-5`, la version K_{ab} de la clé que A est censé partager avec B n'est pas connue de l'intrus dans son état final de connaissances E_{fin}.

Cette formule est prouvable en logique du premier ordre interprétée modulo la théorie équationnelle de la section 3.1 (à un détail près, lié à la nature inductive de la définition de $\mapsto$, que nous verrons en section 3.3). On peut pour s'en convaincre utiliser des assistants de preuve comme Coq [9, 10] ou Isabelle [35], ou des systèmes de démonstration automatique [24]. On peut ainsi vérifier des propriétés de sécurité sous diverses hypothèses initiales. Par exemple, dans la formule ci-dessus, on a pris des hypothèses relativement libérales ; notamment, on n'a pas demandé que les clés K_{as} ou K_{bs} aient les mêmes valeurs pour A et pour S, et pour B et pour S respectivement. En fait, A et B pourraient déjà avoir subi des attaques, avec comme conséquence que par exemple A et S ne sont pas d'accord sur la valeur de K_{as}. Ceci ne compromet pas le secret de K_{ab} à la fin de l'exécution de A, du moment que la clé K_{as} de A et la clé K_{as} de S sont toutes les deux secrètes initialement.

De même, on peut prouver que les différentes variantes des clés à long terme K_{as} et K_{bs} restent toujours secrètes à toute étape du protocole. Par contre, la formule similaire tendant à établir que le K_{ab} de B est secret à la fin de l'exécution de B est fausse et donc non prouvable : l'attaque de la figure 2 est un contre-exemple. La recherche de preuve permet de donner une idée du contre-exemple, néanmoins les systèmes de preuve automatique sont souvent incapables d'extraire un contre-exemple automatiquement. Des systèmes de model-checking comme FDR [27] sont plus directement orientés vers la détection d'attaque, mais ne fournissent aucune garantie de sécurité au cas où aucune attaque n'est trouvée, car ils travaillent sur des abstractions à états finis des protocoles cryptographiques, qui sont des systèmes à états infinis ; certains travaux montrent cependant que sous certaines hypothèses, des abstractions à suffisamment d'états finis suffisent pour obtenir des garanties de sécurité [38].

3.3. Commentaires et extensions

Prédicat $\mapsto$ et théories inductives. Le prédicat $E \mapsto M$ est défini comme étant défini inductivement par les clauses 1.-6. de la section 3.1. Autrement dit, l'ensemble des couples (E, M) tels que $E \mapsto M$ est le *plus petit* tel que les clauses 1.-6. sont vérifiées. L'aspect inductif permet en particulier de montrer des assertions de la forme $E \not\mapsto M$. Or la formalisation en logique du premier ordre ne permet pas d'inclure cet aspect inductif. Il y a plusieurs solutions à ce problème : utiliser un assistant de preuve incluant des principes inductifs, c'est-à-dire de récurrence [9, 10, 35], ou bien coder sous forme de formules de logiques du premier ordre quelques principes suffisants pour démontrer des formules de la forme $E \not\mapsto M$ [10]. Il se trouve qu'en fait un faible nombre de telles formules est nécessaire : El Kadhi [19] dégage une notion de théorème de transfert, de la forme “sous certaines hypothèses, si $E \not\mapsto M$ alors $E, M' \not\mapsto M$”, qui permet de conclure qu'une donnée M secrète le reste même après l'écriture du message M', et montre qu'avec essentiellement trois théorèmes de transfert, on peut établir que certaines données secrètes le restent au cours d'un protocole. ([19] considère en fait le problème plus difficile de valider automatiquement un programme, par analyse statique, en ce qui concerne la non-divulgation de données sensibles en présence de primitives cryptographiques.)

Enrichissement de la théorie équationnelle. La théorie équationnelle des messages de la section 3.1 peut être enrichie pour prendre en compte certaines propriétés algébriques, notamment des algorithmes de chiffrement. Par exemple, le chiffrement à l'aide de l'algorithme RSA a de nombreuses propriétés, parmi lesquels $encrypt(M^n, K) = encrypt(M, K)^n$, qui mènent à des attaques. Pour établir de façon certaine la sécurité d'un protocole cryptographique, toutes les équations valides doivent être ajoutées à la théorie équationnelle. Un certain nombre de telles équations est fourni dans [2].

Appauvrissement de la théorie équationnelle. A contrario, les réalisations informatiques professionnelles de protocoles cryptographiques incluent de nombreuses précautions de codage (chiffrement en mode CBC plutôt que par blocs, inclusion d'information redondante de typage pour éviter les confusions de types, etc. [37]). Par exemple, au lieu de chiffrer M par K en utilisant RSA directement, on va d'abord encoder M par exemple en ASN.1 [25], puis appliquer RSA sur le message résultant avec la clé K. Ces précautions ont essentiellement pour effet d'invalider toutes les équations non triviales (autres que de la forme $x = x$) qui tenaient sur l'algorithme de chiffrement brut. On peut pousser l'élimination de la théorie équationnelle plus loin, dans ces conditions : à cause des informations de types incluses, on peut reconnaître si un message est un couple ou un chiffrement, avec faible probabilité d'erreur. Ceci permet de se passer intégralement des opérateurs p_1, p_2, et $decrypt$. Par exemple, au lieu d'exprimer que $E \mapsto M_1$ et $decrypt(M_1, K_{as}) = pair(N'_a, pair(B', pair(K_{ab}, pair(M, \mathtt{nil}))))$ dans la modélisation du protocole de la section 3.2, on écrira directement que $E \mapsto encrypt(pair(N'_a, pair(B', pair(K_{ab}, pair(M, \mathtt{nil})))), K_{as})$. On peut alors entièrement se passer de l'égalité dans la modélisation (comme dans [10, 23]), ce qui simplifie grandement la recherche de preuve ; notamment la recherche automatique de preuve au premier ordre sans égalité est beaucoup plus simple qu'avec égalité.

L'approche par réécriture. Une autre modélisation des protocoles cryptographiques utilise, elle, uniquement la notion de réécriture [24, 20]. L'état global du système est représenté comme un terme modulo la théorie équationnelle de la section 3.2 et l'associativité et la commutativité de l'opérateur d'union ensembliste utilisé pour former l'ensemble E des messages connus de l'intrus. L'exécution du protocole est représentée par la réécriture de ce terme, directement. Pour montrer que K_{ab} reste secret dans toute exécution du protocole (par exemple), on montre qu'il n'est pas déductible d'aucun état global accessible depuis les états initiaux, soit directement [24], soit par calcul d'un sur-ensemble de cet ensemble des états accessibles par une méthode qui termine toujours (par l'utilisation d'automates d'arbres, par exemple [20]).

Authentification. Nous n'avons traité jusqu'ici que des propriétés de secret. Une autre propriété fondamentale est l'authentification. Un moyen simple de traiter de l'authentification est de conserver dans la modélisation une information sur l'identité du créateur réel de toute donnée. On ajoute pour cela, par exemple, une variable de relation R entre messages et identités de participants parmi les composantes de l'état global : (M, X) est dans la relation R si et seulement si le participant X est capable de produire le message M. Formellement, on peut utiliser un prédicat ternaire $belongs(M, X, R)$ ("(M, X) est dans la relation R") et une constante `i` dénotant l'identité de l'intrus. Les deux propriétés à maintenir durant l'exécution du protocole sont, d'une part, que si $E \models\!\!\rightarrow M$ alors $belongs(M, \mathtt{i}, R)$, et d'autre part que lorsqu'un participant X construit un message M, alors $belongs(M', X, R)$ pour tout sous-message non variable M' de M. Le fait qu'un message M reçu par B, par exemple sous la forme $\{M\}_{K_{ab}}$, vienne authentiquement de A reviendra à vérifier que tout X tel que $belongs(M, X, R)$ est en fait égal à A. Ceci interdit en particulier que $belongs(M, \mathtt{i}, R)$, autrement dit que l'intrus ait pu fabriquer le message M, mais aussi que $belongs(M, C, R)$ par exemple, avec $C \neq A$, ce qui interdit que l'intrus ait pu rejouer un message $\{M\}_{K_{ab}}$ venant de C même sans avoir accès à M. Une telle approche de l'authentification est à l'œuvre, quoique sous une forme en apparence différente, dans [13, 20, 24]. Cette notion d'authentification inclut la notion d'intégrité, qui exprime que le message n'a pas été modifié depuis sa création. Différents auteurs utilisent des notions différentes d'authenticité, voir par exemple la notion d'"agreement" [39] qui exprime que si M est authentiquement de A, alors lorsque B termine sa session, A a lui-même auparavant commencé la partie correspondante de la session.

Multi-sessions. Il est important de comprendre dans combien de sessions un même principal peut participer. Dans le protocole de la section 3.2, par exemple, A et B vont typiquement participer à une session, puis éventuellement à une autre, et ainsi de suite, en séquence ; ce qu'on appelle le mode *multi-sessions séquentiel*. Notre modélisation considérait que A s'arrêtait après une session, ce qui est trop restrictif. Le serveur de clés S, lui, peut participer à de nombreuses sessions en parallèle (le mode *multi-sessions parallèle*). Plus subtilement, on peut envisager des modes de fonctionnement où un même principal endosse plusieurs rôles (celui de A, celui de B par exemple) soit à la suite, soit même en parallèle. Ainsi l'attaque de Paulson [36] sur le protocole d'Otway-Rees demande à ce qu'un même principal joue en même temps le rôle de A dans une session et le rôle de B dans une autre. L'attaque de Lowe [27] sur Needham-Schroeder à clés publiques nécessite que l'intrus joue à la fois le rôle du participant B honnête dans une session avec A, et le rôle d'un participant malhonnête A dans une autre session parallèle avec B.

Il existe des réponses partielles à ce genre de situations. Notamment, l'algorithme de [15] considère tous les principaux comme des complices de l'intrus : si A lit $\{N_b\}_{K_{ab}}$ à l'étape 4. du protocole de la section 3.2, et renvoie $\{N_b + 1\}_{K_{ab}}$ à l'étape 5., par exemple, alors on peut considérer que A agit comme un complice de l'intrus, qui permet à ce dernier de déduire $\{N_b + 1\}_{K_{ab}}$ à partir de $\{N_b\}_{K_{ab}}$ même sans connaître K_{ab}. Ceci est utilisé dans [23] pour modéliser de façon efficace et raisonnablement précise le comportement des principaux en multi-sessions parallèle.

Algorithmes de chiffrement à clés publiques. Jusqu'ici nous n'avons traité que de méthodes de chiffrement et de déchiffrement à clés symétriques : pour déchiffrer un message chiffré avec une clé K, c'est encore K qu'on doit utiliser. Des algorithmes comme RSA utilisent une notion de clés asymétriques : on chiffre avec une clé K, et on déchiffre avec la clé inverse K^{-1}. La théorie équationnelle de la section 3.1 doit alors être modifiée, l'équation portant sur $encrypt$ et $decrypt$ devenant $decrypt(encrypt(M, K), K^{-1}) = M$. Il est aussi nécessaire d'ajouter l'équation ${K^{-1}}^{-1} = K$. Notons que même en présence de paires de clés inverses, il est possible de se passer d'équations, cf. [23].

Autres propriétés de sécurité et commerce électronique. Le commerce électronique pose des défis supplémentaires à relever [12]. D'abord, les propriétés de secret et d'authentification, si elles sont toujours fondamentales, ne sont plus les propriétés essentielles que l'on cherche à prouver. Des propriétés plus complexes sont à établir : non-duplication de messages (factures), non-révocation de messages (commandes), ainsi que nous l'avons déjà mentionné dans l'introduction. Ces propriétés plus complexes requièrent de travailler sur des abstractions du protocole, lesquelles peuvent être en grande partie calculées automatiquement [11, 7]. Un autre aspect complexe des protocoles électroniques est qu'ils incluent typiquement non pas deux principaux (plus un serveur S) de confiance, mais trois principaux (l'acheteur, le vendeur, la banque) tels que tout sous-ensemble de deux se méfie du troisième. Il est donc plus délicat de séparer le monde en deux zones, les principaux de confiance et l'intrus, tout principal étant considéré comme un intrus par les deux autres. Bolignano [12] montre comment moduler une modélisation dans le style de la section 3.2 pour inclure non seulement des hypothèses initiales de secret, mais aussi des hypothèses d'honnêteté des différents principaux du protocole. La méthode de [14] permet aussi de modéliser des secrets entre participants honnêtes qui ne soient pas connus d'autres participants honnêtes.

3.4. Autres approches

Logiques modales pour l'authentification. Une des premières approches de modélisation des protocoles cryptographiques est celle de [13]. Il s'agit d'une logique modale, dans laquelle on peut exprimer qu'un principal croit une conjonction de formules, qu'un principal voit un message (qu'il l'a reçu ou l'a déduit de messages reçus), qu'un principal a dit un message dans le passé, qu'un principal a juridiction sur un message (s'il dit ce message, alors ce message doit être cru par les principaux qui le voient), qu'un message est frais, qu'une clé est bonne pour communiquer de façon confidentielle entre deux principaux, qu'un message est un secret partagé entre deux principaux. A ce langage logique se rajoutent des principes déductifs. Cette logique a eu de nombreux détracteurs et de nombreux imitateurs : on consultera le panorama [21]. On peut considérer que ces logiques, dont la sémantique n'est pas extrêmement claire, sont obsolètes pour une grande part. Une application intéressante de ce genre de logiques est non pas la vérification mais la réalisation d'un modèle de droits d'accès perfectionné [5].

Algèbres de processus. Le pionnier ici est le spi-calcul de [3]. Il s'agit d'un langage de programmation inspiré du pi-calcul [30], un langage à visées fondatrices pour les aspects de calcul distribué, incluant des aspects de mobilité des processus. L'intérêt principal du spi-calcul est de permettre une définition extrêmement précise des propriétés de secret et d'authenticité, qui ne fait pas appel aux notions relativement peu concrètes de déductibilité d'un message par un intrus ou de créateur réel d'un message, comme utilisées dans les sections précédentes. Par exemple, un message M est secret si et seulement si on peut remplacer ce message par n'importe quel autre, sans qu'aucun processus mis en parallèle avec la session étudiée (l'environnement, analogue de l'intrus dans ce modèle) ne puisse observer une différence. Cette différence est exprimée à l'aide de notions classiques en théorie des algèbres de processus, notamment la may-testing equivalence et l'utilisation de notions de bisimulations entre processus. De même, un message M est authentique si et seulement si l'environnement ne peut observer aucune différence entre le protocole où M est effectivement reçu et celui où c'est le message souhaité M_0 qui est (magiquement) reçu par le principal souhaité. Les notions de bisimulations décidables passant au contexte sont cependant difficiles à dégager [4]. Il semble que des disciplines de typage soient une façon effective de définir des critères suffisants de confidentialité ou d'authentification, dont la correction soit prouvable formellement par rapport aux équivalences de test du spi-calcul [1]. Il est à noter que la notion du secret du spi-calcul est beaucoup plus contraignante que celle que nous avons exposée en section 3.2 : ici un message est secret si et seulement si l'intrus ne peut rien en dire ; en particulier l'intrus ne peut même pas détecter si un même message (secret) est envoyé deux fois de suite. (Contrairement à l'intuition, il est possible de concevoir des protocoles ayant cette propriété ; mais les protocoles pratiques ne l'ont pas en général.) Un des derniers raffinements du spi-calcul est le pi-calcul appliqué [2], qui offre notamment la possibilité d'inclure une théorie équationnelle arbitraire, par exemple dans le style de celle de la section 3.2.

Théorie de la complexité. L'approche la plus précise et la plus fidèle à la réalité, mais aussi la plus difficile à exploiter, repose sur la modélisation des protocoles par machines de Turing probabilistes et évaluation des risques de divulgation, de non-authenticité ou autres par majorations des probabilités d'attaques, sous des hypothèses de théorie de la complexité. Par exemple, la notion usuelle de secret dans ce cadre est similaire à celle du spi-calcul, tout en étant plus précise : un message M est secret si et seulement si la probabilité qu'un intrus puisse détecter que M a une propriété P non triviale donnée, en temps polynomial probabiliste, est inférieure à une borne suffisamment petite. (La notion de temps polynomial probabiliste réfère à diverses notions proches, l'intuition étant par exemple que le temps moyen pris par un ordinateur ayant accès à une source de bits aléatoires pour détecter la propriété P est un polynôme en la taille de M.) Une bonne introduction est le livre [22]. L'intérêt est que ce modèle de calcul est exactement celui des cryptologues, ce qui permet d'inclure naturellement les défauts ou avantages précis des algorithmes (de chiffrement, de génération de nombres aléatoires, etc.) dans l'évaluation de la sécurité des protocoles cryptographiques ; ce que nous tentions de faire en section 3.3 en ajoutant des équations à la théorie équationnelle sur les messages. Le domaine est extrêmement fécond, et a permis de définir des protocoles extrêmement sûrs, ou de comprendre précisément pourquoi certains protocoles apparemment sûrs présentent en fait des failles subtiles. Mais cette approche est difficilement mécanisable, encore aujourd'hui.

4. Conclusion

Le domaine de la modélisation et de la vérification des protocoles cryptographiques a connu une véritable explosion dans les années 1990. On dispose à l'entrée des années 2000 de toute une gamme de modèles et de méthodes. Les modèles dérivés de celui de Dolev et Yao semblent être les plus avancés en ce qui concerne l'expressivité et l'automatisation, mais ils sont encore relativement abstraits, comparés notamment au modèle plus opérationnel du spi-calcul et du pi-calcul appliqué, et surtout au modèle réaliste des machines de Turing probabilistes cher aux cryptologues. Il reste encore de nombreuses améliorations à apporter aux modèles à la Dolev-Yao (automatisation, gestion des différents modes multi-sessions, extension aux problèmes propres au commerce électronique ou à d'autres domaines, par exemple le vote anonyme, en particulier). A côté de cela, il est nécessaire aujourd'hui de formaliser les rapports entre les différents modèles, par exemple pour justifier formellement certains choix apparemment arbitraires du modèle de Dolev-Yao (par exemple, pourquoi l'intrus n'est-il pas autorisé à créer des nonces ? Ceci poserait des problèmes dans la vérification, et on peut supposer sans perte de généralité que tous les nonces de l'intrus ont été créés à l'avance — à condition de faire attention à ne pas imposer des conditions initiales trop contraignantes). Plus finement, on peut souhaiter intégrer des considérations probabilistes dans des modèles à la Dolev-Yao, dans le but d'évaluer les risques d'attaques dans le style du modèle à base de machines de Turing probabilistes. Quelques tentatives préliminaires ont déjà été effectuées [26].

Références

[1] M. Abadi and B. Blanchet. Secrecy types for asymmetric communication. In *Foundations of Software Science and Computation Structures (FoSSaCS'01)*, pages 25–41. Springer Verlag LNCS 2030, 2001.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.

[3] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols : The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*. ACM Press, 1997.

[4] M. Abadi and A. D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4) :267–303, 1998.

[5] M. Abadi, E. Wobber, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1) :3–32, 1994. Also appeared as SRC Research Report 117.

[6] D. Amadio, Roberto et Lugiez. On the reachability problem in cryptographic protocols. In *CONCUR'2000*, pages 380–394. Springer Verlag LNCS 1877, 2000.

[7] A. Bidoit, Michel et Boisseau. Abstract interpretation : An algebraic approach. In *Workshop on Algebraic Data Types (WADT'01)*. Springer Verlag LNCS, 2001.

[8] D. Bolignano. Une approche à la vérification formelle des protocoles cryptographiques. *Technique et Science Informatique*, 1995.

[9] D. Bolignano. Vérification formelle de protocoles cryptographiques à l'aide de Coq. In *Actes des journées GDR*, 1995.

[10] D. Bolignano. Formal verification of cryptographic protocols. In *Proceedings of the ACM Conference on Computer and Communication Security*, 1996.

[11] D. Bolignano. Towards a mechanization of cryptographic protocol verification. In O. Grumberg, editor, *9th International Conference on Computer Aided Verification (CAV'97)*, pages 131–142. Springer LNCS 1254, 1997.

[12] D. Bolignano. Towards the formal verification of electronic commerce protocols. In *10th IEEE Computer Security Foundations Workshop*, 1997.

[13] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society*, 426(1871) :233–271, 1989.

[14] V. Cortier, J. Millen, and H. Ruess. Proving secrecy is easy enough. In *14th IEEE Computer Security Foundations Workshop*, 2001.

[15] M. Debbabi, M. Mejri, N. Tawbi, and I. Yahmadi. Formal automatic verification of authentication cryptographic protocols. In *1st IEEE International Conference on Formal Engineering Methods (ICFEM'97)*. IEEE, 1997.

[16] T. Dierks and C. Allen. The TLS protocol, IETF RFC 2246. `http://www.ietf.org/rfc/rfc2246.txt`, Jan. 1999.

[17] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6) :644–654, Nov. 1976.

[18] D. Dolev and A. C. Yao. On the security of pubic key protocols. *IEEE Transactions on Information Theory*, IT-29(2) :198–208, Mar. 1983.

[19] N. El Kadhi. Automatic verification of confidentiality properties of cryptographic programs. *Networking and Information Systems*, 2001.

[20] T. Genet and F. Klay. Rewriting for cryptographic protocol verification (extended version). Technical report, CNET-France Telecom, 1999. `http://www.loria.fr/~genet/Publications/GenetKlay-RR99.ps`.

[21] V. D. Gligor, R. Kailar, S. Stubblebine, and L. Gong. Logics for cryptographic protocols—virtues and limitations. In *4th IEEE Computer Security Foundations Workshop*, 1991. draft.

[22] S. Goldwasser and M. Bellare. Lecture notes on cryptography. `http://www-cse.ucsd.edu/~mihir/papers/gb.ps.gz`, Aug. 1999.

[23] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification (extended abstract). In *Proceedings of the Workshop on Formal Methods in Parallel Programming, Theory and Applications (FMPPTA'2000)*, Lecture Notes in Computer Science LNCS 1800, pages 977–984. Springer Verlag, 2000. L'URL `http://www.dyade.fr/fr/actions/vip/jgl/cpv.ps.gz` pointe vers une version plus complète.

[24] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Proceedings of the 7th Conference on Logic for Programming and Automated Reasoning (LPAR'2000)*, pages 131–160. Springer Verlag LNCS 1955, 2000.

[25] J. Larmouth. *Understanding OSI*, chapter 8. Available on the Web, 1994. `http://www.isi.salford.ac.uk//books/osi/chap8.html`.

[26] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, 1998.

[27] G. Lowe. An attack on the Needham-Schroeder public-key protocol. *Information Processing Letters*, 56(3) :131–133, 1995.

[28] W. Mao and C. Boyd. Towards formal analysis of security protocols. In *6th IEEE Computer Security Foundations Workshop*, pages 147–158, 1993.

[29] C. A. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1) :5–36, 1992.

[30] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, pages 1–40 & 41–77, 1992.

[31] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *6th International Static Analysis Symposium (SAS'99)*, pages 149–163. Springer-Verlag LNCS 1694, 1999.

[32] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12) :993–999, 1978.

[33] D. Otway and O. Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1) :8–10, 1987.

[34] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *10th IEEE Computer Security Foundations Workshop*, pages 84–95, 1997.

[35] L. C. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83, 1997.

[36] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6 :85–128, 1998.

[37] B. Schneier. *Applied Cryptography : Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, 1994.

[38] S. Stoller. A reduction for automated verification of authentication protocols. In *Workshop on Formal Methods and Security Protocols*, 1999.

[39] S. Stoller. A bound on attacks on payment protocols. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, 2001.