THÈSE

présentée à l'École Normale Supérieure de Cachan

pour obtenir le grade de

Docteur de l'École Normale Supérieure de Cachan

par : Véronique CORTIER

Spécialité : INFORMATIQUE

Vérification automatique des protocoles cryptographiques

Soutenue le 20 mars 2003

Composition du Jury :

- Martin ABADI — examinateur
- Roberto AMADIO — rapporteur
- Hubert COMON-LUNDH — directeur de thèse
- Jean-Pierre JOUANNAUD — président du jury
- Dale MILLER — rapporteur
- Leszek PACHOLSKI — rapporteur

Remerciements

Je tiens tout d'abord à remercier chaleureusement tous les membres du jury.

Je témoigne toute ma gratitude à Martin Abadi pour avoir accepté de participer au jury de ma thèse et pour ses réponses patientes et ses conseils précis au cours de ma thèse.

Jean-Pierre Jouannaud me fait l'honneur d'être président du jury, je lui adresse mes plus vifs remerciements.

Merci à Roberto Amadio de s'être attelé à la lourde de tâche de rapporteur et pour les échanges scientifiques qui ont eu lieu ces trois années ; merci à Dale Miller et Leszek Pacholski d'avoir également accepté d'être rapporteurs, tâche d'autant plus difficile que cette thèse est écrite en français.

Enfin, je tiens à exprimer ma sincère et profonde reconnaissance à mon directeur de thèse, Hubert Comon-Lundh, pour son soutien continu, ses conseils avisés, sa disponibilité exceptionnelle, et pour m'avoir « appris la recherche » au cours de toutes ces années. J'ai particulièrement apprécié les nombreuses heures qu'il m'a consacrées au cours desquelles chacun écrivait simultanément sur une partie du tableau en cherchant souvent à prouver le contraire de ce que tentait l'autre, tant et si bien que l'heure du déjeuner était souvent oubliée – enfin... surtout par Hubert...

Je souhaite aussi remercier les autres chercheurs avec qui j'ai eu la chance de travailler. Merci en particulier à Harald Ganziger pour m'avoir accueillie dans son équipe lors de mon premier stage de recherche et à Florent Jacquemard pour m'avoir très gentiment encadrée pendant ce stage. Merci à Jon Millen et Harald Rueß pour m'avoir accueillie dans leur formidable équipe du SRI et pour m'avoir beaucoup appris pendant mon séjour là-bas.

Je tiens également à remercier tous les membres du LSV pour leur disponibilité et leur gentillesse, qui font de ce laboratoire un cadre de travail idéal. Merci en particulier à tous les thésards pour les blagues, les conversations animées, les cagnottes carambar et les goûters, les sauts dans les airs, les rapts et tortures de peluches et autres joyeusetés qui m'ont fait bien rire. Merci à Nicolas et Vincent pour m'avoir supportée dans notre bureau et pour leur aide LaTeX et Linux. Merci à François pour ses discussions, ses bouquins, ses facéties et son apprentissage du C++. Merci à Antoine Petit pour son cours en licence qui m'a permis de réviser mon jugement sur l'informatique : « l'informatique, ça consiste à modifier les fichiers de configuration de Windows, ça ne m'intéresse pas ».

Merci à Frédéric pour sa relecture patiente et impitoyable de cette thèse. C'est certainement grâce à lui qu'il n'y a pas une dizaine de répétitions par page et autant de coquilles. Mais merci surtout pour son soutien et ses encouragements pendant toutes ces années.

Enfin, je n'oublie pas mes parents qui y sont certainement pour beaucoup dans tout ça.

Table des matières

CHAPITRE 1

Introduction

« There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will stop major government from reading your files. »

Bruce SCHNEIER, *Applied Cryptography.*

1.1 Les enjeux

Avec le développement des réseaux de communication comme Internet et les réseaux de téléphonie mobile, le besoin d'assurer la confidentialité et l'authenticité des messages échangés a considérablement augmenté. Les protocoles cryptographiques sont des règles d'échange entre les points du réseau, ils permettent de sécuriser les communications. Ils sont utilisés par exemple dans les distributeurs de billets, les abonnements aux chaînes de télévision payantes, la téléphonie mobile, le commerce électronique.

Ces protocoles sont basés sur la cryptographie : étant donnés un message et un nombre secret assez grand (appelé clef), des algorithmes de chiffrement permettent d'obtenir un message chiffré tel qu'il est impossible de retrouver le premier message à moins d'avoir aussi la clef. La description de ces protocoles étant en général très courte, on pourrait penser qu'il est aisé de s'assurer de la fiabilité d'un protocole. Il n'en est rien : de nouvelles attaques sont trouvées régulièrement et parfois sur des protocoles connus depuis plusieurs années.

La difficulté de la conception des protocoles tient au fait que les messages échangés peuvent être écoutés par une tierce personne, interceptés ou modifiés. Ainsi les systèmes d'abonnements aux chaînes télévisées payantes ont déjà été beaucoup piratés, en interposant un ordinateur entre les messages reçus et le décodeur ou en modifiant certains composants. Nous donnons deux exemples d'attaques très simples [AN95].

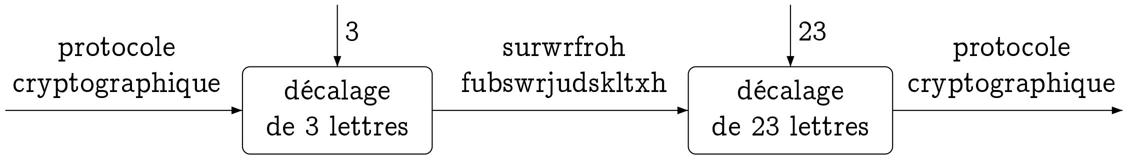
- Lorsqu'un client résilie son abonnement, le serveur du groupement de chaînes envoie usuellement un message au décodeur pour qu'il cesse de décoder. L'attaque « Kentucky Fried Chip » consiste à bloquer ce message, reconnaissable car il n'est pas chiffré.
- Dans un autre système, les communications entre le serveur et les décodeurs étaient simultanées pour tous les utilisateurs. Il était donc possible d'enregistrer le programme crypté puis de le décoder ensuite à l'aide du message communiqué par un utilisateur qui avait payé son abonnement.

Ces attaques ont des répercussions économiques très importantes.

1.2 Cryptographie et protocoles cryptographiques

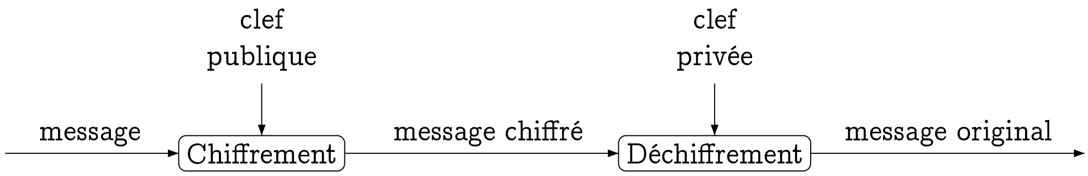
Le chiffrement consiste à transformer un message en un autre de manière à ne plus reconnaître le premier. Le déchiffrement est l'opération inverse.

Un algorithme de chiffrement utilisé par les écoliers consiste à remplacer chaque lettre du message considéré par la lettre suivante dans l'alphabet ou, plus subtilement, à décaler chaque lettre d'un nombre de lettres fixé à l'avance.



Évidemment, cet algorithme de chiffrement n'est pas très robuste puisqu'il suffit d'essayer tous les décalages possibles. Les algorithmes de chiffrement actuels reposent sur des résultats en algèbre (étude des corps de nombre, courbes elliptiques) et sur des problèmes algorithmiquement difficiles comme la décomposition d'un nombre entier en facteurs premiers.

On distingue deux types de chiffrements : les chiffrements symétriques et les chiffrements asymétriques. Le chiffrement symétrique utilise la même clef pour chiffrer comme pour déchiffrer. L'algorithme de chiffrement qui consiste à décaler les lettres de 13 lettres dans l'alphabet est un exemple d'algorithme de chiffrement symétrique. Le chiffrement asymétrique utilise une clef de chiffrement différente de la clef de déchiffrement. La première est souvent divulguée sur le réseau afin que tout participant puisse chiffrer mais la deuxième reste en général secrète pour qu'une seule personne (ou machine) puisse déchiffrer. L'un des algorithmes de chiffrement les plus connus est l'algorithme RSA, dû à R.L. Rivest, A. Shamir et L.M. Adleman [RSA78].



Le chiffrement qui consiste à décaler les lettres de 3 lettres dans l'alphabet n'est pas un bon exemple de chiffrement asymétrique car il est facile de calculer la clef correspondant au déchiffrement : $26 - 3 = 23$. Pour le chiffrement RSA, le record actuel [Eve99] a consisté à « casser » (c'est-à-dire calculer la clef correspondant au déchiffrement) une clef de 512 bits en quatre mois. Personne n'a encore réussi à « casser » une clef de 1024 bits par exemple.

Dresser une liste des algorithmes de chiffrement sort du cadre de cette thèse, nous invitons le lecteur à se reporter au livre d'introduction à la cryptographie de B. Schneier [Sch96b] ou à celui de A. J. Menezes *et al.* [MvOV97].

On pourrait penser que les principales attaques des protocoles cryptographiques reposent sur le décryptage des messages chiffrés. En fait, comme on l'a vu au paragraphe précédent, beaucoup d'attaques reposent sur des principes beaucoup plus simples à mettre en œuvre comme l'interception d'un message et le renvoi d'un autre. Dans le cadre de la vérification des protocoles cryptographiques, une hypothèse très classique est celle *du chiffrement parfait*. Une telle hypothèse assure en particulier qu'il est impossible de décrypter un message chiffré sans avoir la clef correspondante à l'algorithme de déchiffrement. Nous ferons également cette hypothèse dans la plupart des chapitres de la thèse mais nous chercherons à l'affaiblir au chapitre 8 (partie 8.3).

1.3 Terminologie

Dans ce paragraphe, nous présentons la terminologie et les notations usuelles des protocoles cryptographiques, à travers un exemple très classique : le protocole de Needham-Schroeder [NS78]. De très nombreux autres exemples de protocoles sont décrits dans [CJ97].

1.3.1 Protocole

Ce protocole décrit des règles d'échange de messages entre deux *participants*, aussi appelés *agents*. Il se décrit de la façon suivante :

$$\begin{array}{lll} A & \to B: & \{A, N_a\}_{\mathsf{pub}(B)} \\ B & \to A: & \{N_a, N_b\}_{\mathsf{pub}(A)} \\ A & \to B: & \{N_b\}_{\mathsf{pub}(B)} \end{array}$$

À la première étape du protocole, l'agent Alice envoie son nom A et un nombre engendré aléatoirement N_a, aussi appelé *nonce*. Ce message est chiffré par un algorithme de chiffrement asymétrique avec la *clef publique* de B (notée $\mathsf{pub}(B)$), c'est-à-dire que seul l'agent Bob connaît la clef privée correspondant à la clef $\mathsf{pub}(B)$. La notation usuelle pour le chiffrement est $\{\_\}_\_$: le message m chiffré à l'aide d'une clef k est noté $\{m\}_k$. Ainsi, le message envoyé par Alice est noté : $\{A, N_a\}_{\mathsf{pub}(B)}$.

À la deuxième étape du protocole, Bob reçoit le message $\{A, N_a\}_{\mathsf{pub}(B)}$ envoyé par Alice. Comme il a la clef privée (souvent notée $\mathsf{prv}(B)$) lui permettant d'ouvrir le message, il comprend qu'Alice veut lui parler et renvoie le nonce d'Alice ainsi qu'un autre nonce N_b qu'il vient d'engendrer, le tout chiffré avec la clef publique $\mathsf{pub}(A)$ d'Alice.

À la troisième étape du protocole, Alice reçoit le message $\{N_a, N_b\}_{\mathsf{pub}(A)}$ et reconnaît son nonce N_a. Elle en déduit que Bob lui a répondu et elle lui renvoie son nonce N_b chiffré avec sa clef publique pour lui signifier qu'elle connaît maintenant le message N_b. Lorsque Bob reçoit ce message, les deux agents pensent qu'ils sont seuls à connaître le nonce N_b et que celui-ci permet de les *authentifier* : lorsqu'Alice reçoit un message contenant N_b, elle en déduit qu'il vient de Bob et inversement.

Ce protocole peut être joué plusieurs fois avec différents participants et différentes valeurs pour les nonces. Un déroulement du protocole est aussi appelé *session*. De plus, le protocole de Needham-Schroeder décrit ainsi deux programmes : les actions du premier participant (envoi du premier message et recopie du nonce du message reçu) et les actions du deuxième participant (réception d'un message et envoi d'un message avec copie du nonce reçu). Ces deux programmes sont appelés *rôles*. Si un troisième participant Charlie se présente, il pourra ainsi jouer le rôle de A, c'est-à-dire suivre le protocole vu par Alice ou jouer le rôle de B, c'est-à-dire suivre le protocole vu par Bob. Notons que Bob peut également jouer le rôle de A

et inversement, cela signifie tout simplement qu'un participant peut initier une session d'un côté et répondre aussi à une autre session.

1.3.2 Attaque

Comme annoncé au début du chapitre, on suppose que toutes les communications ont lieu en présence d'un intrus. Les premiers intrus considérés étaient *passifs* : ils se contentent d'écouter et d'analyser les messages circulant sur le réseau en déchiffrant les messages lorsqu'ils ont la clef correspondante.

Les intrus considérés à l'heure actuelle (et en particulier dans cette thèse) sont actifs :
- ils interceptent tous les messages circulant sur le réseau,
- ils analysent les messages et en synthétisent de nouveaux,
- ils envoient des messages.

Ainsi G. Lowe [Low96] a découvert une quinzaine d'années après la publication du protocole de Needham-Schroeder que ce dernier avait une faille en présence d'un intrus actif. Cette faille est souvent appelée « man-in-the-middle attack ». Elle est schématisée à la figure 1.1.

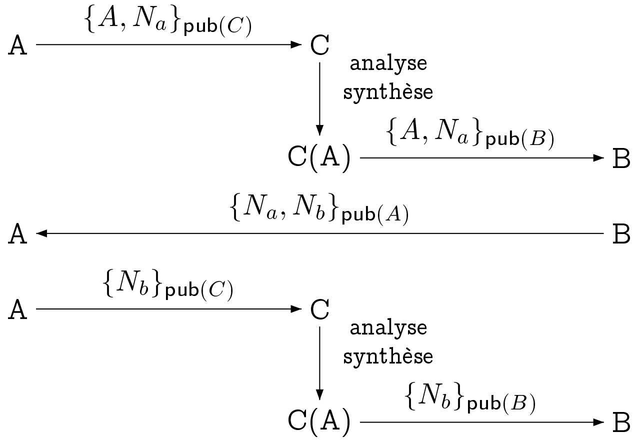


FIGURE 1.1 - *Attaque du protocole de Needham-Schroeder, due à G. Lowe.*

L'agent A commence spontanément une conversation avec un agent C, malhonnête. L'agent C se sert de ce premier message pour se faire passer pour A auprès de B. Celui-ci répond donc à A. L'agent A, reconnaissant son nonce N_a pense que C vient de lui répondre. L'agent A, renvoie donc à C le nonce N_b que l'agent C n'aurait pas dû connaître. L'agent C termine alors le protocole avec B qui croit avoir parlé à A.

Une autre sorte d'attaque est l'attaque par *confusion de type*. Ainsi, G. Lowe a proposé la correction suivante au protocole de Needham-Schroeder :

$$\begin{array}{lll} A & \to B : & \{A, N_a\}_{\mathsf{pub}(B)} \\ B & \to A : & \{B, N_a, N_b\}_{\mathsf{pub}(A)} \\ A & \to B : & \{N_b\}_{\mathsf{pub}(B)} \end{array}$$

L'agent B ajoute maintenant son identité dans le message qu'il envoie. Ce protocole modifié est appelé protocole de Needham-Schroeder-Lowe. Nous prouvons au chapitre 5 que le nonce

Variante du protocole de Needham-Schroeder-Lowe :

$$\begin{array}{lll} A & \to B: & \{A, N_a\}_{\mathsf{pub}(B)} \\ B & \to A: & \{N_a, N_b, B\}_{\mathsf{pub}(A)} \\ A & \to B: & \{N_b\}_{\mathsf{pub}(B)} \end{array}$$

Attaque de la variante du protocole Needham-Schroeder-Lowe :

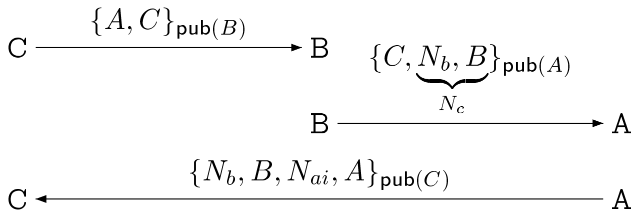


FIGURE 1.2 - *Attaque de type du protocole de Needham-Schroeder-Lowe, due à J. Millen.*

N_b reste secret même lorsque le protocole est joué un nombre arbitraire de fois en présence d'un intrus. Si on le modifie très légèrement en changeant la place de B dans le deuxième message, on obtient un protocole sujet à une attaque dite de *confusion de type* découverte par J. Millen.

Le protocole et l'attaque sont décrits à la figure 1.2 : un agent malhonnête C envoie à B le message $\{A, C\}_{\mathsf{pub}(B)}$. L'agent B confond l'identité de C avec un nonce. Il pense donc répondre à A en lui envoyant le message $\{C, N_b, B\}_{\mathsf{pub}(A)}$. L'agent A en recevant ce message pense que C le sollicite en lui envoyant un nonce $N_c = N_b, B$. Il répond donc par le message $\{N_b, B, N_{ai}, A\}_{\mathsf{pub}(C)}$, ce qui permet à C d'apprendre le nonce N_b qui ne lui était pas destiné. Notons que cette attaque n'est pas très réaliste car la taille des identités et celle des nonces étant très différentes, B ne devrait pas pouvoir confondre C avec un nonce.

Un dernier type d'attaque très classique est l'attaque par *rejeu*. Comme son nom l'indique, une telle attaque consiste à renvoyer un message déjà joué à la place d'un autre. Considérons par exemple le protocole BAN-Yahalom [BAN89] décrit à la figure 1.3. Ce protocole comporte deux participants A et B et un serveur, noté S. Le serveur peut être considéré comme un participant mais dont l'intrus ne peut pas prendre l'identité. Il partage en général des secrets avec chacun des participants, ici une clef symétrique, notée $\mathsf{shr}(A)$ pour la clef partagée entre le serveur et A. Une attaque sur ce protocole [Syv94] consiste pour l'intrus à attendre qu'une session commence entre les agents A et B puis à ouvrir une seconde session avec l'agent B en se faisant passer pour A. L'intrus utilise alors le message émis par B pour le lui renvoyer en se faisant à nouveau passer pour A mais dans la première session. Cette attaque est présentée à la figure 1.3. Elle suppose également une confusion de type comme dans l'exemple précédent : l'agent B confond la paire de deux nonces avec un unique nonce.

Nous venons de présenter deux attaques pour des propriétés de *secret* et également d'*authentification*. Mais les protocoles peuvent assurer d'autres propriétés de sécurité que nous décrivons au paragraphe suivant.

Protocole BAN-Yahalom

1. $A \rightarrow B : A, N_a$
2. $B \rightarrow S : B, N_b, \{A, N_a\}_{\mathsf{shr}(B)}$
3. $S \rightarrow A : N_b, \{B, K_{ab}, N_a\}_{\mathsf{shr}(A)}, \{A, K_{ab}, N_b\}_{\mathsf{shr}(B)}$
4. $A \rightarrow B : \{A, K_{ab}, N_b\}_{\mathsf{shr}(B)}, \{N_b\}_{K_{ab}}$

Attaque du protocole :

$$A \xrightarrow{A, N_a} B$$

$$B \xrightarrow{B, N_b, \{A, N_a\}_{\mathsf{shr}(B)}} S$$

$$C(A) \xrightarrow{A, \underbrace{< N_a, N_b >}_{N'_a}} B$$

$$C(S) \xleftarrow{B, N'_b, \{A, N_a, N_b\}_{\mathsf{shr}(B)}} B$$

$$B \xleftarrow{\{A, N_a(= K_{ab}), N_b\}_{\mathsf{shr}(B)}, \{N_b\}_{N_a}} C(A)$$

FIGURE 1.3 - *Protocole BAN-Yahalom et attaque par rejeu.*

1.3.3 Propriétés de sécurité

Nous appellerons ici *propriété de sécurité* toute propriété qu'un protocole cherche à assurer. Il serait très intéressant de définir plus précisément ce qu'est une propriété de sécurité et, mieux, de définir une logique appropriée mais cela dépasse très largement le cadre de la thèse. Le but de ce paragraphe est de faire une liste (non exhaustive) des propriétés que peuvent tenter d'assurer les protocoles. Les propriétés les plus courantes sont le secret et l'authentification.

1.3.3.1 Secret

On dira en général qu'un protocole assure le secret d'une donnée s si l'intrus ne peut pas connaître cette donnée. Cependant, dans le cadre particulier du spi-calcul [AG97], le secret d'une donnée est une propriété plus forte : on dit qu'une donnée s est secrète si la session qui contient la donnée s est indistinguable de toute session contenant une donnée s' en lieu et place de s. On dira que la première propriété de secret est une propriété d'*accessibilité* tandis que la seconde est une propriété d'*équivalence observationnelle*. D'autre part, parmi les propriétés de secret, on distingue les propriétés de secret globales et locales : on peut demander qu'une donnée soit secrète « tout le temps » ou bien qu'elle soit secrète « tant que la session correspondante n'est pas terminée ». La première notion est plus facile à modéliser puisqu'il suffit d'exprimer que l'intrus ne peut jamais déduire le secret. La seconde propriété demande un modèle plus précis qui permette d'exprimer les débuts et les fins de sessions. Nous verrons en particulier que le modèle du chapitre 3 permet aussi bien d'exprimer les secrets globaux que les secrets locaux.

1.3.3.2 Authentification

L'authentification admet de nombreuses définitions [WL93, Sch97, Low97b, THG99, BAN89, AFG00] et le lien entre celles-ci n'est pas clair. Ainsi, S. Schneider [Sch97] dresse une liste d'une dizaine de définitions distinctes. Dans les grandes lignes, les propriétés d'authentification sont des propriétés de la forme : si l'agent B reçoit un message de la forme m_1 alors l'agent A a envoyé un message de la forme m_2 (qui est en correspondance avec le premier). Ainsi, dans le cas du protocole de Needham-Schroeder, on peut formuler la propriété suivante : si l'agent A émet le message $\{N_b\}_{\mathsf{pub}(B)}$, alors le nonce N_b a bien été engendré par l'agent B. Mais de nombreuses variantes sont possibles : on peut seulement demander que si B accepte le dernier message, alors A a été actif, ou on peut au contraire demander une propriété plus forte à savoir que si l'agent A émet n fois un message de la forme $\{N_b\}_{\mathsf{pub}(B)}$ alors l'agent B a engendré au moins n nonces N_b, *etc.* Comme les outils de vérification des protocoles cryptographiques se consacrent presque tous à la vérification du secret, il serait intéressant de faire le lien entre le secret et l'authentification. Ainsi, au lieu de prouver que le protocole de Needham-Schroeder-Lowe satisfait la propriété d'authentification pour laquelle il a été conçu, on vérifie souvent une propriété de secret dont on pense qu'elle est équivalente. B. Blanchet [Bla02] a fait un premier pas vers un lien formel entre les deux propriétés en associant à une propriété d'authentification particulière, une propriété de secret telle que prouver le secret suffit à assurer l'authentification.

1.3.3.3 Anonymat

Les protocoles utilisés dans la téléphonie mobile visent aussi à assurer l'anonymat [SMA95, SMT94, HKT94] : une personne qui utilise son téléphone portable peut souhaiter qu'on ne puisse pas suivre ses déplacements en examinant les bornes relais qu'elle a utilisé pour transmettre ses conversations. En même temps, le service de téléphonie doit connaître l'identité de la personne pour lui facturer son service. Ce type de protocole doit donc assurer qu'une personne extérieure ne peut pas lier les messages émis à l'identité de la personne qui émet le message ou à l'identité du transmetteur. Ainsi, considérons le cas d'école où Alice signifie à Bob son intention de lui parler en lui envoyant son identité chiffrée avec la clef publique de Bob :

$$A \to B : \{A\}_{\mathsf{pub}(B)}.$$

Un tel protocole n'est pas anonyme. En effet, même si l'intrus ne connaît pas la clef privée de Bob, il peut former les messages $\{C\}_{\mathsf{pub}(B)}$ pour toutes les identités C possibles et les comparer avec le message $\{A\}_{\mathsf{pub}(B)}$ qu'il a vu circuler sur le réseau. Lorsqu'il obtient un message égal à celui envoyé, il déduit qui a envoyé le message. Ce problème d'anonymat est très proche du secret d'un vote : si à la place de l'identité A, on souhaite exprimer le nom d'un candidat, un intrus peut, de la même manière savoir pour qui on a voté. Une correction du protocole décrit ci-dessus est possible en ajoutant un nonce secret dans le message envoyé :

$$A \to B : \{A, N\}_{\mathsf{pub}(B)}.$$

En effet, l'intrus ne peut pas connaître le nonce N et il ne peut donc pas construire le message $\{A, N\}_{\mathsf{pub}(B)}$.

Plusieurs définitions formelles de l'anonymat ont été proposées [Aba02, SH02, Low02]. En particulier, V. Shmatikov et D.J.D Hughes proposent une définition reposant sur l'équivalence observationnelle. En effet, intuitivement, un protocole préserve l'anonymat de l'identité du premier participant (par exemple) si, quel que soit l'agent qui joue le rôle du premier participant, les protocoles obtenus sont indistinguables.

1.3.3.4 Équité

Les signatures de contrat sur Internet amènent de nouveaux problèmes de sécurité. Le premier est la *non répudiation* [KR01]. En effet, comme les messages envoyés sur Internet sont aisément modifiables, un agent peut dénier avoir envoyé ou reçu un message. Aussi, des protocoles spécifiques ont pour but d'établir à la fois un certificat d'émission et de réception des messages. Un deuxième problème des signatures de contrat sur un réseau tient au fait que les communications ne sont pas synchronisées. Aussi, lorsque la première personne a signé le contrat, la deuxième personne peut utiliser cette signature pour obtenir par exemple un meilleur prix auprès d'une tierce personne. Des exemples de protocoles de signature de contrat sont présentés dans [ASW98, GJM99]. Nous décrivons rapidement l'un d'entre eux :

$$\begin{array}{lll} A & \to B : & PCS_A(m,B,T) \\ B & \to A : & PCS_B(m,A,T) \\ A & \to B : & S\text{-}Sig_A(m) \\ B & \to A : & S\text{-}Sig_B(m). \end{array}$$

L'idée de ce protocole est que chaque participant signe d'abord une promesse de signature pour le contrat m. Ils peuvent alors encore chacun annuler l'opération en lançant des sous-protocoles. Mais une fois que l'agent A s'est engagé en signant le contrat, il peut forcer la signature de B grâce d'une part à sa signature $S\text{-}Sig_A(m)$ et d'autre part la promesse $PCS_B(m,A,T)$ de B, à l'aide d'un tiers de confiance.

Les deux participants peuvent éventuellement ne pas suivre complètement le protocole pour tenter d'obtenir un avantage sur l'autre. Ces protocoles de signature de contrat doivent donc assurer qu'aucun des participants n'est avantagé par rapport à l'autre. Plusieurs définitions formelles ont été proposées [SM00b, SM01, CKS01, KR02, CKS01]. En particulier, S. Kremer et J.-F. Raskin utilisent une définition basée sur la théorie des jeux : un protocole est *équitable* pour le participant Alice si le participant Bob n'est pas de stratégie gagnante pour obtenir le contrat signé par Alice à moins qu'Alice ait elle aussi une stratégie gagnante pour obtenir le contrat signé par Bob.

1.3.3.5 Disponibilité

Il est également important de pouvoir établir que le protocole permet une communication entre ses participants. Ainsi, pour les protocoles d'échange de clefs de session, il faut vérifier que si Alice a demandé à un serveur d'établir une communication entre elle et Bob alors Alice et Bob finiront par recevoir une clef commune. Cette propriété est proche des propriétés de « vivacité ». Bien sûr, pour vérifier une telle propriété, il faut supposer que l'intrus n'est pas capable de détourner les messages mais seulement un nombre limité [RSG+00, PW94].

Dans cette thèse, nous traiterons plus particulièrement les propriétés de secret et d'authentification.

1.4 Difficultés de la vérification

La vérification des protocoles cryptographiques est un cas particulier de model-checking où les systèmes considérés sont des protocoles cryptographiques dans un réseau hostile et les propriétés à vérifier sont celles énoncées aux paragraphes précédents (secret, authentification, anonymat). Les difficultés de la vérification tiennent dans le caractère non borné des paramètres du système à vérifier :

- le nombre de sessions n'est pas borné ;
- le nombre de participants n'est pas borné ;
- les messages sont de taille arbitraire ;
- à chaque étape, n'importe quel message de la connaissance de l'intrus peut être envoyé : le système est à branchement infini ;
- l'intrus peut générer un nombre arbitraire de nouvelles clefs et de nouveaux nonces.

De plus, les primitives cryptographiques ont des propriétés algébriques : associativité de la paire, propriété de l'exponentielle utilisée dans les chiffrements RSA, propriétés du « ou » exclusif, *etc.* Pour modéliser ces propriétés, on peut ajouter les théories équationnelles correspondantes [AF01]. Mais ces théories équationnelles augmentent considérablement la difficulté de la vérification. Nous verrons ainsi au chapitre 8 que la décision du secret devient non élémentaire au lieu de 3-EXPTIME pour une classe restreinte de protocoles lorsqu'on ajoute le « ou » exclusif. H. Comon-Lundh et V. Shmatikov [CLS03] ont montré que la décision du secret pour un nombre de sessions borné devient NEXPTIME au lieu de NP-complet toujours en ajoutant le « ou » exclusif.

1.5 Modèles

Pour décrire les protocoles, nous avons jusqu'ici utilisé la représentation intuitive employée dans [CJ97]. Cette présentation est en fait très ambiguë car seul le déroulement *normal* du protocole est décrit. Pour le protocole de Needham-Schroeder, on ne sait pas si le deuxième agent teste si le nonce reçu est bien un nonce, ce qu'il fait si le message n'est pas de la forme attendue ni surtout ce qu'est exactement la « forme attendue » par l'agent. D'autres ambiguïtés de ces spécifications informelles sont décrites par exemple dans [Aba00]. L'étude des protocoles cryptographiques commence donc par une première étape de modélisation. Depuis quatre années environ, de nombreux modèles dédiés aux protocoles se sont développés. Nous allons en présenter une liste que nous espérons représentative. Certains modèles seront présentés plus précisément dans les chapitres 3, 4 et 5.

1.5.1 Modèles de traces

Les modèles de traces représentent les protocoles et l'intrus par des règles d'inférence. Ces règles décrivent les transitions possibles entre *traces*. Les traces contiennent en général l'ensemble des messages échangés sur le réseau et éventuellement les états locaux des participants ou des annotations décrivant explicitement les transitions.

Modèle de D. Dolev et A.C. Yao L'un des premiers modèles est celui développé par D. Dolev *et al.* [DY81, DEK83]. Les protocoles sont décrits par des règles de réécriture de mots ; un mot s est secret s'il n'est pas accessible par réécriture. M. Merritt *et al.* [RDM82, Mer83] ont développé pendant la même période un modèle où les messages sont également représentés sous forme de mots.

Modèle de Paulson Dans le modèle de L.C. Paulson [Pau97b, Pau98, Pau97a], l'*historique* contient l'ensemble des messages envoyés et les règles du protocole sont représentées par des règles de transitions entre historiques. L.C. Paulson introduit les opérateurs Synth et Analz qui calculent, à partir d'un ensemble de messages, l'ensemble des messages obtenus respectivement par analyse ou par synthèse. J. Millen et H. Rueß [MR00, RM00] ont développé un modèle très proche de celui de L.C. Paulson où les secrets à vérifier sont décrits au fur et à mesure du déroulement du protocole, au lieu de formuler une propriété indépendante du protocole.

Strand spaces Le modèle des strand spaces [THG99, GT01] est un modèle utilisé surtout par la communauté américaine. Il s'agit d'un modèle de traces où les transitions entre messages sont exprimées explicitement dans chaque historique. Cela permet en particulier de formaliser la représentation graphique « intuitive » des déroulements de protocoles. Par contre, cela rend les définitions plus difficiles à manipuler. D'autre part, J. Thayer *et al.* introduit la notion d'*idéal* qui est une surapproximation de l'ensemble des messages à protéger pour garantir qu'une donnée ne sera pas connue de l'intrus.

Règles de réécriture Plusieurs modèles représentent les règles des protocoles et le pouvoir de l'intrus par des règles de réécriture sur des termes. Citons principalement le MultiSet Rewriting (MSR) développé par J. Mitchell *et al.* [CDL+99, BCJS02], les règles de réécriture utilisées par l'outil Casrul [RT01] et le modèle basé sur la logique linéaire de K. Compton et S. Dexter [CD99]. La principale difficulté de cette modélisation est d'exprimer les nonces de manière fidèle et d'empêcher la répétition de règles déjà jouées.

Clauses de Horn Une variante de la modélisation sous forme de réécriture est la modélisation sous forme de clauses de Horn [Bla01, CLC03a]. Ces deux formes de modélisation sont très proches et les difficultés rencontrées sont également similaires. La modélisation sous forme de clauses de Horn présente cependant l'avantage de permettre l'utilisation des techniques déjà développées sur les clauses comme les stratégies de résolution. Aussi, plusieurs des résultats présentés dans la thèse seront énoncés dans le cadre d'un modèle sous forme de clauses de Horn.

1.5.2 Algèbres de processus

Un autre type de modélisation est la représentation des protocoles par des processus. Cela correspond à une modélisation plus proche de l'implémentation des protocoles. En effet, chaque rôle est représenté par un processus indépendant des processus représentant les autres rôles. Ces algèbres de processus (comme CSP [Sch97] ou le spi-calcul [AG97]) permettent en général les opérations suivantes :

$$\begin{array}{rll} P := & \mathbf{0} & \text{processus nul} \\ & \mid (\nu n).P & \text{création de nonces} \\ & \mid \bar{p} < m > .P & \text{envoi de messages} \\ & \mid p(x).P & \text{réception de messages} \\ & \mid P_1 \mid P_2 & \text{composition parallèle} \\ & \mid !P & \text{réplication} \\ & \ldots & \end{array}$$

La principale différence entre les différents modèles utilisant des algèbres de processus est la modélisation de la propriété de sécurité. La plupart des modèles [AL00, ALV02, Sch96a, Sch97, Bor01] ramènent les propriétés de sécurité à une propriété d'accessibilité de la forme : $P \xrightarrow{*} err$. Quelques autres [AG98, BDNP99] représentent le secret par exemple sous forme d'une *équivalence obervationnelle*. Intuitivement, un protocole ne révèle rien de la donnée s s'il est indistinguable du même protocole avec la donnée s'.

1.6 Trois approches pour la vérification

Dans le cadre cette thèse, nous nous limitons à l'étude des propriétés de secret et d'authentification. Dans ce domaine, les recherches s'articulent sur trois axes : d'une part la recherche d'attaques à l'aide d'outils de vérification, d'autre part la preuve des propriétés de sécurité (principalement le secret) par abstraction, et enfin la recherche de classes décidables.

1.6.1 Recherche d'attaques

Les premiers résultats en vérification de protocoles cryptographiques ont été des découvertes d'attaques à l'aide d'outils d'analyse de protocoles [Mea00b] comme l'attaque trouvée par G. Lowe [Low96] sur le protocole de Needham-Schroeder à clefs publiques. Nous présentons ainsi plusieurs outils de recherche d'attaques comme Casper [Low97a] et Casrul [JRV00] au chapitre 7.

1.6.2 Preuve par abstraction

Lorsqu'un outil ne trouve pas d'attaques sur un protocole cryptographique, il n'y a aucune assurance que le protocole soit correct pour autant. En effet, l'espace de recherche d'attaques des outils n'est pas complet : les outils recherchant des attaques travaillent souvent pour un nombre de sessions borné. Si un outil ne trouve pas d'attaque pour deux ou trois sessions, il se peut cependant qu'il y ait une attaque utilisant quatre sessions et ainsi de suite. C'est pourquoi il faut également développer des méthodes de *preuves* des protocoles cryptographiques. Comme pour la recherche d'attaque, ces méthodes de preuve se limitent principalement aux propriétés de secret. Cependant, la preuve d'une propriété de sécurité demande une justification plus rigoureuse de la modélisation. En effet, la propriété sera prouvée dans le modèle considéré donc il faut pouvoir en justifier les limites. Ainsi, M. Burrows *et al.* ont prouvé [BAN89] que le protocole de Needham-Schroeder était correct alors que G. Lowe a ensuite trouvé une attaque, tout simplement parce que leurs modèles étaient différents. D'autre part, même lorsqu'on démontre qu'un protocole vérifie la propriété voulue, on ne vérifie pas le protocole tel qu'il est implémenté mais seulement l'une de ses spécifications.

De nombreuses abstractions sont possibles pour prouver le secret des protocoles. Nous allons en donner quelques exemples. Ainsi, Y. Lakhnech *et al.* [BLP03] particularisent une session du protocole où un agent honnête communique avec lui-même et projettent les nonces des autres sessions sur un unique nonce. Cette abstraction est correcte : si on prouve que le protocole est secret dans le modèle abstrait, alors il est secret dans le modèle concret. Par contre, cette abstraction n'est pas complète. Elle permet cependant de montrer le secret de nombreux protocoles de [CJ97] et elle a été implémentée dans l'outil Hermes.

B. Blanchet [Bla01] modélise les nonces par une fonction des messages reçus. Une telle abstraction est à nouveau correcte mais peut introduire des « fausses attaques ».

Pour prouver le secret ou l'authentification des protocoles, J. Heather et S. Schneider [HS00] introduisent la notion de fonctions de *rang* qui calculent une approximation des messages connus et inconnus de l'intrus.

Certaines de ces abstractions (comme l'utilisation d'une fonction de rang) doivent être adaptées à chaque protocole, la preuve du protocole n'est donc pas automatique. Les autres abstractions ne varient pas d'un protocole à l'autre et ont été construites afin de pouvoir vérifier la plupart des protocoles cryptographiques existants. Cependant, il n'y a aucune assurance que ces abstractions permettent de prouver de nouveaux protocoles. Vérifier les protocoles de manière complètement automatique est un enjeu important car d'une part de nouveaux protocoles sont inventés régulièrement et d'autre part, chaque protocole possède de multiples variantes adaptées aux spécificités de l'application dans laquelle il est utilisé. Cela motive la recherche de classes décidables, qui permettent également de mieux comprendre les limites théoriques de la vérification des protocoles cryptographiques.

1.6.3 Classes décidables et indécidables

Tout d'abord, de nombreux résultats sont négatifs. J. Mitchell *et al.* [DLMS99] ont montré que le secret est indécidable pour des protocoles avec nonces, avec des messages de taille bornée et un nombre de sessions non borné. R. Amadio et W. Charatonik [AC02] ont raffiné ce résultat à des protocoles ne possédant qu'une seule primitive cryptographique : le chiffrement. Par ailleurs, nous avons montré [CCM01] que même pour des protocoles sans nonces, le secret est indécidable pour un nombre non borné de sessions.

Dans le cadre d'un nombre borné de sessions, les résultats sont maintenant assez satisfaisants. Citons tout d'abord le résultat de M. Rusinowitch et M. Turuani [RT01] qui montrent dans un cadre très général (clefs composées, chiffrement symétrique et asymétrique) que la décision du secret est un problème co-NP-complet. Dans le contexte des algèbres de processus, R. Amadio *et al.* [AL00, ALV02] montrent un résultat similaire mais sans modéliser les clefs composées. Des résultats de décidabilité très proches sont établis, toujours pour des processus sans réplication par [Bor01, FA01, MS01]. H. Huttel [Hut02] montre également la décidabilité du secret pour des processus sans réplication mais pour une propriété d'équivalence observationnelle. Enfin, ces résultats sont étendus à des protocoles comportant le « ou » exclusif [CLS03].

Dans le cadre d'un nombre de sessions non borné, les restrictions sur les classes de protocoles sont encore très importantes. Ainsi, si on considère des protocoles sans nonces, J. Mitchell *et al.* [DLMS99] ont montré que le secret est décidable si on suppose de plus que la profondeur des messages est bornée. D'autre part, nous montrons au chapitre 8 que le secret des protocoles sans nonces est décidable si on suppose que pour chaque transition, au plus un message arbitraire est copié (mais un nombre arbitraire de noms d'agents peut être copié).

G. Lowe [Low98] a également présenté un résultat de décision pour des protocoles cryptographiques avec des nonces. Nous en rappelons les principales restrictions au paragraphe 6.2.3 du chapitre 6. En particulier, les règles des protocoles doivent être entièrement typées, par des types atomiques et disjoints.

1.7 Points faibles

La vérification des protocoles cryptographiques se heurte à un premier problème : une trop grande diversité des modèles. En effet, chaque auteur a son modèle favori, qui utilise souvent des constructions très particulières. Ces différents modèles sont le plus souvent incomparables du point de vue de l'expressivité : certains permettent les clefs composées, d'autres non, d'autres encore permettent le chiffrement asymétrique, *etc.* De plus, les propriétés exprimées sur les protocoles sont dépendantes du modèle, voire même du protocole à vérifier. Nous avons ainsi évoqué au paragraphe 1.3.3 la différence entre les propriétés de secret exprimées sous forme d'équivalence observationnelle et les propriétés de secret exprimées sous forme d'accessibilité. Mais en fait, au sein même des modèles traitant uniquement de propriétés d'accessibilité, les propriétés ne sont pas identiques et les liens entre elles ne sont pas clairs. Par conséquent, les résultats obtenus dans un modèle donné ne sont jamais directement utilisables dans les autres modèles.

Concernant la recherche de classes décidables, nous avons vu au paragraphe précédent que les résultats étaient satisfaisants dans le cadre d'un nombre borné de sessions. À l'inverse, aucune classe décidable englobant un nombre significatif de protocoles réels n'a encore été exhibée. Par ailleurs, l'hypothèse du chiffrement parfait (que nous allons développer au chapitre 2) est une hypothèse très forte. L. Paulson a par exemple montré que le protocole de Bull [Pau97a] préserve le secret sous l'hypothèse du chiffrement parfait alors que des attaques sont possibles si on prend en compte les propriétés algébriques du « ou » exclusif utilisé dans le chiffrement. Il serait donc intéressant d'établir des résultats de décidabilité en tenant compte des propriétés algébriques de certains opérateurs comme le « ou » exclusif ou l'exponentielle.

Enfin, au moment où cette thèse a commencé, de nombreux logiciels de recherche d'attaques existaient, comme l'outil FDR qui a permis à G. Lowe [Low96] de trouver la faille du protocole de Needham-Schroeder par exemple, mais aucun logiciel de *preuve* des protocoles.

1.8 Contenu de la thèse

Les contributions de cette thèse s'articulent en trois axes.

Sur le plan de la modélisation, nous introduisons un nouveau modèle très général de protocoles cryptographiques sous forme de clauses de Horn. Ce modèle a deux principaux avantages. D'une part, il permet de représenter les protocoles dans toute leur généralité : avec un nombre arbitraire de sessions et de participants, avec des clefs composées et du chiffrement symétrique ou asymétrique. Nous montrons en particulier que tout protocole du modèle de J. Millen et H. Rueß [MR00] peut s'exprimer dans notre modèle sous forme de clauses. Il permet également de représenter une famille de propriétés comprenant le secret et l'authentification. D'autre part, nous pouvons réutiliser sur ce modèle toutes les techniques classiques développées en logique du premier ordre : stratégie de résolution, arbre sémantique, *etc.* Nous avons ainsi obtenu un résultat de réduction sur le nombre d'agents nécessaires à

une attaque.
Par ailleurs, nous nous sommes intéressés à un modèle très différent : le spi-calcul. Nous montrons que l'équivalence observationnelle du spi-calcul (qui permet d'exprimer le secret comme l'indistingabilité de deux processus) se ramène à une forme de bisimulation utilisant une équivalence statique pour un système de transitions plus proche des modèles de trace. Ce résultat a deux principaux intérêts. D'un point de vue conceptuel, il permet de mieux comprendre le lien entre le secret exprimé sous forme d'équivalence observationnelle et sous forme d'accessibilité. En outre, il permet de développer une méthode de preuve de l'équivalence observationnelle entre processus. Nous prouvons dans ce cadre le secret du protocole de Needham-Schroeder-Lowe pour un nombre arbitraire de sessions. À notre connaissance, c'est la première fois que la preuve de ce protocole est faite pour une propriété d'équivalence observationnelle.

Sur le plan de la vérification pratique des protocoles cryptographiques, nous proposons un algorithme de vérification du secret pour des protocoles exprimés dans le modèle de J. Millen et H. Rueß. Cet algorithme a été implémenté et a permis la vérification de nombreux protocoles de [CJ97]. Ce fut l'un des premiers logiciels dédiés à la *preuve* des protocoles.
D'autre part, nous avons montré un résultat de réduction sur le nombre d'agents pour une classe de protocoles très générale et pour une large famille de propriétés. Nous montrons ainsi qu'en général seuls deux agents suffisent (l'un honnête, l'autre malhonnête) pour trouver une attaque. Ce résultat a des applications pratiques et théoriques. D'un point de vue pratique, nous montrons qu'il suffit de considérer un nombre restreint d'agents distincts. Notons que cette hypothèse était en général utilisée en pratique mais sans justification. D'un point de vue théorique, nous démontrons la complétude de certaines abstractions [Pau97a, BLP03].

Enfin, nous apportons quelques réponses sur le plan de la recherche de classes décidables pour les protocoles cryptographiques avec un nombre arbitraire de sessions. Nous avons en particulier considéré la classe des protocoles qui ne copient qu'au plus un message arbitraire par transition. Cette classe paraît pertinente, d'une part parce que de très nombreux protocoles de [CJ97] satisfont naturellement cette hypothèse et d'autre part parce que le secret est indécidable dès que deux copies arbitraires sont autorisées, même pour des protocoles sans nonces. Ainsi, nous montrons que le secret et l'authentification sont décidables en 3-EXPTIME pour des protocoles sans nonces et ne permettant qu'une seule copie arbitraire de message par transition. De plus, nous avons vu précédemment que les protocoles cryptographiques utilisent des primitives comme l'exponentielle ou le « ou » exclusif, qui ont des propriétés algébriques particulières. Modéliser de telles propriétés permet de vérifier des protocoles plus proches de l'implémentation réelle. C'est pourquoi nous étendons notre résultat de décision aux protocoles utilisant le « ou » exclusif : le secret et l'authentification sont décidables pour des protocoles avec « ou » exclusif sans nonces et ne permettant qu'une seule copie arbitraire de message par transition mais pour un nombre arbitraire de sessions. La complexité de la vérification devient alors non élémentaire. À notre connaissance, il s'agit du premier résultat de décidabilité des protocoles avec « ou » exclusif pour un nombre non borné de sessions. Pour ces deux résultats de décision, nous introduisons de nouveaux fragments décidables de la logique du premier ordre, généralisant sur certains points l'une des classes présentées dans [FLHT01].

1.9 Plan de la thèse

Cette thèse est présentée en trois parties. D'une part, l'étude et la comparaison des modèles de protocoles cryptographiques, d'autre part une présentation de résultats utiles pour la vérification pratique des protocoles et enfin, nous présentons des classes décidables de protocoles.

Modèles

Dans la partie **Modèles** de la thèse, nous présentons plusieurs modèles représentatifs des modèles utilisés pour les protocoles cryptographiques. Cela nous amènera à surcharger les définitions de *message*, *nonce*, *protocole*, *etc.*, nous préciserons le contexte lorsqu'il peut y avoir ambiguïté.

Dans le **chapitre 2**, nous décrivons informellement les hypothèses couramment employées pour la modélisation des protocoles cryptographiques.

Dans le **chapitre 3**, nous introduisons un modèle de protocoles sous forme de clauses de Horn, permettant de représenter une très large classe de protocoles.

Au **chapitre 4**, nous rappelons le modèle défini par J. Millen et H. Rueß et nous montrons que tout protocole de ce modèle peut être représenté par un protocole sous forme de clauses de Horn de telle manière que le premier protocole préserve le secret si et seulement si le second préserve également le secret.

Au **chapitre 5**, nous rappelons la définition du spi-calcul tel qu'il a été introduit par M. Abadi et A. Gordon [AG98]. Nous introduisons ensuite, à la manière de M. Boreale *et al.* [BDNP99], un système de transitions pour des processus dans un environnement de telle manière que l'équivalence observationnelle de deux processus du spi-calcul soit équivalente à la bisimulation des deux processus dans des environnements statiquement équivalents. Nous appliquons ce résultat à la preuve de l'équivalence observationnelle. Nous montrons en particulier le secret du protocole de Needham-Schroeder-Lowe pour un nombre non borné de sessions.

Vérification pratique des protocoles cryptographiques

Nous montrons au **chapitre 6** que seulement deux agents suffisent pour trouver une attaque si on suppose qu'un agent peut se parler à lui-même. Ce résultat est valable sans restriction sur les protocoles considérés et pour toute propriété de secret ou d'authentification. Nous généralisons ensuite ce résultat au cas où un agent ne peut pas se parler à lui-même.

Au **chapitre 7**, nous introduisons une procédure de décision pour le secret des protocoles cryptographiques du modèle de J. Millen et H. Rueß. Cette procédure est correcte mais non complète et peut éventuellement ne pas terminer. Nous avons implémenté cette procédure dans l'outil Securify qui a permis de vérifier de nombreux protocoles de [CJ97].

Classes décidables

La dernière partie de cette thèse est constitué d'un unique chapitre (**chapitre 8**) dans lequel sont introduites deux classes décidables de protocoles cryptographiques. Nous montrons en effet que le secret et l'authentification sont décidables pour des protocoles sans nonces et ne permettant qu'une copie arbitraire de message par transition. Nous étendons ensuite ce résultat aux protocoles avec « ou » exclusif.

Pour montrer ces résultats nous introduisons de nouveaux fragments décidables de la logique du premier ordre.

Les chapitres 3, 4 et 6 ont fait l'objet des articles [CLC03b, CLC02]. Les chapitres 4 et 7 ont fait l'objet des articles [CMR01, Cor02a]. Le chapitre 5 a fait l'objet de l'article [Cor02b]. Enfin, le chapitre 8 a fait l'objet de l'article [CLC03a] et en partie des articles [CCM01, CLC03c].

Première partie

Modèles de protocoles cryptographiques

CHAPITRE 2

Principales hypothèses de la modélisation

La modélisation des protocoles cryptographiques fait abstraction de nombreuses particularités des protocoles. Par exemple, la plupart des modèles reposent sur l'hypothèse du « chiffrement parfait ». De nombreuses autres hypothèses peuvent être faites, souvent de manière implicite lors de la définition formelle du modèle.

L'objet de ce chapitre est de dresser une liste des différentes hypothèses et d'expliquer la manière dont elles sont traitées dans les modèles. Nous nous exprimerons en termes assez intuitifs mais nous indiquerons dans les chapitres suivants – qui présentent des modèles de protocoles cryptographiques – où interviennent précisément ces hypothèses.

2.1 Primitives cryptographiques

Nous présentons ici la modélisation courante du *chiffrement*, de la *concaténation* et des *nonces*.

2.1.1 Chiffrement

L'étude des protocoles cryptographiques repose dans la plupart des cas sur le fait suivant : un intrus ne peut déchiffrer un message m chiffré avec une clef k que s'il possède l'inverse de cette clef. Mais les hypothèses induites par le modèle sont souvent encore plus fortes que cela. Par exemple, on suppose que des messages distincts chiffrés avec des clefs distinctes donnent deux chiffrés distincts, alors que, dans la réalité, même si la probabilité est faible, ces deux chiffrés pourraient être identiques. Ce type d'hypothèse conduit à modéliser les messages sous forme de termes. En particulier, supposons que m et m' représentent des messages et que k et k' représentent des clefs. Les deux messages chiffrés $\{m\}_k$ et $\{m'\}_{k'}$ sont égaux si et seulement si $m = m'$ et $k = k'$. Une telle représentation sous forme de termes implique, de plus, que le chiffré d'un chiffré est distinct du message initial, même si la clef est symétrique :

$$\{\{m\}_k\}_k \neq m.$$

Toutes ces hypothèses sont regroupées sous le nom « *hypothèse du chiffrement parfait* ».

Bien sûr, cette hypothèse ne correspond pas à la réalité : certains algorithmes de chiffrement permettent à l'intrus de connaître un message m à partir de son chiffré $\{m\}_k$, dès qu'il a une connaissance partielle de m (le début du message, par exemple) ou s'il possède d'autres messages chiffrés avec la même clef k. Cependant, comme nous l'avons vu au chapitre précédent, de nombreuses attaques de protocoles ne reposent absolument pas sur la façon dont sont chiffrés les messages mais consistent tout simplement à rejouer un vieux message ou à mélanger plusieurs sessions. Il s'agit donc de vérifier les protocoles cryptographiques sans tenir compte des éventuelles failles dues au chiffrement. Il reste ensuite aux cryptologues la tâche d'assurer la quasi-inviolabilité d'un message chiffré.

Plusieurs auteurs ébauchent un lien entre ces deux approches des protocoles cryptographiques avec d'un côté, les modèles « idéaux » qui traitent les messages comme des termes et de l'autre, les modèles probabilistes qui supposent que l'on peut déchiffrer un message avec une certaine probabilité. M. Abadi et P. Rogaway [AR00] montrent ainsi que, sous certaines hypothèses, le secret dans un modèle « idéal » implique le secret dans le modèle probabiliste. J.D. Guttman *et al.* [GTZ01] ont également montré que si un protocole était correct pour un modèle idéal, alors ce protocole était correct avec un certaine probabilité, calculée en fonction du protocole, dans le modèle probabiliste. M. Backes *et al.* [BPW03] fournissent une bibliothèque de correspondance entre les deux modèles pour différentes primitives cryptographiques comme le chiffrement à clefs publiques ou les nonces.

Enfin, il existe deux grandes classes d'algorithme de chiffrement : les algorithmes de chiffrement symétrique et les algorithmes de chiffrement asymétrique. Les premiers utilisent la même clef que les algorithmes de déchiffrement correspondants. Pour les seconds, les clefs utilisées lors du chiffrement et du déchiffrement sont distinctes. Dans le cadre du chiffrement parfait, il semble naturel de modéliser ces deux types d'algorithmes par des symboles de fonction différents comme cela est fait dans [RT01, GL01] par exemple. Cependant, de nombreux modèles [MR00, RM00, Pau98, GL01, THG99] utilisent un même symbole de fonction. L'algorithme utilisé dépend alors implicitement de la forme de la clef : si la clef a la forme d'une clef asymétrique - $\mathsf{pub}(a)$ pour représenter la clef publique de a par exemple - alors l'algorithme de chiffrement utilisé pour crypter le message $\{m\}_k$ est asymétrique, sinon l'algorithme est supposé symétrique. Cette dépendance apparaît en général lors de la définition de l'inverse d'une clef. Notons que lorsqu'un modèle utilise le même symbole de fonction pour les deux types de chiffrement, alors on ne peut pas représenter, par exemple, le message m chiffré avec la clef publique de a par un algorithme de chiffrement symétrique. Il est tout de même réaliste d'utiliser un unique symbole de chiffrement car les clefs symétriques et asymétriques ont des formats (longueurs, ...) très différents.

2.1.2 Concaténation

La deuxième fonction utilisée dans les protocoles cryptographiques est la concaténation. Elle est modélisée par un symbole fonctionnel binaire, souvent le symbole $< \_, \_ >$, appelé aussi paire. Comme les messages sont des termes, le message $< m_1, < m_2, m_3 >>$ est distinct du message $<< m_1, m_2 >, m_3 >$. Pour modéliser l'associativité de la concaténation, il faudrait, par exemple, ajouter la théorie équationnelle de l'associativité du symbole $< \_, \_ >$. De la même manière, il est pertinent de modéliser le « ou » exclusif utilisé comme prétraitement du chiffrement ou de modéliser les propriétés de l'exponentielle, car ces notions sont souvent utilisées dans les algorithmes de chiffrement. L'ajout de théories équationnelles permet ainsi de mieux représenter les propriétés algébriques de certaines primitives cryptographiques. Mais cela rend nettement plus difficile la vérification des protocoles [GLV02, CLS03].

2.1.3 Nonces

Dans les protocoles, les nonces sont des données engendrées aléatoirement. La probabilité d'engendrer deux nonces identiques étant très faible, on choisit de modéliser les nonces comme des données toutes distinctes. Lorsqu'on considère un nombre fini de sessions, il suffit alors de remplacer les nonces par des constantes distinctes deux à deux. Mais dès que l'on considère un nombre non borné de sessions, la modélisation des nonces est plus délicate et conduit très rapidement à l'indécidabilité du secret d'un protocole [DLMS99, AC02].

Dans le cadre d'un nombre non borné de sessions, beaucoup de modèles représentent donc les nonces par un nombre fini de constantes [CCM01, BLP03]. Une autre possibilité, exposée par C. Weidenbach [Wei99] et de B. Blanchet [Bla01], consiste à modéliser les nonces par une fonction des messages reçus par l'agent dans la session en cours. Dans les deux cas, on ne peut éviter l'inconvénient suivant : deux nonces engendrés à deux sessions différentes peuvent être égaux. Cependant, de telles abstractions sont correctes pour la *preuve* du secret ou de l'authentification. En revanche, elles peuvent conduire à la découverte de « fausses » attaques, basées sur le renvoi d'un nonce déjà engendré, ce qui n'est pas possible dans la réalité. Nous verrons ainsi un exemple au paragraphe 8.1.1.2 du chapitre 8.

2.1.4 Typage

Pour décrire les actions d'un agent, on utilise souvent des règles avec variables. Ainsi, pour dire qu'un agent reçoit la concaténation d'un nom d'agent et d'un nonce, on dira qu'il reçoit une instance du terme $< a, n >$ où a est une variable de *type* agent, c'est-à-dire à valeurs dans l'ensemble des noms d'agents tandis que n est une variable de type nonce. Un tel typage n'est pas innocent, il représente la capacité d'un agent à distinguer différentes sortes de messages. Ainsi, on suppose presque toujours que les agents savent distinguer le nom d'un agent d'un message quelconque. On peut en effet imaginer qu'un tel nom est représenté par une suite de bits moins longue que les autres, ou même que les agents connaissent déjà à l'avance les noms de tous les participants potentiels. Par contre, dans la réalité, il n'est pas évident qu'un agent puisse distinguer un nonce (qui est une suite de bits) d'un message composé (qui est une autre suite de bits). De nombreuses attaques (dites « de type ») reposent justement sur la confusion d'un nonce et d'un message composé [Syv94, CJ97].

Cependant, typer les messages permet de vérifier plus facilement les protocoles. Ainsi, G. Lowe [Low98] a montré que si les messages sont entièrement typés et si les types utilisés sont atomiques et disjoints (agents, nonces, clefs, ...), alors, sous réserve d'hypothèses supplémentaires que nous n'expliciterons pas, le secret est décidable. En particulier, il montre que l'on peut borner à l'avance le nombre de sessions nécessaires à une attaque.

De la même façon, de nombreux outils de vérification [ABB+02, CMR01, BLP03, Cor02a] supposent que les messages sont typés et lorsque ce n'est pas le cas, ils n'arrivent pas à prouver le secret.

Enfin, J. Heather, G. Lowe et S. Schneider [HLS00] ont prouvé que le typage des nonces est « implémentable ». Plus précisément, ils ont montré que l'on peut renforcer chaque protocole par un étiquetage qui donne le type de chaque donnée. Cet étiquetage comporte suffisamment de redondance pour permettre aux agents de détecter des confusions de type. Cependant, les protocoles implémentés actuellement n'ont pas été transformés de cette manière. Il importe donc de les vérifier aussi sans faire d'hypothèses supplémentaires.

2.2 Intrus

Dans le cadre du spi-calcul, l'intrus est modélisé par n'importe quel processus descriptible dans la syntaxe. A l'inverse, dans la plupart des autres modèles, les actions de l'intrus sont décrites par un ensemble succinct de règles où apparaissent la capacité de l'intrus à déchiffrer des messages lorsqu'il a l'inverse de la clef ou sa capacité à former de nouveaux messages. De tels modèles sont souvent appelés « modèles à la Dolev-Yao » en référence aux premiers modèles de D. Dolev et A.C. Yao [DY81, DEK83]. Notons que dans ces modèles, la capacité de calcul et de mémoire de l'intrus n'est pas bornée. Ceci contraste avec le cadre des modèles probabilistes dans lequel on suppose que l'intrus a une capacité de calcul bornée et où l'on cherche à montrer que les calculs nécessaires au déchiffrement d'un message dépassent la capacité de calcul de l'intrus.

D'autre part, pourquoi ne considérer qu'un seul intrus et non plusieurs, agissant éventuellement de concert ? En fait, cela revient au même si chaque intrus dispose d'un nombre arbitraire d'identités et des clefs privées correspondantes : ce que deux intrus pourraient apprendre et calculer, un seul intrus peut le faire seul puisque ni sa capacité de calcul ni la taille de sa mémoire ne sont bornées.

2.3 Agents

Les agents testent et stockent des parties de messages en fonction des règles du protocole. Ils peuvent être honnêtes ou malhonnêtes.

2.3.1 Honnêtes/malhonnêtes

Il peut paraître curieux de distinguer agents malhonnêtes et intrus. Cette distinction permet de bien séparer la connaissance apportée par le jeu du protocole lui-même et celle apportée par la capacité d'analyse et de synthèse donnée à l'intrus. Les agents malhonnêtes peuvent être considérés comme des personnes qui collaborent avec l'intrus ou bien comme des identités que l'intrus utilise pour participer au protocole. Dans le premier cas, l'intrus pourrait éventuellement ne pas connaître les clefs des agents malhonnêtes mais seulement les messages que ces agents veulent bien lui transmettre. La grande majorité des modèles [DY81, Low98, THG99, GK00, MR00] choisit de prendre l'hypothèse la plus forte pour l'intrus, à savoir qu'il connaît toutes les clefs privées des agents malhonnêtes. Mais on peut imaginer un protocole entre trois participants A, B et C dans lequel l'agent C souhaite collaborer avec l'intrus afin que celui-ci puisse connaître un secret partagé entre A et B sans que toutefois l'intrus ait accès à un secret partagé entre C et B. Dans un contexte différent comme celui de la signature de contrat [SM00a, KR01, KR02, CKS01], un agent malhonnête est tout simplement un agent qui ne suit pas le protocole.

2.3.2 Mémoire

Dans la plupart des modèles de vérification des protocoles cryptographiques, on choisit de ne donner aux agents et au serveur qu'une mémoire locale à la session. Cela correspond à l'implémentation la plus économique : il serait très coûteux pour un serveur (en mémoire et surtout en temps) de stocker l'ensemble des messages reçus. En effet, conserver trop d'informations augmente les temps d'accès mémoire et donc les temps de réponse. Ceci va à l'encontre de la volonté de rendre un protocole transparent pour l'utilisateur.

D'autres problèmes de sécurité reposent au contraire sur l'étude de l'historique des sessions passées ou en cours. Ainsi « l'audit de log » [And80, Mou97] consiste à rechercher des motifs dans des fichiers répertoriant les actions tentées. Garder en mémoire un historique permet en particulier d'éviter les problèmes de rejeu mais comme nous l'avons dit plus haut, cette solution n'est pas toujours possible techniquement ou économiquement.

C'est pourquoi la grande majorité des modèles considèrent des agents avec une mémoire locale, hypothèse qui correspond le mieux à l'implémentation des protocoles.

CHAPITRE 3

Protocoles cryptographiques sous forme de clauses de Horn

Chaque article ou presque, traitant des protocoles cryptographiques, présente un modèle différent. Nous aurions voulu éviter d'en introduire un de plus mais nous souhaitions disposer d'un modèle suffisamment général pour exprimer les différents protocoles et leurs primitives cryptographiques (clefs publiques, clefs composées, hachage) ainsi que les propriétés à prouver (secret, authentification). La description des protocoles cryptographiques sous forme de clauses présente le double avantage d'être très générale et de permettre l'utilisation d'outils de vérification et de résultats classiques sur le sujet comme l'existence d'un plus petit modèle pour les clauses de Horn ou les stratégies de résolution. Ce modèle nous a ainsi permis d'obtenir un résultat de réduction (chapitre 6) sur le nombre d'agents à considérer pour des propriétés de secret et d'authentification. Il est comparé à d'autres modèles déjà existants dans les chapitres suivants. Il généralise en particulier le modèle développé par J. Millen et H. Rueß [MR00].

Dans ce chapitre, nous commençons par un rappel du formalisme relatif aux clauses et aux systèmes de contraintes. Nous expliquons ensuite comment décrire un protocole par un ensemble de clauses contraintes. Cet ensemble de clauses se divise en deux parties : une partie des clauses, dite *indépendante*, ne varie pas d'un protocole à l'autre. Elle permet la description des primitives cryptographiques et du pouvoir de l'intrus. L'autre partie des clauses, dite *dépendante du protocole* représente les règles du protocole considéré.

3.1 Préliminaires

Nous rappelons ici quelques résultats classiques sur les clauses et les systèmes de contraintes.

3.1.1 Clauses

Nous reprenons la terminologie et les définitions du chapitre « Logic Programming », écrit par K. R. Apt dans *Handbook of Theoretical Computer Science* [Apt90]. Ainsi, nous supposons définies les notions de *langage du premier ordre*, *terme*, *terme clos*. En particulier, les symboles de fonctions ont une arité fixe et l'ordre des fils dans les termes est significatif. Si $\mathcal{F}$ est un alphabet, on note $T(\mathcal{F})$ l'ensemble des termes sur cet alphabet. Un terme est dit *linéaire* si chaque variable n'apparaît qu'au plus une fois dans le terme. On définit aussi ([CDG+97]) un *contexte* comme un terme linéaire de $T(\mathcal{F} \cup X_n)$ où X_n est un ensemble de n variables. L'ensemble des contextes à une variable sur un alphabet $\mathcal{F}$ est noté $\mathcal{C}(\mathcal{F})$. Si C est un contexte de $\mathcal{C}(\mathcal{F})$ et u un terme de $T(\mathcal{F})$, on note $C[u]$ le terme dans lequel la variable de C a été remplacée par u. Les notions de *substitution*, *substitution close*, *domaine d'une substitution* sont également reprises du chapitre de [Apt90]. En particulier, le domaine d'une substitution est fini et l'application d'une substitution σ à un terme t se note $t\sigma$. Étant donnés deux termes t_1 et t_2, on suppose défini le plus général unificateur de t_1 et t_2, noté $mgu(t_1, t_2)$ s'il existe. À partir des termes, on définit les *formules*, *littéraux* et *clauses*. Les substitutions sont étendues aux formules. Une *clause de Horn* est une clause qui contient au plus un littéral positif. Une clause *purement négative* est une clause qui ne contient aucun littéral positif. Nous nous reportons également à [Apt90] pour les définitions de *modèle*, *modèle de Herbrand* et *plus petit modèle de Herbrand*. Avec ce vocabulaire, tout ensemble de formules universelles qui admet un modèle, admet aussi un modèle de Herbrand (théorème 3.5, page 515 [Apt90]). D'autre part, tout ensemble de clauses de Horn qui admet un modèle, admet un plus petit modèle de Herbrand (théorème 3.13, page 518 [Apt90]). Ce plus petit modèle de Herbrand peut être obtenu comme le plus petit point fixe de l'opérateur de « conséquence immédiate ».

Enfin, nous reprenons également les définitions de l'*interprétation* d'un langage du premier ordre et de la *relation de satisfaction* $\models$. Étant données une interprétation $\mathcal{I}$, une substitution σ et une formule ϕ, on notera $\mathcal{I}, \sigma \models \phi$ ce qui est noté $\mathcal{I} \models_\sigma \phi$ dans [Apt90].

3.1.2 Systèmes avec contraintes

Nous introduisons ici des systèmes avec contraintes pour alléger ensuite les clauses de Horn considérées : l'utilisation de contraintes permet de fixer *a priori* les interprétations possibles de certaines variables. De nombreux travaux sur les systèmes avec contraintes ont été effectués [Col82, Col84, KKR90, Com92]. Nous ne nous intéressons ici qu'aux propriétés élémentaires de ces systèmes avec contraintes. Les principales définitions sont tirées de [Com92].

Définition 3.1 (système avec contraintes [Com92]) *Un* système avec contraintes *se compose :*
- *d'un ensemble de formules Φ d'un langage du premier ordre $\mathcal{L}$;*
- *d'une interprétation $\mathcal{I}$ pour les formules de Φ ;*
- *d'un algorithme de décision pour le problème de satisfaction :*

$$\mathcal{I} \models \exists \vec{x} : \phi$$

où $\phi \in \Phi$ a pour variables libres $\vec{x}$.

On appelle aussi contraintes *les formules de Φ.*

Nous nous plaçons ici dans un cadre simplifié pour les systèmes de contraintes : ils sont définis par des formules de la forme $P_1(x_1) \wedge \cdots \wedge P_k(x_k)$ où les P_i sont des prédicats de $\mathcal{L}$ et les x_i des variables. L'interprétation de $\mathcal{L}$ est définie par un automate d'arbre.

Notation : En s'inspirant du vocabulaire de [KKR90, Wei98] par exemple, on dira que les x_i sont de *sorte* P_i au lieu d'écrire formellement la contrainte $P_1(x_1) \wedge \cdots \wedge P_k(x_k)$. De la même manière, on dira que le terme clos t est de *sorte* P si t appartient à l'interprétation de P dans $\mathcal{I}$.

Définition 3.2 (automate d'arbre[CDG+97]) *Un* automate d'arbre fini *sur un alphabet* $\mathcal{F}$ *est un quadruplet* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ *où* Q *est un ensemble d'états,* $Q_f \subseteq Q$ *un ensemble d'états finals, et* Δ *un ensemble de règles de transition de la forme :*

$$f(q_1, \ldots, q_n) \to q.$$

Notation : On peut écrire $f : q_1, \ldots, q_n \to q$ au lieu de $f(q_1, \ldots, q_n) \to q$.

Remarque : Nous considérons ici des automates d'arbre ascendants.

Étant donné un automate d'arbre $\mathcal{A}$, la *relation de transition* $\underset{\mathcal{A}}{\to}$ est définie par :

$$\forall t, t' \in T(\mathcal{F} \cup Q) \qquad t \underset{\mathcal{A}}{\to} t' \Leftrightarrow \begin{cases} \exists C \in \mathcal{C}(\mathcal{F} \cup Q), \\ \exists f(q_1, \ldots, q_n) \to q \in \Delta \\ t = C[f(q_1, \ldots, q_n)], t' = C[q]. \end{cases}$$

La clôture transitive de la relation $\underset{\mathcal{A}}{\to}$ est notée $\overset{*}{\underset{\mathcal{A}}{\to}}$. On dit alors qu'un terme clos t de $T(F)$ est *accepté* par un automate d'arbre $\mathcal{A}$ s'il existe un état final q de Q_f tel que

$$t \overset{*}{\underset{\mathcal{A}}{\to}} q.$$

Remarque : Les automates d'arbre avec ϵ-transition reconnaissent les mêmes langages que les automates d'arbre sans ϵ-transition [CDG+97]. On pourra donc utiliser, par la suite, des automates d'arbre avec ϵ-transition.

Définition 3.3 *Soit* $\mathcal{L}$ *un langage du premier ordre. Soit* $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ *un automate d'arbre tel que l'alphabet* $\mathcal{F}$ *contienne l'alphabet de* $\mathcal{L}$ *et tel que l'ensemble* $\mathcal{P}$ *des prédicats de* $\mathcal{L}$ *soit inclus dans* Q*. L'interprétation* $\mathcal{I}$ associée à l'automate d'arbre $\mathcal{A}$ *est définie de la manière suivante :* $\mathcal{I}$ *a pour domaine* $T(F)$ *et :*
- *les termes de* $T(\mathcal{F})$ *sont interprétés par eux-mêmes ;*
- *chaque prédicat* $P \in \mathcal{P}$ *est interprété par*

$$\{t \in T(\mathcal{F}) \mid t \overset{*}{\underset{\mathcal{A}}{\to}} P\}.$$

Étant données une formule ϕ et une contrainte $K \in \Phi$, on appelle *formule contrainte* la paire $\phi \mid K$. Dans la suite de ce chapitre, on considère en particulier des clauses contraintes. Il reste à définir la notion de modèle pour une formule contrainte dans notre cadre.

Définition 3.4 *Soit* $\phi \mid K$ *une formule contrainte telle que l'interprétation* $\mathcal{I}$ *du système de contraintes est donnée par un automate d'arbre. On définit*

$$[\![\phi \mid K]\!] \overset{def}{=} \{\phi\sigma \mid \mathcal{I}, \sigma \models K\}.$$

On étend $[\![\_]\!]$ *aux ensembles de formules contraintes par* $[\![E]\!] \overset{def}{=} \{[\![\phi \mid K]\!] \mid \ \phi \mid K \in E\}$*. Un* modèle *de* E *est alors un modèle de l'ensemble de formules* $[\![E]\!]$*.*

En particulier, si on considère un ensemble E de formules contraintes tel que chaque formule est une clause de Horn, alors $[\![E]\!]$ est un ensemble de clauses de Horn. Par conséquent, s'il existe un modèle de E, il existe un plus petit modèle de Herbrand.

Notation : On note ϕ la formule contrainte par la clause vide : $\phi \mid \top$.

3.2 Messages et traces

Les protocoles cryptographiques sont définis par des ensembles de clauses contraintes. Comme annoncé dans la partie précédente, ces contraintes, de la forme $P_1(x_1) \wedge \cdots \wedge P_k(x_k)$, fixent les interprétations possibles des variables $x_1, \ldots, x_n$. Les prédicats utilisés dans ces contraintes sont Num, Agent, Ha, Da, Liste, Message, Event, Trace et définissent, entre autre, les messages et les traces. L'interprétation de ces prédicats est définie par l'automate d'arbre $\mathcal{A}$ décrit à la figure 3.1. En particulier, on note $\mathcal{F}$ l'ensemble des symboles de fonction utilisés dans la définition de $\mathcal{A}$.

Remarque : Pour certains symboles de fonction, on utilise la notation infixée, le « _ » désignant la place des arguments. Par exemple, le symbole $<$ _ , _ $>$ a deux arguments.

Nous allons commenter en trois parties la définition de $\mathcal{A}$ et la signification intuitive des symboles de fonctions utilisés : tout d'abord les composantes élémentaires des messages (comme les agents ou les nonces), puis les messages eux-mêmes et enfin les traces.

Notation : si f est un symbole de fonction d'arité 1 et t un terme, on pourra écrire $f^n(t)$ au lieu de $\underbrace{f(f(\cdots f}_{n \text{ fois}}(t)\cdots))$.

Relations entre prédicats

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{Ha} & \rightarrow & \mathsf{Agent} & \mathsf{Da} & \rightarrow & \mathsf{Agent} \\ \mathsf{Agent} & \rightarrow & \mathsf{Message} & \mathsf{Message} & \rightarrow & \mathsf{Event} \end{array}$$

Composantes élémentaires

$$\begin{array}{rcl@{\qquad}rrcl} 0: & \rightarrow & \mathsf{Num} & s: & \mathsf{Num} & \rightarrow & \mathsf{Num} \\ h: & \rightarrow & \mathsf{Ha} & s_h: & \mathsf{Ha} & \rightarrow & \mathsf{Ha} \\ d: & \rightarrow & \mathsf{Da} & s_d: & \mathsf{Da} & \rightarrow & \mathsf{Da} \\ []: & \rightarrow & \mathsf{Liste} & \_ :: \_ : & \mathsf{Agent}, \mathsf{Liste} & \rightarrow & \mathsf{Liste} \\ srv_i & \rightarrow & \mathsf{Agent} & n: & \mathsf{Num}, \mathsf{Liste}, \mathsf{Num} & \rightarrow & \mathsf{Message} \end{array}$$

Messages

$$\begin{array}{rrcl@{\qquad}rrcl} <\_,\_>: & \mathsf{Message}, \mathsf{Message} & \rightarrow & \mathsf{Message} & \{\_\}_{\_}: & \mathsf{Message}, \mathsf{Message} & \rightarrow & \mathsf{Message} \\ \mathsf{pub}: & \mathsf{Message} & \rightarrow & \mathsf{Message} & \mathsf{prv}: & \mathsf{Message} & \rightarrow & \mathsf{Message} \\ \mathsf{shr}: & \mathsf{Message} & \rightarrow & \mathsf{Message} & \mathsf{h}: & \mathsf{Message} & \rightarrow & \mathsf{Message} \end{array}$$

Traces

$$\begin{array}{rrcl@{\qquad}rrcl} st: & \mathsf{Agent}, \mathsf{Num}, \mathsf{Num}, \mathsf{Message} & \rightarrow & \mathsf{Event} & & & & \\ \perp: & & \rightarrow & \mathsf{Trace} & [\_,\_]\cdot\_: & \mathsf{Event}, \mathsf{Num}, \mathsf{Trace} & \rightarrow & \mathsf{Trace} \end{array}$$

FIGURE 3.1 - *Les règles de transitions de l'automate $\mathcal{A}$.*

3.2.1 Composantes élémentaires

Les termes de sorte Num sont des termes de la forme $s^n(0)$ pour n entier. Le prédicat Num est uniquement utilisé pour la représentation interne des nonces et des numéros de sessions qui étiquettent chacun des messages joués par les participants. Il est important de définir explicitement une représentation pour Num car on considére par la suite un modèle de Herbrand. Cependant, la suite de la description des protocoles n'utilisera jamais explicitement la représentation de Num. Par abus de notation, on pourra parfois écrire l'entier représenté par un terme de sorte Num au lieu du terme lui-même. Ainsi, on pourra écrire 2 au lieu de $s(s(0))$.

Les prédicats Ha et Da servent à représenter respectivement les agents honnêtes et malhonnêtes. Bien sûr, les règles d'un protocole « réaliste » ne doivent pas faire la différence entre les deux types d'agents. Mais cette distinction est nécessaire pour l'expression des propriétés de sécurité. Ainsi, on demandera qu'une donnée *partagée par des agents honnêtes* reste secrète de l'intrus.

Les constantes srv_i représentent des noms d'agents particuliers : les noms des serveurs. Ainsi, l'ensemble des termes de sorte Agent est la réunion disjointe des termes de sorte Ha, de sorte Da et des constantes srv_i, ce qui se traduit dans l'automate d'arbre par les ϵ-transitions Ha $\to$ Agent et Da $\to$ Agent.

Le prédicat Liste sert tout simplement à définir une liste d'agents. Il est utilisé dans la description des nonces comme expliqué ci-après.

Notation : $[a_1; \ldots; a_k]$ désigne le terme de sorte liste : $a_1 :: (a_2 :: \cdots :: (a_k :: [\,]) \cdots)$.

Le symbole n est utilisé pour construire les nonces. Il prend pour argument un terme i, une liste l et un autre terme s. Intuitivement, i correspond au numéro de la règle du protocole au cours de laquelle le nonce a été engendré. La liste d'agents l est formée de l'agent qui a engendré le nonce ainsi que des agents qui sont supposés recevoir le nonce. Enfin, le numéro de session s permet d'assurer que deux nonces engendrés à deux sessions différentes sont distincts. En effet, on modélise les nonces par des termes que l'intrus ne connaît pas au départ et tels que deux termes correspondants à deux sessions différentes soient distincts. De plus, l'intrus ne peut jamais *construire* des termes avec le symbole de fonction n et il ne peut donc pas deviner un nonce en fonction des nonces précédents. Nous verrons cela de manière plus explicite lors de la description des règles de l'intrus au paragraphe 3.4.1.1. Cette modélisation correspond à la modélisation classique des nonces lorsqu'on se place dans le cadre de la cryptographie parfaite, comme nous l'avons présenté au chapitre 2.

Notation : On peut parfois mettre le premier argument de n en indice. Ainsi, $n_i(l, s)$ désigne le terme $n(i, l, s)$.

3.2.2 Messages

Les messages sont des termes construits sur l'alphabet

$$\mathcal{F}_{\mathsf{msg}} = \{<\_,\_>, \{\_\}_\_, \mathsf{pub}, \mathsf{prv}, \mathsf{shr}, \mathsf{h}\}$$

et à partir des termes de sortes Agent et des « nonces », c'est-à-dire des termes de la forme $n(i, l, s)$ pour $i, s \in$ Num et $l \in$ Liste.

Le symbole $<\_,\_>$ est le symbole classique de la paire.

Le symbole $\{\_\}_{\_}$ représente le chiffrement : le terme $\{m_1\}_{m_2}$ représente le message m_1 chiffré avec la clef m_2. On peut noter ici que tout message peut servir de clef ce qui n'est pas possible dans tous les modèles.

Les symboles pub, prv et shr, appliqués à un terme de sorte Agent représentent respectivement les clefs publiques, privées et partagées desdits agents. Les clefs publiques et privées sont inverses l'une de l'autre alors que la clef partagée avec le serveur est symétrique (on chiffre et on déchiffre avec la même clef). On suppose en fait que lorsque le symbole $\{\_\}_{\_}$ est appliqué à une clef publique (ou privée), on utilise un algorithme de chiffrement asymétrique, alors que dans tous les autres cas, on utilise un algorithme de chiffrement symétrique.

Définition 3.5 *L'*inverse *d'un terme m est :*
- pub(m') *si m est de la forme* prv(m'),
- prv(m') *si m est de la forme* pub(m'),
- *m dans tous les autres cas.*

Remarque : Chaque agent ne dispose ici que d'une seule clef publique mais quitte à ajouter des symboles de fonctions, nous pourrions leur associer un nombre de clefs publiques arbitraires. Nous avons choisi ici de ne pas alourdir les notations. D'autre part, comme indiqué dans le chapitre 2, nous avons choisi de n'utiliser qu'un seul symbole de chiffrement, représentant un chiffrement symétrique ou asymétrique suivant le type de clef utilisé. Nous aurions pu choisir de considérer un symbole pour chaque algorithme, auquel cas, tous les messages pourraient être utilisés comme clef symétrique ou asymétrique suivant le symbole de fonction utilisé.

Enfin, le hachage est représenté par l'application du symbole h. D'autres symboles de fonction peuvent être ajoutés.

Par abus de notation, on pourra écrire $\{x, y\}_z$ au lieu de $\{< x, y >\}_z$, ou $< x, y, z >$ au lieu de $< x, < y, z >>$ (on supposera, par défaut, que le premier argument de la paire est toujours celui le plus à gauche).

Remarque : Cette manière de construire les messages n'est pas innocente : elle reflète l'hypothèse du chiffrement parfait expliquée au chapitre 2. En particulier, si m, m', k, k' sont des termes distincts alors les termes $\{m\}_k$ et $\{m'\}_{k'}$ sont distincts alors que dans la réalité, deux messages chiffrés pourraient être identiques.

3.2.3 Traces

Une trace représente un historique des sessions déjà jouées ainsi que celles en cours. Un élément d'une trace (c'est-à-dire un terme de sorte Event) est soit un terme de sorte Message qui représente un message déjà envoyé sur le réseau, soit un terme de la forme $st(a, i, j, m)$. Intuitivement, le terme $st(a, i, j, m)$ représente le fait que l'agent a a dans sa mémoire le message m et qu'il est à l'étape j du rôle numéro i . Un agent peut éventuellement jouer plusieurs rôles dans une même session et peut confondre différentes sessions si le message mémorisé ne lui permet pas de faire la différence. La mémoire *locale* d'un agent contient par exemple certains des nonces déjà engendrés par lui et éventuellement, selon la modélisation, certains nonces qu'il a reçus. Cela correspond à la modélisation habituelle de la mémoire des agents (cf chapitre 2).

Remarque : notre modèle permet tout de même de représenter un agent avec mémoire globale : il suffit d'ajouter des états globaux $glob(a, s, t)$ où t n'est plus un message mais la trace tout entière. Cette modélisation ne sera jamais utilisée dans la suite.

3.3 Modèle

Nous pouvons maintenant définir formellement les protocoles et la notion d'attaque d'un protocole.

3.3.1 Système de contraintes cryptographiques

Soit $\mathcal{F}$, l'ensemble des symboles de fonction représentés à la figure 3.1 et soit $\mathcal{V}$ un ensemble de variables. Dans la suite de ce chapitre, nous fixons le système $\mathcal{S}$ de contraintes considéré. Le système $\mathcal{S}$ est défini par :

- un ensemble de formules Φ qui est l'ensemble des conjonctions de littéraux de la forme $P(x)$ où $P \in \{\mathsf{Num}, \mathsf{Agent}, \mathsf{Ha}, \mathsf{Da}, \mathsf{Liste}, \mathsf{Message}, \mathsf{Event}, \mathsf{Trace}\}$ et x est une variable ;
- une interprétation $\mathcal{I}$ définie par l'automate d'arbre décrit à la figure 3.1 ;
- un algorithme de décision pour le problème de satisfaction :

  $$\mathcal{I} \models \exists \vec{x} : \phi$$

  qui est tout simplement l'algorithme de décision du vide de l'intersection de langages d'automates d'arbre (EXPTIME-complet [CDG+97]).

3.3.2 Définitions

Soit $\mathcal{P}$ un ensemble de prédicats contenant un prédicat unaire T . Intuitivement, le prédicat T reconnaît l'ensemble des traces possibles. En particulier, les clauses considérées par la suite vérifient que tout terme de l'interprétation de T est un terme de sorte Trace.

Définition 3.6 *Soit $\mathcal{C}$ un ensemble de clauses contraintes. On dit que $\mathcal{C}$ est un* protocole sous forme clausale *si pour toute clause contrainte $\phi \mid K$ de $\mathcal{C}$, ϕ est une clause de Horn qui possède exactement un littéral négatif.*

On suppose, en fait, avoir seulement spécifié ce qui était possible pour le protocole $\mathcal{C}$. D'après [Apt90], il existe un plus petit modèle de Herbrand $\mathcal{H}_{\mathcal{C}}$ pour $\mathcal{C}$ car $[\![\mathcal{C}]\!]$ reste un ensemble de clauses de Horn (cf paragraphe 3.1.2).

Remarque : lorsqu'il n'y a pas d'ambiguïté, on utilisera plus simplement le terme de protocole au lieu de protocole sous forme clausale. Ce sera en particulier le cas dans la suite de ce chapitre.

Définition 3.7 *Une* trace valide *pour un protocole $\mathcal{C}$ est un membre de l'interprétation de T dans $\mathcal{H}_{\mathcal{C}}$.*

On considère une clause contrainte $\phi \mid K$ qui modélise la propriété de sécurité à vérifier.

Définition 3.8 *Un protocole $\mathcal{C}$* satisfait la propriété $\phi \mid K$ *si $\mathcal{H}_{\mathcal{C}} \models [\![\phi \mid K]\!]$ $(= \{\phi\sigma \mid \mathcal{I}, \sigma \models C\})$.*

Inversement, si $\mathcal{H}_\mathcal{C} \not\models [\![\phi \mid K]\!]$, on dit qu'il y a une attaque. Dans ce cas, par compacité [EFT96], il existe un contre-exemple fini $\mathcal{H}_0$, sous ensemble de $\mathcal{H}_\mathcal{C}$, tel que $\mathcal{H}_0 \not\models [\![\phi \mid K]\!]$.

Définition 3.9 *Une* attaque *sur $\mathcal{C}$ pour la propriété $\phi \mid K$ est un sous-ensemble fini $\mathcal{H}_0 \subset \mathcal{H}_\mathcal{C}$ tel que : $\mathcal{H}_0 \not\models [\![\phi \mid K]\!]$. De plus, $\mathcal{H}_0$ est une attaque* avec n agents *si $\mathcal{H}_0$ contient exactement n termes distincts de sorte* Agent.

Des exemples de propriétés ϕ de secret ou d'authentification seront donnés partie 3.4.2.2. Un exemple d'attaque sera donné partie 3.4.2.3.

3.3.3 Discussion

Les définitions de protocole et d'attaque sont très générales. Ainsi il est possible de construire un « protocole » $\mathcal{C}$ et une clause contrainte $\phi \mid K$ tels qu'il y ait une « attaque » sur $\mathcal{C}$ pour $\phi \mid K$ et sans pour autant que $\mathcal{C}$ ait un réel rapport avec un protocole cryptographique. Nous avons volontairement choisi un modèle très général pour que n'importe quel protocole puisse y être représenté. Ainsi, on montre au chapitre suivant que le modèle de J. Millen et H. Rueß [MR00, RM00] peut être entièrement traduit dans notre formalisme. L'objet de la prochaine partie est de montrer tout d'abord comment un protocole P peut être « naturellement » codé en un protocole sous forme de clauses $\mathcal{C}_P$ tel qu'une attaque sur le protocole P corresponde bien à une attaque sur l'ensemble de clauses $\mathcal{C}_P$.

3.4 Exemple

3.4.1 Clauses indépendantes du protocole

Les clauses dites *indépendantes du protocole* sont celles qui permettent de décrire le pouvoir de l'intrus ainsi que des prédicats auxiliaires assurant par exemple la fraîcheur des nonces. Ces clauses ne varieront pas d'un protocole à l'autre, à moins que l'on veuille changer la théorie de l'intrus.

Il faut noter que nous donnons ici un *exemple* de codage possible de la théorie de l'intrus. Nous sommes amenés à faire des choix sur les pouvoirs donnés à l'intrus. Cependant, il est tout à fait possible de faire d'autres choix et de les formaliser sous forme de clauses. C'est une souplesse laissée à notre modèle.

3.4.1.1 Intrus

Comme annoncé dans le chapitre 2, nous considérons ici un intrus avec une capacité de calcul et de mémoire illimitée, capable d'intercepter n'importe quel message envoyé sur le réseau et capable d'en envoyer de nouveaux.

La connaissance de l'intrus est décrite à l'aide d'un prédicat binaire I. Ce prédicat prend deux arguments : un message m et une trace t. Intuitivement, $I(m, t)$ signifie que l'intrus connaît le message m après exécution de la trace t. Les clauses définissant I se répartissent en trois catégories : les premières définissent la connaissance initiale, les secondes le pouvoir d'analyse et de synthèse de l'intrus, et les troisièmes la capacité de l'intrus à apprendre de nouveaux messages.

Connaissance initiale : ensemble de clauses $\mathcal{C}_{I_{\mathsf{init}}}$.

	$\Rightarrow$	$I(x,t)$	$\mid \mathsf{Agent}(x)$	L'intrus connaît l'identité de chacun des agents.	(3.1)
$I(x,t)$	$\Rightarrow$	$I(\mathsf{pub}(x),t)$	$\mid \top$	L'intrus connaît toutes les clefs publiques.	(3.2)
	$\Rightarrow$ $\Rightarrow$	$I(\mathsf{prv}(x),t)$ $I(\mathsf{shr}(x),t)$	$\mid \mathsf{Da}(x)$ $\mid \mathsf{Da}(x)$	L'intrus dispose des clefs privées des agents malhonnêtes.	(3.3)

Un agent malhonnête se caractérise donc par le fait que l'intrus connaît toutes ses clefs privées, alors que l'intrus ne connaît aucune des clefs privées des agents honnêtes. Dans ce cas, un agent malhonnête peut encore choisir de ne pas communiquer à l'intrus les nonces qu'il crée. Si l'on préfère modéliser que les agents malhonnêtes communiquent absolument toutes leurs données à l'intrus, il suffit d'ajouter la clause :

$$\Rightarrow I(n(i,x::l,s),t) \quad \mid \mathsf{Da}(x), \mathsf{Num}(i), \mathsf{Liste}(l), \mathsf{Num}(s).$$

Remarque : nous choisissons ici d'autoriser l'application du symbole de fonction pub à n'importe quel message et non seulement à des agents. Cela permet à l'intrus de construire plus de clefs composées. Mais on peut aussi choisir de restreindre l'application de ce symboles à des termes de sorte Agent : il suffit d'ajouter la contrainte $\mathsf{Agent}(x)$ à l'équation 3.2.

Analyse et synthèse : ensemble de clauses $\mathcal{C}_{I_1}$.
Le prédicat Sym sera défini au paragraphe suivant. Intuitivement $\mathsf{Sym}(m)$ signifie que m est un message « symétrique », c'est-à-dire que si m est utilisé comme clef alors la clef inverse est m lui-même.

$I(x,t), I(y,t)$	$\Rightarrow$	$I(<x,y>,t)$	L'intrus peut former la paire de deux messages connus.	(3.4)
$I(x,t), I(y,t)$	$\Rightarrow$	$I(\{x\}_y,t)$	L'intrus peut chiffrer un message connu avec une clef connue.	(3.5)
$I(x,t)$	$\Rightarrow$	$I(\mathsf{h}(x),t)$	L'intrus peut hacher un message.	(3.6)
$I(<x,y>,t)$ $I(<x,y>,t)$	$\Rightarrow$ $\Rightarrow$	$I(x,t)$ $I(y,t)$	L'intrus peut projeter sur la première ou la deuxième composante.	(3.7)
$I(\{x\}_y,t), \mathsf{Sym}(y), I(y)$ $I(\{x\}_{\mathsf{pub}(y)},t), I(\mathsf{prv}(y),t)$ $I(\{x\}_{\mathsf{prv}(y)},t), I(\mathsf{pub}(y),t)$	$\Rightarrow$ $\Rightarrow$ $\Rightarrow$	$I(x,t)$ $I(x,t)$ $I(x,t)$	L'intrus peut déchiffrer un message s'il possède l'inverse de la clef.	(3.8)

Remarque : ici encore, on utilise l'hypothèse du chiffrement parfait : l'intrus peut déchiffrer le message m chiffré avec la clef k uniquement dans le cas où il possède l'inverse de la clef k. En particulier, il ne peut pas utiliser la force brute ou deviner le message.

Interception et mémorisation : ensemble de clauses $\mathcal{C}_{I_2}$.

$$T([x,s]\cdot t) \quad \Rightarrow \quad I(x,[x,s]\cdot t) \mid \mathsf{Num}(s) \qquad \text{Tous les messages envoyés sur le réseau peuvent être interceptés.} \tag{3.9}$$

$$I(x,t) \quad \Rightarrow \quad I(x,y\cdot t) \qquad \mid \mathsf{Event}(y) \qquad \text{Les messages sont conservés aussi longtemps que l'intrus le souhaite.} \tag{3.10}$$

On note $\mathcal{C}_I$ l'union des clauses $\mathcal{C}_{I_{\mathsf{init}}}$, $\mathcal{C}_{I_1}$ et $\mathcal{C}_{I_2}$. Cet ensemble représente les capacités de l'intrus.

3.4.1.2 Prédicats auxiliaires

Comme annoncé au paragraphe précédent, le prédicat Sym accepte uniquement les messages symétriques, i.e. les messages égaux à leur inverse.

$$\Rightarrow \quad \mathsf{Sym}(x) \quad \mid \mathsf{Agent}(x) \tag{3.11}$$
$$\Rightarrow \quad \mathsf{Sym}(n(i,l,s)) \quad \mid \mathsf{Num}(i), \mathsf{Num}(s), \mathsf{Liste}(l) \tag{3.12}$$
$$\Rightarrow \quad \mathsf{Sym}(f(m_1,\cdots,m_k)) \quad \mid \mathsf{Message}(m_1),\ldots,\mathsf{Message}(m_k) \tag{3.13}$$
$$\text{pour } f \in \{<\_,\_>, \{\_\}_{\_}, h, \mathsf{shr}\}$$

Sup simule la relation « supérieur strict » pour les termes de sorte Num. Sup vérifie que $\mathsf{Sup} \subseteq \mathsf{Num} \times \mathsf{Num}$ et $\mathsf{Sup}(s^n(0), s^m(0))$ est vrai si et seulement si $n > m$.

$$\Rightarrow \quad \mathsf{Sup}(s(x),0) \quad \mid \mathsf{Num}(x) \tag{3.14}$$
$$\mathsf{Sup}(x,y) \quad \Rightarrow \quad \mathsf{Sup}(s(x),s(y)) \tag{3.15}$$

$\mathsf{Fresh}(t,s)$ est vrai si t est une trace et si s est un numéro de session plus grand que tous ceux qui étiquettent les évènements de t.

$$\Rightarrow \quad \mathsf{Fresh}(\perp,s) \quad \mid \mathsf{Num}(s) \tag{3.16}$$
$$\mathsf{Fresh}(t,s), \mathsf{Sup}(s,s') \quad \Rightarrow \quad \mathsf{Fresh}([e,s']\cdot t, s) \quad \mid \mathsf{Event}(e) \tag{3.17}$$

In simule la relation d'appartenance pour les événements étiquetés d'une trace t.

$$\Rightarrow \quad \mathsf{In}([e,s],[e,s]\cdot t) \quad \mid \mathsf{Event}(e), \mathsf{Num}(s), \mathsf{Trace}(t) \tag{3.18}$$
$$\mathsf{In}(x,t) \quad \Rightarrow \quad \mathsf{In}(x,[e,s]\cdot t) \quad \mid \mathsf{Event}(e), \mathsf{Num}(s) \tag{3.19}$$

$\mathsf{NotPlayed}(a,i,k,s,t)$ exprime le fait que l'agent a n'a encore jamais été dans l'état i pour le rôle k, la session s et la trace t. Ce prédicat est utilisé pour empêcher le rejeu d'une règle dans une même session. Remarquons que même si ce prédicat définit une propriété « négative » :

l'agent *n'a pas* encore joué la règle i dans la session s pour le rôle k, ce prédicat est défini positivement : toutes les clauses ont un littéral positif.

$$\Rightarrow \mathsf{NotPlayed}(a,i,k,s,\bot) \quad | \ \mathsf{Agent}(a), \mathsf{Num}(i), \mathsf{Num}(k), \mathsf{Num}(s) \tag{3.20}$$

$$\mathsf{NotPlayed}(a,i,k,s,t), \mathsf{Sup}(s,s') \Rightarrow \mathsf{NotPlayed}(a,i,k,s,[e,s']\cdot t) \quad | \ \mathsf{Event}(e) \tag{3.21}$$

$$\mathsf{NotPlayed}(a,i,k,s,t), \mathsf{Sup}(s',s) \Rightarrow \mathsf{NotPlayed}(a,i,k,s,[e,s']\cdot t) \quad | \ \mathsf{Event}(e) \tag{3.22}$$

$$\mathsf{NotPlayed}(a,i,k,s,t) \Rightarrow \mathsf{NotPlayed}(a,i,k,s,[m,s']\cdot t) \quad | \ \mathsf{Message}(m) \tag{3.23}$$

$$\mathsf{NotPlayed}(a,i,k,s,t), \mathsf{Sup}(k,k') \Rightarrow \mathsf{NotPlayed}(a,i,k,s,[st(a,k',j,m),s]\cdot t) \quad | \ \mathsf{Message}(m), \mathsf{Num}(j) \tag{3.24}$$

$$\mathsf{NotPlayed}(a,i,k,s,t), \mathsf{Sup}(k',k) \Rightarrow \mathsf{NotPlayed}(a,i,k,s,[st(a,k',j,m),s]\cdot t) \quad | \ \mathsf{Message}(m), \mathsf{Num}(j) \tag{3.25}$$

$$\mathsf{NotPlayed}(a,i,k,s,t), \mathsf{Sup}(i,j) \Rightarrow \mathsf{NotPlayed}(a,i,k,s,[st(a,k,j,m),s]\cdot t) \quad | \ \mathsf{Message}(m) \tag{3.26}$$

Enfin, il faut initialiser la définition des traces en admettant la clause vide $\bot$ comme trace valide.

$$\Rightarrow T(\bot) \tag{3.27}$$

Les autres clauses définissant le prédicat T sont des clauses dépendantes du protocole, qui font l'objet du paragraphe suivant.

L'ensemble des clauses définies dans ce paragraphe est noté $\mathcal{C}_{\mathsf{aux}}$.

3.4.2 Clauses dépendantes du protocole

Comme leur nom l'indique, les clauses *dépendantes du protocole* sont redéfinies pour chaque protocole considéré. Nous prenons ici l'exemple du protocole de Yahalom [CJ97], décrit à la figure 3.2. Nous traduisons ci-dessous ce protocole en clauses de Horn à l'aide des prédicats définis à la section précédente.

1. $A \to B : A, N_a$
2. $B \to S : B, \{A, N_a, N_b\}_{\mathsf{shr}(B)}$
3. $S \to A : \{B, K_{ab}, N_a, N_b\}_{\mathsf{shr}(A)}, \{A, K_{ab}\}_{\mathsf{shr}(B)}$
4. $A \to B : \{A, K_{ab}\}_{\mathsf{shr}(B)}, \{N_b\}_{K_{ab}}$

FIGURE 3.2 - *Le protocole de Yahalom.*

Dans tout ce qui suit, les clauses considérées sont contraintes par la formule :

$$K_0 \stackrel{\text{def}}{=} \mathsf{Agent}(a) \wedge \mathsf{Agent}(b) \wedge \mathsf{Message}(x) \wedge \mathsf{Message}(y) \wedge \mathsf{Message}(z) \wedge \mathsf{Num}(s) \wedge \mathsf{Trace}(t).$$

3.4.2.1 Les clauses décrivant le protocole de Yahalom

La première clause (contrainte) de Horn ne correspond pas directement à une règle du protocole. C'est une clause qui sert à initialiser une session : à tout moment, les participants peuvent commencer une nouvelle session.

Initialisation

$$\mathsf{Fresh}(t,s), T(t) \Rightarrow T(\ \begin{array}{l}[st(a,1,1,<a,b,srv>),s] \\ \cdot[st(b,2,1,<b,srv>),s] \\ \cdot[st(srv,3,1,srv),s]\cdot t)\end{array}$$

Si t est une trace valide (i.e. un historique possible pour ce protocole) et si s est un numéro de session non utilisé dans t alors trois états sont ajoutés à la trace : a, b et srv sont prêts à démarrer la session s en jouant respectivement les rôles 1, 2 et 3.

Message 1 L'agent A joue le rôle 1 et envoie son premier message composé de son identité et d'un nonce, puis A passe dans l'état 2 où il mémorise son nonce et le fait qu'il a déjà joué le premier message de la session.

$$\left.\begin{array}{r}\mathsf{NotPlayed}(a,1,2,s,t), T(t) \\ \mathsf{In}([st(a,1,1,<a,b,srv>),s],t)\end{array}\right\} \Rightarrow \begin{array}{ll}T(& [<a,n_1([a;b],s)>,s] \\ & \cdot[st(a,1,2,<a,b,srv,n_1([a;b],s)>),s] \\ & \cdot t)\end{array}$$

Intuitivement, cette clause peut se lire de la manière suivante : si t est une trace valide, si a n'est pas déjà dans l'état 2 pour cette session et pour le rôle 1 (i.e. a n'a pas déjà joué la première règle du premier rôle), alors le premier message est envoyé et a se retrouve dans l'état 2.

Message 2 Si B reçoit un message de la forme $<a,x>$, alors la requête correspondante est envoyée au serveur :

$$\left.\begin{array}{r}\mathsf{NotPlayed}(b,2,2,s,t), \\ T(t), I(<a,x>,t), \\ \mathsf{In}([st(b,2,1,<b,srv>),s],t)\end{array}\right\} \Rightarrow \begin{array}{ll}T(& [<b,\{a,x,n_2([a;b],s)\}_{\mathsf{shr}(b)}>,s] \\ & \cdot[st(b,2,2,<b,srv,a,n_2([a;b],s)>),s] \\ & \cdot t)\end{array}$$

Message 3 Le serveur engendre une clef de session $n_3([a;b],s)$ pour A et B.

$$\left.\begin{array}{r}I(<b,\{a,x,y\}_{\mathsf{shr}(b)}>,t), \\ \mathsf{NotPlayed}(srv,3,2,s,t), T(t), \\ \mathsf{In}([st(srv,3,1,srv),s],t)\end{array}\right\} \Rightarrow \begin{array}{ll}T(& [m,s] \\ & \cdot[st(srv,3,2,srv),s]\cdot t)\end{array}$$

où $m=<\{b,n_3([a;b],s),x,y\}_{\mathsf{shr}(a)},\{a,n_3([a;b],s)\}_{\mathsf{shr}(b)}>$.

Message 4 L'agent A renvoie la clef à B et un message d'authentification, chiffré avec cette clef.

$$\left.\begin{array}{r}T(t), \mathsf{In}([u_1,s],t), \\ \mathsf{NotPlayed}(a,1,3,s,t), \\ I(<\{b,x,n_1([a;b],s),y\}_{\mathsf{shr}(a)},z>,t)\end{array}\right\} \Rightarrow T([<z,\{y\}_x>,s]\cdot[u_2,s]\cdot t)$$

où $u_1 = st(a, 1, 2, <a, b, srv, n_1([a; b], s)>)$ et $u_2 = st(a, 1, 3, <a, b, srv, n_1([a; b], s), x, y>)$.

Nous verrons au chapitre 4 que cet ensemble de clauses correspond à la traduction du protocole de Yahalom, formalisé dans le modèle d'Harald Rueß et Jonathan Millen. Ceci donne un argument supplémentaire pour affirmer que l'exemple de formalisation donné ici modélise de manière satisfaisante les protocoles cryptographiques.

3.4.2.2 Les propriétés de sécurité

Les clauses de Horn permettent d'exprimer des propriétés de sécurité très variées. Ainsi, le protocole de Yahalom a pour double but d'assurer le secret de la clef de session K_{ab} et l'authentification mutuelle des deux agents A et B. De telles propriétés peuvent être traduites dans notre modèle.

Secret à long terme. On peut exprimer qu'une donnée (un nonce ou une clef long terme par exemple) reste secrète entre deux agents honnêtes à tout moment du protocole. Ainsi, la formule

$$(\forall t, x, y, s).\neg T(t) \vee \neg I(n_2([x; y], s), t) \ \mid \ \mathsf{Ha}(x) \wedge \mathsf{Ha}(y)$$

signifie que quelles que soient la trace valide t et la session s considérées, si a et b sont des agents honnêtes, le terme $n_2([x; y], s)$ n'est pas connu de l'intrus. Le terme $n_2([x; y], s)$ correspond ici à la clef de session K_{ab} engendrée par le serveur à la troisième étape du protocole.

Pour exprimer le secret long terme d'une donnée « long terme » (i.e. une constante du protocole), il suffit de remplacer dans la formule le terme $n_2([x; y], s)$ par la donnée considérée, $\mathsf{prv}(a)$ par exemple.

Authentification. Comme nous l'avons expliqué dans le chapitre d'introduction, il existe de très nombreuses propriétés d'authentification. Ici, nous n'en présentons qu'une mais la plupart de ces propriétés peuvent, de la même manière, être traduite en clauses de Horn. Ainsi, pour exprimer que le message d'authentification $\{N_b\}_{K_{ab}}$ reçu par B dans le protocole de Yahalom a effectivement été construit par A, on peut écrire la formule suivante :

$$\begin{aligned}\neg T(t) \ \vee \ \neg I(< \{n_2([x; y], s)\}_{n_3([x;y],s)} >, t) \\ \vee \ \mathsf{In}([st(x, 1, 3, < x, y, srv, z, n_3([x; y], s), n_2([x; y], s) >), s], t) \\ \mid \ \mathsf{Ha}(x) \wedge \mathsf{Ha}(y) \wedge \mathsf{Message}(z).\end{aligned}$$

On vérifie que le message $\{n_2([x; y], s)\}_{n_3([x;y],s)}$ (qui correspond à $\{N_b\}_{K_{ab}}$) envoyé à B est bien construit à partir du nonce $n_2([x; y], s)$ (c-à-d N_b) et de la clef $n_3([x; y], s)$ (c-à-d K_{ab}) mémorisés par A à la dernière étape.

Secret court terme. Des propriétés plus fines de secret peuvent aussi être exprimées. Ainsi, on peut souhaiter qu'une donnée générée à la session s ne soit pas connue tant que la dernière règle de la session correspondante n'est pas jouée. Une telle propriété est pertinente pour des protocoles qui révèlent un nonce à la fin des sessions. Un tel cas de figure se présente lorsqu'un agent prouve qu'il a bien reçu un message en émettant la partie décryptée. Une telle propriété peut aussi s'appliquer à des protocoles modifiés où les agents terminent le protocole en envoyant les nonces qu'ils ont générés. On peut ainsi simuler le cas où les agents « perdent » les nonces qu'ils ont engendrés. Le protocole de Yahalom modifié de cette manière est décrit à la figure 3.3.

1. $A \to B : A, N_a$
2. $B \to S : B, \{A, N_a, N_b\}_{\mathsf{shr}(B)}$
3. $S \to A : \{B, K_{ab}, N_a, N_b\}_{\mathsf{shr}(A)}, \{A, K_{ab}\}_{\mathsf{shr}(B)}$
4. $A \to B : \{A, K_{ab}\}_{\mathsf{shr}(B)}, \{N_b\}_{K_{ab}}$
5. $B \to A : N_b$

FIGURE 3.3 - *Protocole de Yahalom modifié.*

Pour un tel protocole, on veut alors pouvoir prouver le secret de N_b « tant que B ne l'a pas révélé lui-même » et ce, même si les nonces N'_b des autres sessions sont révélés à la fin de chacune des sessions correspondantes. Cela s'exprime de la manière suivante :

$$(\forall t, x, y, s).\neg T(t) \vee \neg \mathsf{Ha}(x) \vee \neg \mathsf{Ha}(y) \vee \neg I(n_2([x; y], s), t) \vee \neg \mathsf{NotPlayed}(y, 2, 3, s, t).$$

Le nonce $n_2([x; y], s)$ (correspondant à N_b) n'est pas connu de l'intrus tant que l'agent y n'est pas dans l'état 3 pour la session s.

3.4.2.3 Attaque

Il n'y a pas d'attaque connue sur le protocole de Yahalom. Par contre, M. Burrows, M. Abadi et R. Needham en ont proposé une version simplifiée, appelée BAN-Yahalom [BAN89], dont on a ensuite trouvé de nombreuses attaques [Syv94]. Nous présentons figure 3.4 une version inspirée du protocole de BAN-Yahalom dont nous allons montrer qu'elle admet le même type d'attaques (basé sur le rejeu de messages) que ce protocole. Les clauses contraintes pour ce protocole modifié sont les cinq clauses décrites au paragraphe 3.4.2.1, modifiées en tenant compte du léger changement de la deuxième règle, auxquelles on ajoute la règle suivante, correspondant à la dernière règle du protocole :

$$\left.\begin{array}{r} \mathsf{NotPlayed}(b, 2, 3, s, t), \\ T(t), I(<\{a, y\}_{\mathsf{shr}(b)}, \{n_2([a; b], s)\}_y >, t), \\ \mathsf{In}([st(b, 2, 2, < b, srv, a, n_2([a; b], s) >), s], t) \end{array}\right\} \Rightarrow \quad \begin{array}{ll} T(& [\{n_3([a; b], s)\}_y, s] \\ & \cdot u \cdot t) \end{array}$$

où $u \stackrel{\text{def}}{=} [st(b, 2, 3, < b, srv, a, n_2(a, b, s) >, y, n_3(a, b, s)), s]$.

Supposons que l'on veuille vérifier le secret du nonce N'_b, la propriété correspondante est :

$$\phi = \neg T(t) \vee \neg I(n_3([x; y], s), t) \ \mid \ \mathsf{Ha}(x) \wedge \mathsf{Ha}(y).$$

Une des attaques est construite de la manière suivante (décrite figure 3.5) : les deux premières règles se jouent normalement puis l'intrus, se faisant passer pour A, renvoie à B une partie de son propre message $\{A, N_a\}_{\mathsf{shr}(B)}$ et un message d'authentification $\{N_b\}_{N_a}$. L'agent B croit

1. $A \to B : A, N_a$
2. $B \to S : B, N_b, \{A, N_a\}_{\mathsf{shr}(B)}$
3. $S \to A : \{B, K_{ab}, N_a, N_b\}_{\mathsf{shr}(A)}, \{A, K_{ab}\}_{\mathsf{shr}(B)}$
4. $A \to B : \{A, K_{ab}\}_{\mathsf{shr}(B)}, \{N_b\}_{K_{ab}}$
5. $B \to A : \{N_b'\}_{K_{ab}}$

FIGURE 3.4 - *BAN-Yahalom modifié.*

1. $A \to B : A, N_a$
2. $B \to S : B, N_b, \{A, N_a\}_{\mathsf{shr}(B)}$
3. règle omise
4. $I(A) \to B : \{A, N_a\}_{\mathsf{shr}(B)}, \{N_b\}_{N_a}$
5. $B \to A : \{N_b'\}_{N_a}$

FIGURE 3.5 - *Attaque du BAN-Yahalom modifié.*

alors que la clef partagée (K_{ab}) a pour valeur la valeur de N_a, il renvoie donc son nonce secret N_b' chiffré avec N_a. Comme l'intrus connaît N_a, il peut aisément déchiffrer le message.

L'attaque formelle (définie au paragraphe 3.3.2) est la suivante : on considère le plus petit modèle de Herbrand $\mathcal{H}_P$ de $\mathcal{C}_P$ et on montre qu'il existe un sous-ensemble $\mathcal{H}_0 \subset \mathcal{H}_P$ tel que $\mathcal{H}_0 \not\models [\![\phi]\!]$. Soient :

$$\begin{aligned}
t_1 &\stackrel{\text{def}}{=} [st(h,1{,}1,<h,s_h(h),srv>),0]\cdot[st(s_h(h),2{,}1,<s_h(h),srv>),0]\cdot[st(srv,3,1,srv),0]\cdot\bot,\\
t_2 &\stackrel{\text{def}}{=} [<h,n_1([h;s_h(h)],0)>,0]\cdot[st(h,1,2,<h,s_h(h),srv,n_1([h;s_h(h)],0)>),0]\cdot t_0,\\
t_3 &\stackrel{\text{def}}{=} [<s_h(h),n_2([h;s_h(h)],0),\{h,n_1([h;s_h(h)],0)\}_{\mathsf{shr}(s_h(h))}>,s]\\
&\quad\cdot[st(s_h(h),2,2,<s_h(h),srv,h,n_2([h;s_h(h)],0)>),0]\cdot t_1\\
t_4 &\stackrel{\text{def}}{=} [\{n_3([h;s_h(h)],0)\}_{n_1([h;s_h(h)],0)},0]\\
&\cdot[st(s_h(h),2{,}3,<s_h(h),srv,h,n_2([h;s_h(h)],0)>,n_1([h;s_h(h)],0),n_3([h;s_h(h)],0)),0]\cdot t_2.
\end{aligned}$$

Les termes t_i correspondent respectivement à la trace obtenue après l'application de la $i^{\text{ième}}$ étape de l'attaque. On peut vérifier, en appliquant les clauses de $\mathcal{C}_P$, que : $T(t_1), T(t_2), T(t_3)$, $T(t_4) \in \mathcal{H}_P$. En particulier, on montre $T(t_4) \in \mathcal{H}_P$ en utilisant que $T(t_3) \in \mathcal{H}_P$ et :

$$I(<\{h,n_1([h;s_h(h)],0)\}_{\mathsf{shr}(s_h(h))},\{n_2([h;s_h(h)],0)\}_{n_1([h;s_h(h)],0)}>,t_3) \in \mathcal{H}_P.$$

De plus, $I(\{n_2([h;s_h(h)],0)\}_{n_1([h;s_h(h)],0)},t_4) \in \mathcal{H}_P$ et $I(n_1([h;s_h(h)],0),t_4) \in \mathcal{H}_P$ assurent $I(n_3([h;s_h(h)],0),t_4) \in \mathcal{H}_P$.

Soit $\mathcal{H}_0 \stackrel{\text{def}}{=} \{T(t_4), I(n_3([h; s_h(h)], 0), t_4)\}$, on a $\mathcal{H}_0 \subset \mathcal{H}_P$ et $\mathcal{H}_0 \not\models \phi\sigma_0$ pour $\sigma_0 = [h/a, s(h)/b]$. Comme $\phi\sigma_0 \in [\![\phi]\!]$, $\mathcal{H}_0$ est une attaque sur $\mathcal{C}_P$ pour la propriété ϕ.

3.5 Expressivité

Ce modèle de protocole est très général et expressif, tant au niveau des primitives cryptographiques qu'au niveau des propriétés de secret expressibles dans ce modèle. Nous montrons en particulier au chapitre 4 que le modèle de J. Millen et H. Rueß peut se traduire en un protocole sous forme de clauses de Horn.

En contrepartie, la plupart des propriétés (secret, authentification, ...) sont indécidables. Ainsi, on peut coder le problème de correspondance de Post comme l'ont fait S. Even et O. Golreich dans [EG83]. Nous présentons rapidement un de ces codages, suggéré par M. Rusinowitch.

Soient Σ un alphabet fini et $(u_i, v_i)_{1 \leqslant i \leqslant n}, u_i, v_i \in \Sigma^*$ une entrée du problème de correspondance de Post. On considère le protocole à trois règles défini ci-dessous :

$$\begin{array}{rcll} A & \rightarrow & B : \{< 0, 0 >\}_{K_{ab}} & \\ B & \rightarrow & A : \{< N_1, N_2 >\}_{K_{ab}} & \\ A : \{< x, y >\}_{K_{ab}} & \rightarrow & B : \{< xu_i, yv_i >\}_{K_{ab}}, \{s\}_{\{< xu_i, xu_i >\}_{K_{ab}}} & 1 \leqslant i \leqslant n \end{array}$$

Dans la troisième règle, A envoie à B la concaténation des messages formés pour chaque couple (u_i, v_i). La partie gauche de la troisième règle décrit le message attendu par A. Les symboles N_1, N_2 et s représentent des nonces, K_{ab} une clef long terme, secrète entre A et B. On peut alors montrer que s est secret si et seulement s'il n'y a pas de solution au problème de correspondance de Post.

Ce codage présente l'inconvénient d'utiliser des clefs composées. On peut pallier cela en modifiant le protocole de manière à ce que l'agent A teste lui-même l'égalité des éléments de la paire reçue :

$$\begin{array}{rcll} A & \rightarrow & B : \{< 0, 0 >\}_{K_{ab}} & \\ B & \rightarrow & A : \{< N_1, N_2 >\}_{K_{ab}} & \\ A : \{< x, y >\}_{K_{ab}} & \rightarrow & B : \{< xu_i, yv_i >\}_{K_{ab}} & 1 \leqslant i \leqslant n \\ A : \{x, x\} & \rightarrow & A : s & \end{array}$$

L'inconvénient est alors que la dernière règle de ce protocole peut être jouée uniquement si le protocole correspond à une entrée qui est solution du problème de correspondance de Post. Le codage obtenu ne représente donc pas un protocole très « réaliste ».

Nous présentons dans l'article [CC01] un codage qui n'utilise pas les clefs composées et tel que le protocole peut toujours être joué en entier. Le protocole obtenu code alors l'accessibilité de la configuration vide pour un réseau de Petri avec Reset. Ce codage montre l'indécidabilité du secret pour une classe restreinte de protocoles de notre modèle sous forme de clauses de Horn.

Cependant, notre modèle nous permet, au chapitre 6, de montrer un résultat de réduction très général sur le nombre d'agents nécessaire à une attaque. De plus, nous présentons au chapitre 8 des restrictions décidables de ce modèle.

CHAPITRE 4

Modèle Millen-Rueß

Le modèle développé par J. Millen et H. Rueß [MR00, RM00, CMR01] est un modèle inspiré de celui de L. Paulson [Pau98]. Il s'agit d'un modèle de trace : on considère l'historique de tous les messages envoyés sur le réseau lors du déroulement de différentes instances du protocole. L'historique contient aussi les états courants des participants pour chacune des sessions en cours. Une règle du protocole décrit l'évolution de l'historique en fonction de ce qu'il contient : si un agent est dans l'état i et que le message m est dans l'historique, on ajoute le message m' correspondant au message envoyé par l'agent lorsqu'il reçoit le message m à l'étape i.

Ce modèle présente la particularité de définir la propriété à vérifier en même temps que le protocole lui-même. Le plus souvent, on spécifie d'abord les règles du protocole, c'est-à-dire le comportement de chaque agent en fonction des messages qu'il reçoit, et seulement ensuite les propriétés à vérifier. Cette dernière approche paraît *a priori* plus naturelle mais elle présente l'inconvénient de rendre plus difficile la spécification de la propriété à vérifier. Comment exprimer, par exemple, que les clefs de sessions engendrées *par des agents honnêtes* restent secrètes alors qu'aucune contrainte n'est imposée sur les clefs engendrées par des agents malhonnêtes ?

Ainsi, dans notre modèle présenté au chapitre 3, on introduit explicitement dans le terme représentant un nonce l'identité de l'agent qui l'a engendré. De cette façon, le terme $n(a, b, s)$ représente le nonce engendré par a, à l'intention de b, au cours de la session s. Une telle construction est artificielle car un nonce est une suite de bits engendrés aléatoirement, qui ne dépend absolument pas de a ou de b.

Le modèle Millen-Rueß ne nécessite pas d'introduire une telle dépendance. En contrepartie, il ne permet d'exprimer qu'une seule propriété : le secret (long terme) de nonces ou de clefs engendrés au cours du protocole.

Dans ce chapitre, nous présentons le modèle Millen-Rueß puis nous montrons que tout protocole exprimé dans ce modèle peut se traduire en un ensemble de clauses de Horn de telle manière que le protocole de départ est secret si et seulement si sa traduction l'est.

4.1 Présentation du modèle

Nous commençons ici par définir les messages et les ensembles inductifs définis sur les ensembles de messages avant de présenter la modélisation du protocole et la notion de secret.

4.1.1 Messages

Les messages sont modélisés par des termes, de la même manière qu'au chapitre précédent. Pour modéliser le pouvoir d'analyse et de synthèse de l'intrus, on définit les ensembles parts, analz et synth. De plus, le modèle Millen-Rueß introduit la notion d'*idéal* : l'idéal d'un ensemble S est une sur-approximation de l'ensemble S' qui doit rester secret pour qu'aucun des messages de S ne soit connu de l'intrus. Tous ces ensembles sont définis inductivement.

4.1.1.1 Primitives cryptographiques

Les parties atomiques des messages se décomposent en trois ensembles dénombrables de constantes : $Agent$, Key, et $Nonce$, qui représentent respectivement les données de types agents, clefs et nonces. Ces ensembles sont supposés disjoints et sont des sous-ensembles de l'ensemble $Fields$ des messages. Les clefs et les nonces forment l'ensemble $Basic$. Les messages $m \in Basic$ sont appelés *messages basiques*. De la même façon, les messages $m \in Agent \cup Basic$ sont appelés *messages atomiques*. Dans ce modèle, l'ensemble $Basic$ contient les seules données dont on peut demander le secret. Ainsi, on ne pourra pas exprimer le secret d'un message composé ou d'un agent.

Remarque : Les propriétés expressibles dans le modèle sous forme de clauses de Horn décrit au chapitre 3 ne sont pas aussi restreintes puisqu'on peut demander le secret d'un message quelconque. On peut également exprimer des propriétés d'authentification.

Notation : par convention, les variables A, B et leurs variantes (A_i, B_j, ...) représentent des agents, c'est-à-dire, désignent des variables à valeurs dans $Agent$; K et ses variantes désignent des clefs et N et ses variantes désignent des nonces. Inversement, X, Y et Z représentent des messages quelconques. En s'inspirant du vocabulaire introduit au chapitre précédent 3.1.2, on dira qu'une substitution σ est *bien sortée* si pour toute variable A représentant des agents, $\sigma(A) \in Agent$, et si pour toute variable N représentant des nonces, $\sigma(N) \in Nonce$ et ainsi de suite.

Chaque agent possède des clefs *à long terme*, c'est-à-dire des clefs qui ne varient pas d'une session du protocole à l'autre. Ainsi, chaque agent a a une clef publique $\mathsf{pub}(a)$, une clef privée $\mathsf{prv}(a)$ et une clef symétrique $\mathsf{shr}(a)$. L'ensemble des clefs à long terme est noté $\mathcal{K}_L$. De plus, on distingue les clefs $\mathsf{prv}(a)$ et $\mathsf{shr}(a)$ qui sont supposées rester secrètes au cours du protocole :

$$\begin{aligned} ltk(a) &= \{\mathsf{prv}(a), \mathsf{shr}(a)\} \\ \mathcal{K}_{LS} &= \{ltk(a) \mid a \in Agent\} \\ \mathcal{K}_L &= \mathcal{K}_{LS} \cup \{\mathsf{pub}(a) \mid a \in Agent\}. \end{aligned}$$

Les *clefs* sont définies comme les éléments de $Key \cup \mathcal{K}_L$.

L'ensemble $Fields$ est clôt par concaténation, notée $<\_,\_>$, chiffrement, noté $\{\_\}_\_$, et hachage, noté h (on reprend la même notation infixée qu'au chapitre 3).

Définition 4.1 *L'ensemble de messages Fields est le plus petit ensemble de termes sur* $\{<\_,\_>, \{\_\}_\_, \mathsf{h}, \mathsf{pub}, \mathsf{prv}, \mathsf{shr}\} \cup Agent \cup Key \cup Nonce$ *tel que :*

1. $Agent, Key, Nonce, \mathcal{K}_L \subseteq Fields$ *;*

2. *si* $m_1, m_2 \in Fields$ *alors* $< m_1, m_2 > \in Fields$ *;*
3. *si* $m \in Fields$ *et* $k \in Key \cup \mathcal{K}_L$*, alors* $\{m\}_k \in Fields$ *;*
4. *si* $m \in Fields$ *alors* $\mathsf{h}(m) \in Fields$*.*

Notation : comme au chapitre précédent, on écrit $< x, y, z >$ au lieu de $< x, < y, z >>$ et on ne met pas les chevrons lorsqu'il n'y a pas d'ambiguïté.

Le chiffrement d'un message m par une clef k est toujours noté $\{m\}_k$ quelque soit le type de la clef (symétrique ou non). De même qu'au chapitre 3, cette modélisation du chiffrement met en oeuvre l'hypothèse du chiffrement parfait, développée au chapitre 2.

Définition 4.2 *L'*inverse *d'une clef* $k \in Key \cup \mathcal{K}_L$ *est notée* k^{-1} *et est définie de la manière suivante :*

1. $k^{-1} = k$ *si* $k \in Key$ *;*
2. $\mathsf{pub}(a)^{-1} = \mathsf{prv}(a)$ *;*
3. $\mathsf{prv}(a)^{-1} = \mathsf{pub}(a)$ *;*
4. $\mathsf{shr}(a)^{-1} = \mathsf{shr}(a)$*.*

Enfin, srv est une constante spéciale de l'ensemble $Agent$. Intuitivement, srv désigne un serveur de confiance qui connaît toutes les clefs symétriques $\mathsf{shr}(a)$ des agents.

4.1.1.2 Ensembles parts, analz et synth

Il est naturel de définir sur l'ensemble de messages $Fields$ des opérateurs qui permettent de décrire le pouvoir de l'intrus. Ainsi $\mathsf{parts}(S)$ désigne l'ensemble maximal des sous-termes des messages de S qui pourrait connaître l'intrus, c'est-à-dire tous les sous-termes de S qui n'apparaissent pas en position de clefs.

Définition 4.3 $\mathsf{parts}(S)$ *est le plus petit ensemble* S' *tel que :*

1. $S \subseteq S'$ *;*
2. *si* $< m_1, m_2 > \in S'$ *alors* $m_1, m_2 \in S'$ *;*
3. *si* $\{m\}_k \in S'$ *alors* $m \in S'$*.*

L'opérateur analz calcule l'ensemble exact des sous termes d'un ensemble de messages S, accessibles à l'intrus à partir de S.

Définition 4.4 $\mathsf{analz}(S)$ *est le plus petit ensemble* S' *tel que :*

1. $S \subseteq S'$ *;*
2. *si* $< m_1, m_2 > \in S'$ *alors* $m_1, m_2 \in S'$ *;*
3. *si* $\{m\}_k \in S'$ *et* $k^{-1} \in S'$ *alors* $m \in S'$*.*

Inversement, $\mathsf{synth}(S)$ désigne l'ensemble des messages que l'on peut construire à partir de S.

Définition 4.5 $\mathsf{synth}(S)$ *est le plus petit ensemble* S' *tel que :*

1. *$S \subseteq S'$;*
2. *si $m_1, m_2 \in S'$ alors $< m_1, m_2 > \in S'$;*
3. *si $m, k \in S'$, $k \in Key \cup \mathcal{K}_L$ alors $\{m\}_k \in S'$;*
4. *si $m \in S'$ alors $\mathsf{h}(m) \in S'$.*

Les opérateurs parts, analz et synth sont des opérateurs de clôture qui vérifient les relations décrites ci-dessous.

Proposition 4.1 ([MR00]) *Soit $S \subseteq Fields$ un ensemble de messages. Les égalités suivantes sont vérifiées :*

$$\begin{array}{rcl} \mathsf{parts}(\mathsf{analz}(S)) & = & \mathsf{parts}(S), \\ \mathsf{analz}(\mathsf{parts}(S)) & = & \mathsf{parts}(S), \\ \mathsf{parts}(\mathsf{synth}(S)) & = & \mathsf{parts}(S) \cup \mathsf{synth}(S), \\ \mathsf{synth}(\mathsf{analz}(S)) & = & \mathsf{analz}(S) \cup \mathsf{synth}(S). \end{array}$$

En conséquence, l'ensemble synth(analz(S)) est clos par analz et synth :

$$\mathsf{analz}(\mathsf{synth}(\mathsf{analz}(S))) = \mathsf{synth}(\mathsf{analz}(S)). \tag{4.1}$$

Remarque : Cette propriété repose sur le fait que les clefs sont atomiques. Lorsque les clefs sont composées, une alternation arbitrairement profonde d'opérations d'analyse et de synthèse peut être nécessaire pour connaître un terme.

Ainsi, soient $s, k \in Nonce$ et $(t_n)_n$ la suite de termes définie de la manière suivante :

$$t_0 =< \{s\}_{<k,k>}, k > \quad \text{et} \quad t_{n+1} =< \{t_n\}_{<\{t_n\}_k, \{t_n\}_k>}, \{t_n\}_k > .$$

Il faut alors au moins $n+2$ alternations des opérateurs analz et synth pour atteindre s à partir de l'ensemble $\{t_n\}$. Plus précisément, $s \notin (\mathsf{synth} \circ \mathsf{analz})^{n+1}(\{t_n\})$ et $s \in (\mathsf{synth} \circ \mathsf{analz})^{n+2}(\{t_n\})$.

A partir d'un ensemble de messages S, l'intrus peut potentiellement construire tous les messages accessibles par combinaisons d'opérations d'analyse et de synthèse. La propriété 4.1 motive donc la définition de l'ensemble fake(S) qui représente tous les messages connus de l'intrus à partir de S.

Définition 4.6 *Soit S un ensemble de messages. On définit :*

$$\mathsf{fake}(S) \stackrel{def}{=} \mathsf{synth}(\mathsf{analz}(S)).$$

4.1.1.3 Idéaux

L'objet de cette partie est de définir un sur-ensemble de l'ensemble des secrets à protéger. Cet ensemble, appelé *idéal*, est clos par concaténation avec des messages arbitraires et clos par chiffrement par des clefs dont l'inverse n'est pas dans l'ensemble considéré. En effet, si le message $< s, m >$ est compromis, alors s l'est aussi. De la même manière, si un message $\{s\}_k$ est compromis et si k ne fait pas partie des secrets, alors s peut être aussi compromis.

Définition 4.7 (Idéal) *Soit S un ensemble de messages, soient $m, m_1, m_2 \in$ Fields des messages et $k \in Key \cup \mathcal{K}_L$ une clef. On définit l'*idéal *de S, noté $\mathcal{I}(S)$, comme le plus petit ensemble tel que :*

1. *$S \subseteq \mathcal{I}(S)$;*
2. *si $m_1 \in \mathcal{I}(S)$ ou si $m_2 \in \mathcal{I}(S)$ alors $< m_1, m_2 > \in \mathcal{I}(S)$;*
3. *si $m \in \mathcal{I}(S)$ et si $k^{-1} \notin \mathcal{I}(S)$ alors $\{m\}_k \in \mathcal{I}(S)$.*

Si on suppose que tous les termes qui ne sont pas dans l'idéal peuvent être compromis alors il est nécessaire de protéger tous les termes de l'idéal pour garantir le secret. En effet, dans ce cas, il est suffisant qu'un terme de l'idéal soit compromis pour qu'un terme de S le soit également. Nous verrons dans la partie 4.1.3 en quoi l'idéal n'est cependant, en général, qu'une sur-approximation de l'ensemble des secrets à protéger et non l'ensemble exact.

Le complémentaire d'un idéal $\mathcal{I}(S)$, appelé *coidéal* de S et noté $Coideal(S)$, est l'ensemble des messages qui peuvent être révélés sans compromettre, en aucun cas, les termes de S.

Proposition 4.2 ([MR00]) *Soit S un ensemble de messages. L'ensemble $Coideal(S)$ est clos par analyse :*

$$\mathsf{analz}(Coideal(S)) = Coideal(S).$$

La même propriété est vraie pour la synthèse, à condition de ne considérer que des messages atomiques.

Proposition 4.3 ([MR00]) *Soit S un ensemble de messages atomiques, c'est-à-dire $S \subseteq Agent \cup Nonce \cup Key$. L'ensemble $Coideal(S)$ est clos par synthèse :*

$$\mathsf{synth}(Coideal(S)) = Coideal(S).$$

Remarque : Si les messages ne sont pas atomiques, la propriété est fausse. Ainsi, considérons l'ensemble $S = \{\{s\}_k\}$. Comme $\mathcal{I}(S)$ ne contient que des messages m tels que $\{s\}_k$ est un sous terme de m, s et k ne sont pas dans l'idéal engendré par S, c'est-à-dire, $s, k \in Coideal(S)$. Donc $\{s\}_k \in \mathsf{synth}(Coideal(S))$ et pourtant $\{s\}_k \notin Coideal(S)$.

Par conséquent, si S est un ensemble de messages atomiques, l'ensemble des messages calculables par l'intrus à partir de $Coideal(S)$ reste dans $Coideal(S)$:

$$\mathsf{fake}(Coideal(S)) = Coideal(S).$$

Ces ensembles nous permettent de définir les règles d'un protocole.

4.1.2 Protocole

Un protocole permet trois sortes d'*événements* : l'émission d'un message, la mise à jour d'un état local d'un agent et la déclaration qu'une donnée doit rester secrète. Un protocole spécifie quels événements peuvent être ajoutés à partir d'un historique donné.

4.1.2.1 Événements et historique

Message Le premier type d'événements est l'émission d'un message. Il est représenté tout simplement par le terme qui correspond au message envoyé.

État local Le second type d'événement est la déclaration de l'état local d'un agent. Il est noté $\mathsf{a}_{i,j}(m)$ où $a \in Agent$ est un terme représentant un agent, i est un entier naturel qui représente le numéro de rôle joué par a, j est un entier naturel qui représente l'étape du protocole et $m \in Fields$ est un message représentant la mémoire de l'agent. Cet événement correspond au terme $st(a, i, j, m)$ dans notre premier modèle sous forme de clauses de Horn. Ainsi, le terme $\mathsf{a}_{1,2}(< n_a, n_b >)$ signifie que l'agent a joue le premier rôle et est à l'étape 2 du protocole, en ayant mémorisé les nonces n_a et n_b. Un même agent peut être dans différents états en même temps, cela signifie qu'il participe à plusieurs sessions en même temps. Il pourra en particulier confondre des messages provenant de sessions différentes si le protocole ne permet justement pas de faire la distinction.

Définition 4.8 *L'ensemble des états, noté States, est défini de la manière suivante :*

$$States \stackrel{def}{=} \{\mathsf{a}_i(m) \mid a \in Agent, i \in \mathbb{N}, m \in Fields\}.$$

Étant donné un état $q \in States$, $q = \mathsf{a}_{i,j}(m)$, on note $Mem(\mathsf{a}_{i,j}(m)) \stackrel{def}{=} m$.

Spécification d'un secret Le troisième type d'événements est le moins habituel dans la description des protocoles cryptographiques. Il s'agit de la *spécification* d'un secret sous la forme d'un terme $C = S \ddagger L$. L'ensemble S désigne un ensemble de messages basiques : $S \subset Basic$ et l'ensemble L un ensemble d'agents : $L \subset Agent$. Intuitivement, le terme $S \ddagger L$ spécifie que les messages de S ne doivent être connus qu'au plus par les agents de L.

Définition 4.9 *L'ensemble des termes de la forme $C = S \ddagger L$ est noté Spec.*

$$Spec \stackrel{def}{=} \{S \ddagger L \mid S \subset Basic, L \subset Agent\}.$$

De plus, si $C = S \ddagger L$, on définit $SpecSec(C) \stackrel{def}{=} S$ et $SpecAgents(C) \stackrel{def}{=} L$. De plus, on associe à chaque spécification, *un ensemble de secrets qui correspond à l'ensemble des secrets basiques S augmenté des clefs privées à long terme des agents :*

$$Sec(C) \stackrel{def}{=} SpecSec(C) \cup ltk(SpecAgents(C)).$$

On verra par la suite qu'au lieu de protéger seulement l'ensemble de secrets S de $C = S \ddagger L$, on cherche à protéger aussi les clefs à long terme des agents de L susceptibles de détenir les secrets de S.

Définition 4.10 *L'ensemble des événements, noté Event, est l'union des ensembles Fields, States et Spec.*

Historique Un *historique* est tout simplement un ensemble d'événements. Intuitivement, il contient l'ensemble des messages déjà envoyés sur le réseau, les états locaux des agents pour chacune des sessions en cours et les secrets déjà spécifiés.

Notation : Par convention, la lettre H et ses variantes (H_i, ...) désignent un ensemble d'événements. L'ensemble des messages d'un historique H est noté

$$Msg(H) \stackrel{\text{def}}{=} H \cap Fields.$$

De la même façon, l'ensemble des valeurs « à garder secrètes » d'un historique H est défini par :

$$Sec(H) \stackrel{\text{def}}{=} \{Sec(C) | C \in H\}.$$

Un message basique est dit *frais* dans H, s'il n'apparaît nul part dans H. Plus précisément, on définit :

$$Frais(H) \stackrel{\text{def}}{=} \{X \in Basic \mid X \notin \mathsf{parts}(Msg(H)),\, X \notin Sec(H),\, X \notin \bigcup_{S \in H \cap States} Mem(S)\}.$$

Cette définition est utilisée pour garantir la fraîcheur des nonces créés au cours du protocole.

4.1.2.2 Règles du protocole

Un protocole spécifie les transitions possibles d'un historique à un autre.

Définition 4.11 *Une* transition de protocole t *est une règle de la forme*

$$Pre(t) \stackrel{New(t)}{\longrightarrow} Post(t)$$

telle que $Pre(t)$ *et* $Post(t)$ *sont des ensembles d'événements et* $New(t)$ *est un ensemble de messages basiques. De plus,* t *doit vérifier les conditions suivantes.*

1. *Tout message atomique qui apparaît dans* $Post(t)$ *(c'est-à-dire qui est sous-terme d'un terme de* $Post(t)$*) apparaît aussi dans* $Pre(t)$ *ou* $New(t)$*. Cette condition est appelée* régularité.
2. *Pour tout événement* $e = S \ddagger L$ *de* $Post(t)$*,* S *est inclus dans* $New(t)$*. Cette condition est appelée* fraîcheur.
3. *Pour tout couple d'événements* $e_1 = S_1 \ddagger L_1$ *et* $e_2 = S_2 \ddagger L_2$ *de* $Post(t)$*, les ensembles* S_1 *et* S_2 *sont disjoints.*
4. *Il n'y a pas à la fois des messages et des spécifications dans* $Post(t)$*.*
5. $Pre(t) \cap States$ *est soit vide, soit un singleton. Dans ce dernier cas,* $Post(t) \cap States$ *est aussi un singleton et concerne le même agent pour le même rôle :*

$$Pre(t) \cap States = \{\mathsf{a}_{j,i}(m_1)\} \quad \text{et} \quad Post(t) \cap States = \{\mathsf{a}_{j,i+1}(m_2)\}.$$

Intuitivement, la règle $Pre(t) \stackrel{New(t)}{\longrightarrow} Post(t)$ spécifie que si les événements de $Pre(t)$ sont dans un historique accessible H et si les nonces de $New(t)$ n'apparaissent pas déjà dans H (cette hypothèse sera explicitée dans la définition des transitions possibles entre historiques) alors l'historique H auquel on a ajouté les événements de $Post(t)$ est accessible. La condition de régularité assure qu'aucune donnée non définie n'est introduite dans un historique. La condition de fraîcheur assure, elle, que seules des données nouvelles sont déclarées secrètes. La dernière condition assure que la transition t spécifie au plus un changement d'état pour un agent.

Un protocole est simplement un ensemble de spécifications de transitions.

Définition 4.12 (protocole) *Un* protocole P du modèle MR *est un ensemble fini de règles rl de la forme $Pre(rl) \stackrel{New(rl)}{\longrightarrow} Post(rl)$ telle que pour toute règle rl de P et pour toute substitution close bien sortée σ de rl, alors $rl\sigma$ est une* transition de protocole. *Les transitions d'un protocole P sont les transitions de protocole, instances d'une des règles du protocole P.*

Lorsqu'il n'y a pas d'ambiguïté, on utilisera le terme de *protocole* au lieu de *protocole du modèle MR*.

4.1.2.3 Exemple

Nous reprenons l'exemple du protocole de Yahalom, introduit au chapitre 3.

$$
\begin{array}{rl}
\textbf{Yahalom :} & 1.\ A \to B : A, N_a \\
& 2.\ B \to S : B, \{A, N_a, N_b\}_{\mathsf{shr}(B)} \\
& 3.\ S \to A : \{B, K_{ab}, N_a, N_b\}_{\mathsf{shr}(A)}, \{A, K_{ab}\}_{\mathsf{shr}(B)} \\
& 4.\ A \to B : \{A, K_{ab}\}_{\mathsf{shr}(B)}, \{N_b\}_{K_{ab}}
\end{array}
$$

Le protocole P du modèle MR correspondant au protocole de Yahalom est construit de la manière suivante.

Initialisation

$$\emptyset \stackrel{N_b, K_{ab}}{\longrightarrow} \left\{ \begin{array}{l} \mathsf{A}_{1,1}(A, B, srv), \mathsf{B}_{2,1}(B, srv, N_b), srv_{3,1}(srv, K_{ab}), \\ \{N_b\} \ddagger \{A, B\}, \{K_{ab}\} \ddagger \{A, B\} \end{array} \right\}$$

Cette première règle spécifie que les participants A, B et le serveur srv sont prêts à commencer une nouvelle session du protocole. De plus, B et le serveur engendrent au préalable les nonces ou clefs N_b et K_{ab} qui vont être utilisés dans la session et qui sont déclarés secrets. Ainsi, la spécification $\{N_b\} \ddagger \{A, B\}$ indique que N_b n'a le droit d'être connu que de A et B. Comme le membre gauche de cette règle est vide, cette règle peut s'appliquer à tout moment.

Envoi du premier message

$$\{\mathsf{A}_{1,1}(A, B, srv)\} \stackrel{N_a}{\longrightarrow} \{< A, N_a >, \mathsf{A}_{1,2}(A, B, srv, N_a)\}$$

Si l'agent A est dans son état initial, il engendre un nouveau nonce N_a ; envoie le message $< A, N_a >$ et passe dans l'état $\mathsf{A}_{1,2}(A, B, srv, N_a)$ qui signale que l'agent est à la deuxième étape du protocole et qu'il a mémorisé le nonce engendré N_a.

Réponse du deuxième participant

$$\{\mathsf{B}_{2,1}(B, srv, N_b), \{N_b\} \ddagger \{A, B\}, < A, X >\} \stackrel{\emptyset}{\longrightarrow} \left\{ \begin{array}{l} \mathsf{B}_{2,2}(B, srv, N_b), \\ < B, \{A, X, N_b\}_{\mathsf{shr}(B)} > \end{array} \right\}$$

Si l'agent B est dans son état initial et s'il reçoit un message de la forme $< A, X >$ (la variable X peut être instanciée par n'importe quel message), alors B utilise son nonce N_b engendré

$$\emptyset \xrightarrow{N_b, K_{ab}} \left\{\begin{array}{l}\mathsf{A}_{1,1}(A, B, srv), \mathsf{B}_{2,1}(B, srv, N_b), \\ srv_{3,1}(srv, K_{ab}), \\ \{N_b\} \ddagger \{A, B\}, \{K_{ab}\} \ddagger \{A, B\}\end{array}\right\} \quad (4.2)$$

$$\{\mathsf{A}_{1,1}(A, B, srv)\} \xrightarrow{N_a} \{< A, N_a >, \mathsf{A}_{1,2}(A, B, srv, N_a)\} \quad (4.3)$$

$$\left\{\begin{array}{l}\mathsf{B}_{2,1}(B, srv, N_b), \{N_b\} \ddagger \{A, B\}, \\ < A, X >\end{array}\right\} \xrightarrow{\emptyset} \left\{\begin{array}{l}\mathsf{B}_{2,2}(B, srv, N_b), \\ < B, \{A, X, N_b\}_{\mathsf{shr}(B)} >\end{array}\right\} \quad (4.4)$$

$$\left\{\begin{array}{l}srv_{3,1}(srv, K_{ab}), \{K_{ab}\} \ddagger \{A, B\}, \\ < B, \{A, X, Y\}_{\mathsf{shr}(B)} >\end{array}\right\} \xrightarrow{\emptyset} \left\{\begin{array}{l}srv_{3,2}(srv), \{A, K_{ab}\}_{\mathsf{shr}(B)} \\ < \{B, K_{ab}, X, Y\}_{\mathsf{shr}(A)} >\end{array}\right\} \quad (4.5)$$

$$\left\{\begin{array}{l}\mathsf{A}_{1,2}(A, B, srv, N_a), \\ < \{B, X, N_a, Y\}_{\mathsf{shr}(A)}, Z >\end{array}\right\} \xrightarrow{\emptyset} \left\{\begin{array}{l}\mathsf{A}_{1,3}(A, B, srv, N_a, X), \\ < Z, \{Y\}_X >\end{array}\right\} \quad (4.6)$$

FIGURE 4.1 - *Le protocole de Yahalom, exprimé dans le modèle MR.*

en début de session et supposé secret pour envoyer le message $< B, \{A, X, N_b\}_{\mathsf{shr}(B)} >$. De plus, B change d'état local.

Les autres règles sont construites de manière similaires. L'ensemble des règles correspondant au protocole de Yahalom est décrit figure 4.1.

Remarque : Dans la spécification du protocole de Yahalom décrit à la figure 4.1, le protocole est peu typé : nous supposons uniquement que les agents savent distinguer un nom d'agent d'un message quelconque mais nous ne supposons pas qu'un agent sache faire la différence entre un nonce engendré par un autre participant et un message composé. On suppose bien sûr qu'un agent saura toujours reconnaître les nonces qu'il a lui-même engendré au cours de la session. Inversement, nous présentons au chapitre 7 des protocoles où presque toutes les variables sont typées. En effet, cela est nécessaire pour le bon fonctionnement de notre outil, comme annoncé au chapitre 2.

4.1.2.4 Transitions

Un protocole s'exécute à partir d'un historique vide. L'étude d'un protocole revient à l'étude de l'ensemble des historiques accessibles *en suivant le protocole*. L'objet de cette partie est de définir, étant donné un protocole, les historiques accessibles.

Définition 4.13 (successeur [CMR01]) *Soit P un protocole et I un ensemble de messages, représentant la connaissance initiale de l'intrus. Soient H et H' deux historiques.*

- *H' est un* successeur honnête *de H, noté $honest(P, I)(H, H')$, s'il existe une* transition applicable *t de P telle que*

$$H' = (H \backslash (Pre(t) \cap States)) \cup Post(t).$$

Une transition t est dite applicable *à H si $Pre(t) \subseteq H$ et si $New(t) \subseteq Frais(H)$.*

- *H' est un* successeur malhonnête *de H, noté $fake(P, I)(H, H')$, s'il existe un message $m \in \mathsf{fake}(Msg(H) \cup I)$ tel que $H' = H \cup \{m\}$.*

Dans les deux cas, on dit que H' est un successeur *de H :*

$$global(P,I)(H,H') \Leftrightarrow honest(P,I)(H,H') \vee fake(P,I)(H,H').$$

Comme on suppose que le système de transition part de l'historique vide, on définit l'ensemble des historiques accessibles, noté $reachable(P,I)$, comme le plus petit ensemble tel que :
- $\emptyset \in reachable(P,I)$;
- *si* $H \in reachable(P,I)$ *et* $global(P,I)(H,H')$ *alors* $H' \in reachable(P,I)$.

Exemple 4.1 *Soient $a, b \in Agent$, $n_1, n_2 \in Nonce$ et $k, k' \in Key$. Soit P le protocole décrit à la figure 4.1 et $I = \{k'\}$. On considère*

$$\begin{aligned}
H_0 &= \{\mathsf{a}_{1,1}(a,b,srv), \mathsf{b}_{2,1}(b,srv,n_2), srv_{3,1}(srv,k), \{n_2\} \ddagger \{a,b\}, \{k\} \ddagger \{a,b\}\},\\
H_1 &= \{\mathsf{a}_{1,2}(a,b,srv,n_1), \mathsf{b}_{2,1}(b,srv,n_2), srv_{3,1}(srv,k), \{n_2\} \ddagger \{a,b\}, \{k\} \ddagger \{a,b\}, <a,n_1>\},\\
H_2 &= \{\mathsf{a}_{1,2}(a,b,srv,n_1), \mathsf{b}_{2,1}(b,srv,n_2), srv_{3,1}(srv,k), \{n_2\}\ddagger\{a,b\}, \{k\}\ddagger\{a,b\}, <a,n_1>, \{k'\}_{n_1}\}.
\end{aligned}$$

Alors les propriétés suivantes sont vérifiées : $honest(P,I)(\emptyset, H_0)$,$honest(P,I)(H_0,H_1)$ et $fake(P,I)(H_1,H_2)$, donc $H_0, H_1, H_2 \in reachable(P,I)$.

4.1.3 Secret

Il nous reste à définir la notion de secret.

4.1.3.1 Définition

Une spécification est dite *compatible* avec la connaissance initiale I de l'intrus si les secrets de la spécification n'apparaissent pas comme parties des messages de I.

Définition 4.14 (spécification compatible [CMR01]) *Soit I un ensemble de messages représentant la connaissance initiale de l'intrus. Alors l'ensemble des spécifications* compatibles *avec I, noté $compatible(I)$ est défini par :*

$$compatible(I) \stackrel{def}{=} \{C \mid Sec(C) \cap \mathsf{parts}(I) = \emptyset\}.$$

Définition 4.15 (protocole impénétrable [CMR01]) *Soit I un ensemble de messages représentant la connaissance initiale de l'intrus. Un historique H est dit I-*impénétrable *si pour toute spécification C de H, compatible avec I, alors les messages de H restent dans le coidéal de C :*

$$Msg(H) \subseteq Coideal(Sec(C)).$$

Un protocole est impénétrable *si pour toute connaissance initiale I de l'intrus, pour tout historique H accessible, H est I-impénétrable.*

Exemple 4.2 *Soient $a, b \in Agent$, $n_1, n_2 \in Nonce$ et $k \in Key$. Soit*

$$I = \{c, \mathsf{shr}(c), \mathsf{prv}(c), \mathsf{pub}(c), b, \mathsf{pub}(b), a, \mathsf{pub}(a)\},$$

l'intrus connaît une identité c dont il possède toutes les clefs et il connaît les clefs publiques et les identités des agents a et b. On considère :

$$H \stackrel{def}{=} \{srv_{3,2}(srv), \mathsf{a}_{1,3}(a, b, srv, n_1, k), \mathsf{b}_{2,2}(b, srv, n_2), \{n_2\} \ddagger \{a, b\}, \{k\} \ddagger \{a, b\}, \\ < a, n_1 >, < b, \{a, n_1, n_2\}_{\mathsf{shr}(b)} >, < \{b, k, n_1, n_2\}_{\mathsf{shr}(a)}, \{a, k\}_{\mathsf{shr}(b)} >, \\ < \{a, k\}_{\mathsf{shr}(b)}, \{n_2\}_k >\}.$$

H est un historique accessible pour le protocole de Yahalom P et H correspond à l'historique obtenu après le déroulement normal d'une seule session du protocole. Or $\{n_2\}_k \in \mathcal{I}(\{n_2\} \ddagger \{a, b\})$ car $k = k^{-1} \notin \mathcal{I}(\{n_2\} \ddagger \{a, b\})$. Donc P n'est pas impénétrable.

On voit ici que cette notion de discrétion introduite dans [MR00] est une notion plus forte que la notion habituelle du secret. Ainsi, un jeu normal d'une seule session du protocole suffit à montrer que le protocole de Yahalom n'est pas impénétrable et pourtant l'intrus ne connaît ni le nonce n_2 $(= N_b)$, ni la clef k $(= K_{ab})$. De plus, la notion de discrétion est vérifiée *pour toute connaissance initiale de l'intrus.* Or on pourrait vouloir vérifier le secret des données du protocole, seulement pour certaines connaissances initiales de l'intrus bien choisies. Cela motive une autre définition du secret.

Définition 4.16 (secret) *Un protocole préserve le* secret *pour la connaissance initiale I de l'intrus si pour tout historique H accessible et pour toute spécification C de H, compatible avec I, alors les secrets de C ne sont pas connus de l'intrus :* $\mathsf{fake}(Msg(H) \cup I) \cap Sec(C) = \emptyset$.

Remarque : Si un protocole P est impénétrable, alors pour toute connaissance initiale I de l'intrus, P préserve le secret pour I.

Exemple 4.3 *En reprenant les notations de l'exemple 4.2, H n'est plus un contre-exemple à la propriété de secret car on a bien :*

$$\mathsf{fake}(Msg(H) \cup I) \cap Sec(\{n_2\} \ddagger \{a, b\}) = \emptyset \quad \textit{et} \quad \mathsf{fake}(Msg(H) \cup I) \cap Sec(\{k\} \ddagger \{a, b\}) = \emptyset$$

Cependant, l'intérêt de la notion de discrétion est qu'elle possède de bonnes propriétés ce qui permet de démontrer plus facilement la discrétion d'un protocole et donc, *a fortiori*, de garantir le secret du protocole.

4.1.3.2 Propriétés

Tout d'abord, par construction des protocoles et de la propriété de fraîcheur, un même nonce ne peut pas apparaître dans deux spécifications différentes.

Proposition 4.4 (Secrets disjoints [MR00]) *Soient P un protocole et I un ensemble de messages. Soient H un historique accessible ($H \in reachable(P, I)$) et $C_1, C_2 \in H$, $C_1 = S_1 \ddagger L_1$, $C_2 = S_2 \ddagger L_2$. Si $S_1 \cap S_2 \neq \emptyset$ alors $C_1 = C_2$.*

J. Millen et H. Rueß ont montré qu'on pouvait séparer la preuve de la discrétion en une partie dépendante du protocole et une partie indépendante du protocole qu'ils ont établie une fois pour toute. La partie dépendante du protocole consiste à vérifier que pour tout historique H accessible ne compromettant aucun secret, alors les événements ajoutés par les successeurs directs de H ne compromettent pas non plus de secret.

Définition 4.17 (confidentialité [MR00]) *Soit P un protocole. P préserve la* confidentialité *si*

- *pour tout ensemble de messages I, représentant la connaissance de l'intrus,*
- *pour tout historique H accessible ($H \in reachable(P, I)$) tel que H est I-impénétrable,*
- *pour toute spécification C de H, tel que C est compatible avec I,*
- *pour toute transition t applicable à H,*

alors :

$$Msg(Post(t)) \subseteq Coideal(Sec(C)).$$

Théorème 4.1 (discrétion [MR00])
Soit P un protocole. P est impénétrable si et seulement si P préserve la confidentialité.

Ce théorème permet de simplifier les preuves et est à la base de notre procédure de vérification de la discrétion d'un protocole du modèle MR. Cette procédure est présentée au chapitre 7.

Malgré ce théorème de réduction, la discrétion d'un protocole reste indécidable : il est par exemple toujours possible de coder l'accessibilité d'un réseau de Petri avec Reset comme cela est fait dans [CC01].

4.2 Comparaison du modèle de clauses et du modèle de Millen-Rueß

Le modèle Millen-Rueß apparaît moins expressif que le modèle sous forme de clauses de Horn, présenté au chapitre 3. En particulier, il ne permet pas de modéliser les clefs composées. De plus, il ne permet d'exprimer qu'une seule propriété : le secret. L'objet de cette partie est de montrer que le modèle du chapitre 3 capture bien tout le modèle Millen-Rueß. Nous allons donc montrer formellement comment associer à tout protocole P du modèle Millen-Rueß, un ensemble de clauses de Horn C_P et un ensemble fini de propriétés tels que le protocole P est secret si et seulement s'il n'y a pas d'attaque sur C_P pour les propriétés considérées.

Nous rappelons que les messages sont construits sur l'ensemble des symboles de fonctions :

$$\mathcal{F}_{\mathsf{msg}} \stackrel{\mathrm{def}}{=} \{<\_,\_>, \{\_\}_{\_}, \mathsf{h}, \mathsf{pub}, \mathsf{prv}, \mathsf{shr}\}.$$

Soit P un protocole du modèle Millen-Rueß et I_0 un ensemble de messages représentant la connaissance initiale de l'intrus.

Le modèle Millen-Rueß permet de typer les messages. Pour exprimer le typage dans le modèle sous forme de clauses, on introduit deux nouveaux prédicats Nonce et Clef définis respectivement par les ensembles $\mathcal{C}_{\mathcal{N}}$ et $\mathcal{C}_{\mathcal{K}}$, qui sont eux-mêmes construit au fur et à mesure de la traduction du protocole. Dans les clauses correspondant aux règles du protocole, ces prédicats permettent de tester si les messages reçus sont bien des nonces ou des clefs.

4.2.1 Traduction de la connaissance initiale

On suppose que l'ensemble I_0 des connaissances initiales de l'intrus est définissable par un ensemble $\mathcal{I}_0$ de clauses de Horn, c'est-à-dire : $m \in I_0$ si et seulement si $\mathcal{I}_0 \vdash I_0(m)$. L'ensemble

I_0 définit naturellement les agents honnêtes et malhonnêtes : les agents malhonnêtes sont ceux dont l'intrus connaît au moins une clef privée. On pose DA l'ensemble des agents malhonnêtes :

$$\mathsf{DA} \stackrel{\text{def}}{=} \{a \mid \mathsf{prv}(a) \in \mathsf{parts}(I_0) \quad \text{ou} \quad \mathsf{shr}(a) \in \mathsf{parts}(I_0)\}.$$

L'ensemble des agents honnêtes est l'ensemble des agents qui ne sont pas malhonnêtes :

$$\mathsf{HA} \stackrel{\text{def}}{=} \mathit{Agent} \backslash \mathsf{DA}.$$

On définit alors une substitution qui envoie les agents honnêtes sur les termes de la forme $s_h^n(h)$ et les agents malhonnêtes sur les termes de la forme $s_d^n(d)$. Soit σ une substitution bijective :

$$\sigma : \mathit{Agent} \to \{s_h^n(h) \mid n \in \mathbb{N}\} \cup \{s_d^n(d) \mid n \in \mathbb{N}\},$$

telle que $\sigma(\mathsf{HA}) = \{s_h^n(h) \mid n \in \mathbb{N}\}$ et $\sigma(\mathsf{DA}) = \{s_d^n(d) \mid n \in \mathbb{N}\}$.

Pour chaque nonce n apparaissant dans I_0, on définit $\sigma(n) = c_n$ où c_n est un nouveau symbole de constante, et on ajoute à $\mathcal{C_N}$ la clause :

$$\mathsf{Nonce}(c_n).$$

De même, pour chaque clef k, on définit $\sigma(k) = c_k$ et on ajoute à $\mathcal{C_K}$ la clause :

$$\mathsf{Clef}(c_k).$$

On étend σ sur les messages par $\sigma(f(m_1, \dots m_p)) = f(\sigma(m_1), \dots, \sigma(m_p))$ pour tout $f \in \mathcal{F}$.

Enfin, on construit l'ensemble de clauses de Horn $\mathcal{C}_{I_0}$ à partir de $\mathcal{I}_0$ en renommant chaque message atomique $m \in \mathit{Agent} \cup \mathit{Nonce} \cup \mathit{Key}$ par $\sigma(m)$ et en renommant le prédicat I_0 par I'_0. De plus, on ajoute à $\mathcal{C}_{I_0}$ la clause :

$$I'_0(m), T(t) \Rightarrow I(m,t). \tag{4.7}$$

Cette clause permet à l'intrus d'utiliser sa connaissance initiale à tout moment du protocole. On vérifie facilement que $\mathcal{I}_0 \vdash I_0(m)$ si et seulement si $\mathcal{C}_{I_0} \vdash I'_0(\sigma(m))$.

4.2.2 Traduction du protocole

Nous définissons ici une construction qui, à chaque règle rl du protocole P, associe une clause contrainte $\overline{rl}$.

On commence par étendre σ à l'ensemble M_P des messages atomiques des règles de P : $\sigma(n) = c_n$ pour $n \in \mathit{Nonce} \cap M_P$ et $\sigma(k) = c_k$ pour $k \in \mathit{Key} \cap M_P$. De plus, on augmente les ensembles $\mathcal{C_N}$ et $\mathcal{C_K}$:

$$\mathsf{Nonce}(c_n) \in \mathcal{C_N} \text{ si } n \in \mathit{Nonce} \cap M_P \quad \text{et} \quad \mathsf{Clef}(c_k) \in \mathcal{C_K} \text{ si } k \in \mathit{Key} \cap M_P.$$

Ces nonces et ces clefs correspondent à des constantes du protocole. Pour tout message atomique m apparaissant dans les règles de P, on définit $\overline{m} = \sigma(m)$.

4.2.2.1 Variables

Nous allons maintenant définir $\overline{\cdot}$ sur les variables de P. On suppose tout d'abord que, pour toutes règles rl et rl', les variables à valeurs dans $Nonce$ ou Key apparaissant dans $New(rl)$ et celles apparaissant dans $New(rl')$ sont toutes distinctes. De plus, on se donne une numérotation de ces variables : il existe une fonction injective numero qui a toute variable x à valeurs dans $Nonce$ ou Key du protocole P associe un entier $\mathsf{numero}(x) \in \mathbb{N}$.

Exemple 4.4 *Nous prenons pour exemple tout au long de cette partie, le protocole de Yahalom, présenté au paragraphe 4.1.2.3, figure 4.1. Nous commençons donc par numéroter les variables de nonces ou de clefs :*

$$\mathsf{numero}(N_a) = 1, \quad \mathsf{numero}(N_b) = 2, \quad \mathsf{numero}(K_{ab}) = 3.$$

Une traduction de ce protocole est présentée à la figure 4.2.

On suppose fixée une règle rl du protocole P. La transformation $\overline{\cdot}$ est redéfinie pour chaque règle.

Soit $\mathcal{A}$ l'ensemble des variables de rl à valeur dans $Agent$ et $\mathcal{M}$ l'ensemble des variables de rl à valeur dans $Fields$. La transformation $\overline{\cdot}$ laisse les éléments de $\mathcal{A}$ et $\mathcal{M}$ inchangés.

Soit $\mathcal{N}_1$ (resp. $\mathcal{K}_1$) l'ensemble des variables à valeur dans $Nonce$ (resp. dans Key) apparaissant dans $Pre(rl)$ et $\mathcal{N}_2$ (resp. $\mathcal{K}_2$) l'ensemble des variables à valeur dans $Nonce$ (resp. dans Key) apparaissant dans $New(rl)$. Si on suppose qu'au moins une instance de cette règle est applicable à un certain historique H, alors la condition de fraîcheur sur les transitions de protocole impose que $\mathcal{N}_1$ et $\mathcal{N}_2$ d'une part, $\mathcal{K}_1$ et $\mathcal{K}_2$ d'autre part sont disjoints. Comme rl est une règle de protocole, toutes les variables de $Post(rl)$ sont des variables de $Pre(rl)$ ou $New(rl)$. Il suffit donc de définir la transformation $\overline{\cdot}$ pour les variables de $Pre(rl)$ et $New(rl)$. On suppose que les spécifications de $Post(rl)$ et $Pre(rl)$ sont numérotées.

1. Si $x \in \mathcal{N}_1$ alors x est laissé inchangé : $\overline{x} = x$.
2. Si $x \in \mathcal{N}_2$ et si x n'apparaît dans aucune spécification de $Post(rl)$, alors x représente un nouveau nonce sur lequel on ne demande aucune propriété de secret. On pose $\overline{x} = n(\mathsf{numero}(x), [\,], s)$ et on ajoute à $\mathcal{C}_\mathcal{N}$ la clause :

$$\Rightarrow \mathsf{Nonce}(n(\mathsf{numero}(x), [\,], s)) \quad \mid \mathsf{Num}(s).$$

3. Enfin, si $x \in \mathcal{N}_2$ et si x apparaît dans une spécification $C = S \ddagger \{a_1, \ldots, a_k\}$ de $Post(rl)$, on pose $\overline{x} = n(\mathsf{numero}(x), [a_1; \ldots; a_k], s)$ pour un certain ordre des a_i. De plus, on ajoute à $\mathcal{C}_\mathcal{N}$ la clause :

$$\Rightarrow \mathsf{Nonce}(n(\mathsf{numero}(x), [a_1; \ldots; a_k], s)) \quad \mid \mathsf{Num}(s) \wedge \mathsf{Agent}(a_1) \wedge \cdots \wedge \mathsf{Agent}(a_k).$$

La transformation $\overline{\cdot}$ est définie exactement de la même manière pour les variables de $\mathcal{K}_1 \cup \mathcal{K}_2$ à ceci près qu'on ajoute à $\mathcal{C}_\mathcal{K}$ des clauses de la forme :

$$\Rightarrow \mathsf{Clef}(n(\mathsf{numero}(x), [\,], s)) \quad \mid \mathsf{Num}(s).$$

Exemple 4.5 (suite de l'exemple 4.4) *On considère la première règle du protocole de Yahalom :*

$$rl_1 = \emptyset \xrightarrow{N_b, K_{ab}} \left\{ \begin{array}{l} \mathsf{A}_{1,1}(A, B, srv), \mathsf{B}_{2,1}(B, srv, N_b), \\ srv_{3,1}(srv, K_{ab}), \\ \{N_b\} \ddagger \{A, B\}, \{K_{ab}\} \ddagger \{A, B\} \end{array} \right\}.$$

Alors $\mathcal{N}_1 = \mathcal{K}_1 = \emptyset$ *et* $\mathcal{N}_2 = \{N_b\}$ $\mathcal{K}_2 = \{K_{ab}\}$. $\overline{N_b} = n(2, [A; B], s)$, $\overline{K_{ab}} = n(3, [A; B], s)$ *et les clauses :*

$$\begin{array}{ll} \Rightarrow \mathsf{Nonce}(n(2, [A; B], s)) & \mid \mathsf{Agent}(A) \wedge \mathsf{Agent}(B) \wedge \mathsf{Num}(s) \\ \Rightarrow \mathsf{Clef}(n(3, [A; B], s)) & \mid \mathsf{Agent}(A) \wedge \mathsf{Agent}(B) \wedge \mathsf{Num}(s) \end{array}$$

sont ajoutées respectivement à $\mathcal{C}_\mathcal{N}$ *et* $\mathcal{C}_\mathcal{K}$.

4.2.2.2 Règles

On étend la transformation $\bar{\cdot}$ aux événements de la façon suivante :

$$\left\{ \begin{array}{rcl} \overline{f(t_1, \ldots, t_n)} & = & f(\overline{t_1}, \ldots, \overline{t_n}) \quad f \in \mathcal{F}_{\mathsf{msg}} \\ \overline{\mathsf{A}_{i,j}(m)} & = & st(\overline{A}, s^i(0), s^j(0), \overline{m}) \\ \overline{\{s_1, \ldots, s_k\} \ddagger \{a_1, \ldots, a_n\}} & = & \mathsf{cast}([\overline{s_1}; \ldots; \overline{s_k}], [\overline{a_1}; \ldots; \overline{a_n}]) \end{array} \right.$$

où les variables $s_1, \ldots, s_k$ sont numérotées dans un ordre compatible avec l'ordre numero déjà existant sur les variables de type nonce ou clef.

La règle rl considérée est de la forme

$$Pre(rl) \xrightarrow{New(rl)} \{m_1, \ldots, m_k, st_1, \ldots, st_m, e_1, \ldots, e_l\}$$

où les m_i sont des messages, les st_i des états et les e_i des spécifications. D'après la définition des règles de protocole, deux cas sont possibles : ou bien $Pre(t) \cap States$ est vide, ou bien c'est un singleton. Dans le premier cas, la clause contrainte $\overline{rl}$ correspondant à la règle rl est définie comme suit :

$$\begin{array}{rl} \overline{rl} \stackrel{\text{def}}{=} \neg T(t) \vee \neg \mathsf{Fresh}(s, t) & \vee \displaystyle\bigvee_{m \in Pre(rl) \cap Fields} \neg I(\overline{m}, t) \\ & \vee \displaystyle\bigvee_{C \in Pre(rl) \cap Spec} \neg \mathsf{In}([\overline{C}, s_C], t) \\ & \vee \displaystyle\bigvee_{N \in \mathcal{N}_1} \neg \mathsf{Nonce}(N) \vee \bigvee_{K \in \mathcal{K}_1} \neg \mathsf{Clef}(K) \\ & \vee \; T([\overline{m_1}, s] \cdots [\overline{m_k}, s] \cdot [\overline{st_1}, s] \cdots [\overline{st_m}, s] \cdot [\overline{e_1}, s] \cdots [\overline{e_l}, s] \cdot t) \\ & \quad \mid C_{rl} \end{array} \tag{4.8}$$

où la contrainte C_{rl} est définie par :

$$C_{rl} \stackrel{\text{def}}{=} \bigwedge_{A \in \mathcal{A}} \mathsf{Agent}(A) \wedge \bigwedge_{X \in \mathcal{M}} \mathsf{Message}(X) \wedge \mathsf{Num}(s) \wedge \bigwedge_{C \in Pre(rl) \cap Spec} \mathsf{Num}(s_C).$$

Si $Pre(t) \cap States = \{st\}$ est un singleton, la clause contrainte $\overline{rl}$ correspondant à la règle

rl est définie comme suit. On pose $st = \mathsf{A}_{i,j}(m)$. $Post(t) \cap States = \{\mathsf{A}_{i,j+1}(m')\}$.

$$\begin{aligned}
\overline{rl} \stackrel{\text{def}}{=} \neg T(t) \quad & \vee \bigvee_{m \in Pre(rl) \cap Fields} \neg I(\overline{m}, t) \\
& \vee \neg \mathsf{In}([\overline{st}, s], t) \vee \neg \mathsf{NotPlayed}(\overline{A}, s^i(0), s^{j+1}(0), s, t) \\
& \vee \bigvee_{C \in Pre(rl) \cap Spec} \neg \mathsf{In}([\overline{C}, s_C], t) \qquad (4.8) \\
& \vee \bigvee_{N \in \mathcal{N}_1} \neg \mathsf{Nonce}(N) \vee \bigvee_{K \in \mathcal{K}_1} \neg \mathsf{Clef}(K) \\
& \vee T([\overline{m_1}, s] \cdots [\overline{m_k}, s] \cdot [st(A, s^i(0), s^{j+1}(0), \overline{m'}), s] \cdot [\overline{e_1}, s] \cdots [\overline{e_l}, s] \cdot t) \\
& \quad \mid C_{rl}
\end{aligned}$$

où la contrainte C_{rl} est définie comme précédemment.

Dans les deux cas, la clause contrainte peut se lire de la manière suivante :

– si t est une trace valide pour le protocole,

– si s est un numéro de session plus grand que tous ceux déjà utilisés dans la trace,

– si l'intrus peut produire les messages correspondants aux messages de $Pre(rl)$,

– et si les états et les spécifications correspondants à ceux de $Pre(rl)$ sont dans la trace, alors on peut appliquer la règle et ajouter à la trace les événements correspondants à ceux de $Post(rl)$.

Remarque : Le prédicat $\mathsf{NotPlayed}(\overline{A}, s^i(0), s^{j+1}(0), s, t)$ permet d'assurer que la $j^{\text{ième}}$ règle du protocole n'a pas déjà été jouée pour le rôle i de la session s. En effet, dans une trace valide, on peut avoir, pour une même session et pour un même agent, des états différents : la trace garde l'historique de tous les états par lesquels sont passés les agents. Le « véritable » état de l'agent considéré correspond en fait à l'état qui possède le numéro d'étape i le plus élevé. Dans le modèle Millen-Rueß, les états sont effacés au fur et à mesure que les agents avancent dans la session.

En fait, la clause $\overline{rl}$ ne suffit pas à représenter la règle rl. En effet, comme les ensembles $\{s_1, \ldots, s_k\}$ et $\{a_1, \ldots, a_n\}$ des spécifications sont représentées par des listes, il faut considérer toutes les permutations possibles des éléments de l'ensemble lorsqu'on cherche si une spécification de la forme $\{s_1, \ldots, s_k\} \ddagger \{a_1, \ldots, a_n\}$ est dans la trace. C'est pourquoi on définit :

$$\overline{\{s_1, \ldots, s_k\} \ddagger \{a_1, \ldots, a_n\}}^{\mathcal{E}} \stackrel{\text{def}}{=} \bigcup_{\theta, \theta' \in \mathcal{P}(\{1, \ldots, k\})} \left\{\mathsf{cast}([\overline{s_{\theta(1)}}; \ldots; \overline{s_{\theta(k)}}], [\overline{a_{\theta'(1)}}; \ldots; \overline{a_{\theta'(n)}}])\right\}$$

où $\mathcal{P}(\{1, \ldots, k\})$ désigne l'ensemble des permutations de $\{1, \ldots, k\}$.

L'ensemble $\overline{rl}^{\mathcal{E}}$ des clauses qui représente $\overline{rl}$ est l'union pour t_C membre de $\overline{C}^{\mathcal{E}}$, des clauses $\overline{rl}$ où la ligne 4.8 de la clause est remplacée par

$$\vee \bigvee_{C \in Pre(rl) \cap Spec} \neg \mathsf{In}([t_C, s_C], t).$$

Exemple 4.6 (suite de l'exemple 4.4) *On considère la $4^{\grave{e}me}$ règle du protocole de Yahalom :*

$$rl_4 = \left\{ \begin{array}{l} srv_{3,1}(srv, K_{ab}), \{K_{ab}\} \ddagger \{A, B\}, \\ < B, \{A, X, Y\}_{\mathsf{shr}(B)} > \end{array} \right\} \xrightarrow{\emptyset} \left\{ \begin{array}{l} srv_{3,2}(srv), \\ < \{B, K_{ab}, X, Y\}_{\mathsf{shr}(A)}, \{A, K_{ab}\}_{\mathsf{shr}(B)} > \end{array} \right\}.$$

Elle contient une unique spécification, et deux représentations sous forme de termes sont possibles pour cette spécification :

$$\mathsf{cast}([K_{ab}],[A;B]) \quad \textit{et} \quad \mathsf{cast}([K_{ab}],[B;A]).$$

L'ensemble $\overline{rl_4}^{\mathcal{E}}$ *est donc composé de deux clauses, suivant la représentation choisie. Nous n'écrivons ici que la première de ces clauses :*

$$\begin{array}{ll} \neg T(t) & \vee \ \ \neg I(<B,\{A,X,Y\}_{\mathsf{shr}(B)}>,t) \\ & \vee \ \ \neg \mathsf{In}([st(\sigma(srv),3,1,<\sigma(srv),K_{ab}>),s],t) \vee \neg \mathsf{NotPlayed}(\sigma(srv),3,2,s,t) \\ & \vee \ \ \neg \mathsf{In}([\mathsf{cast}([K_{ab}],[A;B]),s_v],t) \vee \neg \mathsf{Clef}(K_{ab}) \\ & \vee \ \ T\left([<\{B,K_{ab},X,Y\}_{\mathsf{shr}(A)},\{A,K_{ab}\}_{\mathsf{shr}(B)}>,s]\cdot[st(\sigma(srv),3,2,\sigma(srv)),s]\cdot t\right) \\ & \qquad \mid \mathsf{Agent}(A) \wedge \mathsf{Agent}(B) \wedge \mathsf{Message}(X) \wedge \mathsf{Message}(Y) \\ & \qquad\qquad\qquad\qquad\qquad\qquad \wedge \mathsf{Num}(s_u) \wedge \mathsf{Num}(s_v) \wedge \mathsf{Num}(s) \end{array}$$

où $v = \{K_{ab}\} \ddagger \{A,B\}$.

4.2.2.3 Spécification du secret

Il reste à définir les propriétés de sécurité induites par les spécifications introduits dans les règles du protocole. Soit rl une règle du protocole, on définit :

$$\Phi_{rl} \stackrel{\text{def}}{=} \bigcup_{\substack{\{s_1,\ldots,s_k\}\ddagger\{a_1,\ldots,a_q\}\in Post(rl) \\ 1\leqslant i\leqslant k}} \neg T(t) \vee \neg I(\overline{s_i},t) \quad \mid \mathsf{Ha}(a_1) \wedge \cdots \wedge \mathsf{Ha}(a_q) \wedge \mathsf{Num}(s).$$

L'ensemble des propriétés de sécurité associées à un protocole P est alors :

$$\Phi_P \stackrel{\text{def}}{=} \bigcup_{rl\in P} \Phi_{rl} \cup \left\{ \begin{array}{l} \neg I(\mathsf{shr}(a),t) \vee \neg T(t) \mid \mathsf{Ha}(a), \\ \neg I(\mathsf{prv}(a),t) \vee \neg T(t) \mid \mathsf{Ha}(a) \end{array} \right\},$$

à moins que les règles de P ne contiennent aucune spécification, auquel cas $\Phi_P \stackrel{\text{def}}{=} \emptyset$: il n'y aucune propriété à vérifier. Les clauses $\neg I(\mathsf{shr}(a),t) \vee \neg T(t) \mid \mathsf{Ha}(a)$ et $\neg I(\mathsf{prv}(a),t) \vee \neg T(t) \mid \mathsf{Ha}(a)$ permettent de tenir compte du fait que la définition du secret d'un protocole assure également la protection des clefs secrètes et à long terme des agents.

Exemple 4.7 (suite de l'exemple 4.4) *L'ensemble des propriétés de sécurité associées au protocole de Yahalom est :*

$$\begin{array}{rl} \Phi_{\mathsf{Yah}} = \{ & \neg T(t) \vee \neg I(n(2,[a;b],s) \mid \mathsf{Ha}(a) \wedge \mathsf{Ha}(b) \wedge \mathsf{Num}(s), \\ & \neg T(t) \vee \neg I(n(3,[a;b],s) \mid \mathsf{Ha}(a) \wedge \mathsf{Ha}(b) \wedge \mathsf{Num}(s), \\ & \neg I(\mathsf{shr}(a),t) \vee \neg T(t) \mid \mathsf{Ha}(a), \\ & \neg I(\mathsf{prv}(a),t) \vee \neg T(t) \mid \mathsf{Ha}(a)\}. \end{array}$$

Enfin, l'ensemble $\mathcal{C}_P$ des clauses contraintes associées au protocole P est l'union

$$\mathcal{C}_P \stackrel{\text{def}}{=} \mathcal{I}_0 \cup \mathcal{C}_{I_1} \cup \mathcal{C}_{I_2} \cup \mathcal{C}_{\mathsf{aux}} \cup C_{\mathcal{N}} \cup C_{\mathcal{K}} \cup \mathcal{C}_{\mathsf{regles}},$$

où $\mathcal{C}_{I_1}$ et $\mathcal{C}_{I_2}$ ont été définis au paragraphe 3.4.1.1 et $\mathcal{C}_{\mathsf{aux}}$ au paragraphe 3.4.1.2 et :

$$\mathcal{C}_{\mathsf{regles}} \stackrel{\text{def}}{=} \bigcup_{rl\in P} \overline{rl}^{\mathcal{E}}.$$

Exemple 4.8 (suite et fin de l'exemple 4.4) *Les ensembles de clauses $C_{\mathcal{N}}$, $C_{\mathcal{K}}$ et $\mathcal{C}_{\mathsf{regles}}$ associés au protocole de Yahalom sont décrits à la figure 4.2.*

4.2.2.4 Correspondance

Cette traduction vérifie bien la correspondance voulue.

Théorème 4.2 *Soit P un protocole du modèle Millen-Rueß. On suppose que pour toute instanciation des variables d'agents et pour tout historique H, toutes les règles du protocole peuvent être jouées dans un certain ordre à partir de H. Soit I_0 un ensemble de messages représentant la connaissance initiale de l'intrus, tel que I_0 puisse être défini par un ensemble de clauses de Horn. Soit $\mathcal{C}_P$ l'ensemble de clauses correspondant à P, défini au paragraphe précédent, et Φ_P l'ensemble des propriétés de sécurité associées à P.*

Alors P n'est pas secret pour la connaissance initiale I_0 si et seulement s'il existe une attaque sur $\mathcal{C}_P$ pour une des propriétés de Φ_P.

La preuve de ce théorème est présentée en annexe A. Elle consiste à vérifier par induction que les historiques accessibles pour P et les traces valides pour $\mathcal{C}_P$ se correspondent.

Remarque : L'hypothèse selon laquelle, à tout moment et pour toute instanciation des variables d'agents, toutes les règles du protocole peuvent être jouées dans un certain ordre est une hypothèse toujours vérifiée par les protocoles cryptographiques. En effet, à tout moment, les agents peuvent jouer une session « normale » du protocole.

Cette hypothèse est en particulier utile dans la preuve montrant que s'il existe un historique H tel que l'intrus apprend la clef privée d'un agent honnête A, alors, quitte à augmenter l'historique, on peut supposer qu'il y a au moins une spécification $C = S \ddagger L$ dans H tel que a fait partie de L.

Clauses définissant les nonces.

$$C_{\mathcal{N}} = \{ \begin{array}{ll} \mathsf{Nonce}(n(1,[\,],s)) & \mid \mathsf{Num}(s) \\ \mathsf{Nonce}(n(2,[A;B],s)) & \mid \mathsf{Agent}(A) \wedge \mathsf{Agent}(B) \wedge \mathsf{Num}(s)\} \end{array}$$

Clauses définissant les clefs.

$$C_{\mathcal{K}} = \{\mathsf{Clef}(n(3,[A;B],s)) \quad \mid \mathsf{Agent}(A) \wedge \mathsf{Agent}(B) \wedge \mathsf{Num}(s)\}$$

Clauses représentant les règles du protocole.
On ne représente ici qu'une clause par ensemble $\overline{rl}^{\mathcal{E}}$ pour chaque règle rl du protocole.

Règle 4.2

$$\begin{array}{ll} \neg T(t) & \vee \quad \neg\mathsf{Fresh}(s,t) \\ & \vee \quad T\left(\begin{array}{l} [st(A,1,1,<A,B,\sigma(srv)>),s]\cdot[st(B,2,1,<B,\sigma(srv),n(2,[A;B],s)>),s] \\ \cdot[st(\sigma(srv),3,1,<\sigma(srv),n(3,[A;B],s)>),s] \\ \cdot[\mathsf{cast}([n(2,[A;B],s)],[A;B]),s]\cdot[\mathsf{cast}([n(3,[A;B],s)],[A;B]),s]\cdot t \end{array}\right) \\ & \mid \mathsf{Agent}(a) \wedge \mathsf{Agent}(b) \wedge \mathsf{Num}(s) \end{array}$$

Règle 4.3

$$\begin{array}{ll} \neg T(t) & \vee \quad \neg\mathsf{In}([st(A,1,1,<A,B,\sigma(srv)>),s],t) \vee \neg\mathsf{NotPlayed}(A,1,2,s,t) \\ & \vee \quad I([<A,n(1,[\,],s)>,s]\cdot[st(A,1,2,<A,B,\sigma(srv),n(1,[\,],s)>),s]\cdot t) \\ & \quad \mid \mathsf{Agent}(a) \wedge \mathsf{Agent}(b) \wedge \mathsf{Num}(s) \wedge \mathsf{Num}(s_u) \end{array}$$

Règle 4.4

$$\begin{array}{ll} \neg T(t) & \vee \quad \neg\mathsf{In}([st(B,2,1,<B,\sigma(srv),N_b)>),s],t) \vee \neg\mathsf{NotPlayed}(B,2,2,s,t) \\ & \vee \quad \neg\mathsf{In}([\mathsf{cast}([N_b],[A;B]),s_v],t) \vee \neg I(<A,X>,t) \\ & \vee \quad \neg\mathsf{Nonce}(N_b) \\ & \vee \quad I([<B,\{A,X,N_b\}_{\mathsf{shr}(B)}>,s]\cdot[st(B,2,2,<B,\sigma(srv),N_b>),s]\cdot t) \\ & \quad \mid \mathsf{Agent}(a) \wedge \mathsf{Agent}(b) \wedge \mathsf{Message}(X) \wedge \mathsf{Num}(s) \wedge \mathsf{Num}(s_u) \wedge \mathsf{Num}(s_v) \end{array}$$

où $v = \{N_b\} \ddagger \{A,B\}$.

Règle 4.5

$$\begin{array}{ll} \neg T(t) & \vee \quad \vee\neg I(<B,\{A,X,Y\}_{\mathsf{shr}(B)}>,t) \\ & \vee \quad \neg\mathsf{In}([st(\sigma(srv),3,1,<\sigma(srv),K_{ab}>),s],t) \vee \neg\mathsf{NotPlayed}(\sigma(srv),3,2,s,t) \\ & \vee \quad \neg\mathsf{In}([\mathsf{cast}([K_{ab}],[A;B]),s_v],t) \vee \neg\mathsf{Clef}(K_{ab}) \\ & \vee \quad T\left([<\{B,K_{ab},X,Y\}_{\mathsf{shr}(A)},\{A,K_{ab}\}_{\mathsf{shr}(B)}>,s]\cdot[st(\sigma(srv),3,2,\sigma(srv)),s]\cdot t\right) \\ & \quad \mid \mathsf{Agent}(A) \wedge \mathsf{Agent}(B) \wedge \mathsf{Message}(X) \wedge \mathsf{Message}(Y) \\ & \qquad\qquad \wedge\mathsf{Num}(s_u) \wedge \mathsf{Num}(s_v) \wedge \mathsf{Num}(s) \end{array}$$

où $v = \{K_{ab}\} \ddagger \{A,B\}$.

Règle 4.6

$$\begin{array}{ll} \neg T(t) & \vee \quad \vee\neg I(<\{B,X,N_a,Y\}_{\mathsf{shr}(A)},Z>,t) \vee \neg\mathsf{Nonce}(N_a) \\ & \vee \quad \neg\mathsf{In}([st(A,1,2,<A,B,c_{Srv},N_a>),s],t) \vee \neg\mathsf{NotPlayed}(A,1,3,s,t) \\ & \vee \quad T([<Z,\{Y\}_X>,s]\cdot[st(A,1,2,<A,B,c_{Srv},N_a,X>),s]\cdot t) \\ & \quad \mid \mathsf{Agent}(A) \wedge \mathsf{Agent}(B) \wedge \mathsf{Message}(X) \wedge \mathsf{Message}(Y) \wedge \mathsf{Message}(Z) \\ & \qquad\qquad \wedge\mathsf{Num}(s_u) \wedge \mathsf{Num}(s) \end{array}$$

FIGURE 4.2 - *Traduction du protocole de Yahalom en clauses de Horn.*

CHAPITRE 5

Le Spi-Calcul

Le spi-calcul est un langage de processus appliqué aux protocoles cryptographiques. Il s'agit une extension du pi-calcul [MPW92] avec des primitives cryptographiques. Il est proche du langage CSP (*Communicating Sequential Processes*) [Sch97, RSG⁺00] et du langage développé par R. Amadio *et al.* dans [AL00, ALV02]. Il permet en particulier d'exprimer l'envoi et la réception de messages, la création de nonces, la réplication et la mise en parallèle de processus ; certaines constructions sont dédiées à la modélisation du chiffrement et du déchiffrement.

Les langages de processus présentent une différence majeure avec les autres modèles de protocoles cryptographiques. Ces derniers expriment en effet les propriétés de sécurité comme des propriétés d'accessibilité : « il existe une trace valide telle que l'intrus peut déduire le secret » ou « le processus peut atteindre l'état *erreur* ». Pour les langages de processus comme le spi-calcul, le secret ainsi que les autres propriétés peuvent être exprimés sous forme d'équivalences observationnelles et l'intrus peut être représenté par n'importe quel processus descriptible dans la syntaxe. On dit ainsi qu'un protocole $P(s)$ est secret si $P(s)$ et $P(s')$ sont observationnellement équivalents quel que soit le processus mis en parallèle. Une telle propriété prend en compte, par exemple, la capacité d'un intrus à comparer les messages, ce qui n'est en général pas le cas dans les autres modèles.

Malheureusement, il est très difficile de faire la preuve de l'équivalence observationnelle de deux processus. Ainsi, il n'existe, à notre connaissance, aucun outil ne permettant de prouver l'équivalence observationnelle entre processus. Une possibilité est d'introduire un autre calcul et une autre notion d'équivalence, plus proches des systèmes de traces et plus faciles à manipuler, dont on montre une fois pour toutes l'équivalence avec le premier calcul. L'intérêt d'un tel résultat est double : d'une part, il fournit une méthode de preuve pour l'équivalence observationnelle ; d'autre part, il permet d'établir un lien entre le spi-calcul et les modèles de traces couramment utilisés pour les protocoles cryptographiques.

Ainsi, M. Boreale *et al.* [BDNP99] ont montré dans un contexte restreint que l'équivalence observationnelle est équivalente à une forme de bisimulation utilisant une équivalence « statique ». Cependant, le cadre considéré ne permet pas d'exprimer les clefs composées ni le chiffrement asymétrique. D'autre part, M. Abadi et C. Fournet [AF01] ont établi un résultat similaire dans un contexte extrêmement général. Mais même l'équivalence statique est indécidable dans ce cadre.

Notre contribution est de généraliser le résultat de M. Boreale en préservant la décidabilité de l'équivalence statique. Pour cette généralisation, la définition du secret sous forme d'équivalence observationnelle n'est plus pertinente ce qui nous amène à en proposer une nouvelle. Nous appliquons ce nouveau résultat à la preuve du protocole de Needham-Schroeder-Lowe pour un nombre arbitraire de sessions, ce qui n'avait encore jamais été fait dans le contexte du spi-calcul. Ce résultat a fait l'objet d'un rapport de recherche [Cor02b].

Dans la première partie de ce chapitre, nous définissons le spi-calcul et l'équivalence observationnelle associée. Nous reprenons en grande partie les notions développées dans [AG97, BDNP99]. Nous introduisons ensuite un calcul de processus en présence d'un environnement. Ce calcul est une extension de celui présenté par M. Boreale. La troisième partie du chapitre est consacrée à la preuve de l'équivalence des deux notions d'équivalence introduites. Nous appliquons enfin ces résultats à la preuve du protocole de Needham-Schroeder-Lowe et nous les comparons en particulier à ceux de M. Boreale et M. Abadi.

5.1 Langage

Nous définissons ici le spi-calcul et l'équivalence observationnelle qui lui est associée.

5.1.1 Syntaxe

Comme M. Abadi, nous considérons un ensemble arbitraire de symboles de fonction pour construire les termes du langage. Comme M. Boreale, nous considérons des processus gardés par des formules mais nous ajoutons un ensemble arbitraire de prédicats représentant des ensembles réguliers de termes.

5.1.1.1 Symboles de fonction

L'ensemble des symboles de fonction est noté $\mathcal{F}$. Il se décompose en quatre sous-ensembles suivant la sémantique sous-jacente des symboles de fonction considérés.

noms : l'ensemble $\mathcal{F}$ contient un ensemble infini dénombrable de constantes $\mathcal{N}$, représentant les noms. Ces noms sont utilisés pour représenter les identités des agents, les nonces et les clefs engendrés.

symboles inversibles : l'ensemble $\mathcal{IF}$ correspond aux constructions dont les composantes peuvent être calculées par l'intrus. Ainsi, la paire de messages, notée $< m_1, m_2 >$ appartient à l'ensemble $\mathcal{IF}$.

symboles one-way : l'ensemble $\mathcal{OF}$ correspond aux constructions dont aucune des composantes ne peut être calculée par l'intrus. Ainsi, les fonctions de hachage appartiennent à l'ensemble $\mathcal{OF}$. De plus, on suppose que les trois symboles de fonctions unaires pub, prv et shr, correspondant aux constructeurs des clefs publiques, privées et partagées avec le serveur, appartiennent à l'ensemble $\mathcal{OF}$.

symboles partiellement inversibles : ils correspondent aux constructions dont le calcul des composantes est soumis à une certaine connaissance de l'intrus. En fait, le seul symbole de fonction partiellement inversible considéré ici est le chiffrement. Un tel symbole de fonction binaire, noté $\{m\}_k$ vérifie que l'intrus peut calculer m à partir de $\{m\}_k$ seulement s'il connaît l'inverse de la clef k.

L'inverse d'un message est défini de la même façon qu'aux chapitres 3 et 4 : l'inverse d'un message k, notée k^{-1}, est k lui-même sauf si k est égal à $\mathsf{pub}(m)$ ou $\mathsf{prv}(m)$, auxquels cas $\mathsf{pub}(m)^{-1} = \mathsf{prv}(m)$ et $\mathsf{prv}(m)^{-1} = \mathsf{pub}(m)$.

En résumé, $\mathcal{F} = \mathcal{N} \uplus \mathcal{IF} \uplus \mathcal{OF} \uplus \{\{\_\}\_\}$ avec $\mathsf{pub}, \mathsf{prv}, \mathsf{shr} \in \mathcal{OF}$. Le symbole $\uplus$ représente l'union disjointe.

De plus, certains symboles de fonctions sont accessibles par l'intrus (comme la paire ou le chiffrement) tandis que d'autres sont privés (comme le symbole constructeur des clefs privées : prv). Cela nous conduit à décomposer l'ensemble $\mathcal{F}$ en deux nouveaux ensembles : $\mathcal{PF}$, l'ensemble des *symboles privés* et $\mathcal{VF}$, l'ensemble des *symboles publics* ou *visibles*. Notons que cette seconde décomposition est indépendante de la première : la première décomposition correspond aux capacités d'analyse de l'intrus tandis que la seconde correspond aux capacités de synthèse. On a ainsi :

$$\mathcal{F} = \mathcal{N} \uplus \mathcal{IF} \uplus \mathcal{OF} \uplus \{\{\_\}\_\} = \mathcal{PF} \uplus \mathcal{VF}.$$

On note $\mathcal{M}$ l'ensemble des termes construits sur l'alphabet $\mathcal{F}$, aussi appelés *messages*.

Remarque : Dans ce calcul de processus, il n'y a pas de distinction entre constantes et variables. Tous les noms peuvent être *a priori* touchés par une substitution. Aussi, on emploiera parfois le mot de variable au lieu de nom.

5.1.1.2 Calcul

L'ensemble de la syntaxe du spi-calcul est résumé à la figure 5.1.

Notation : Soit f un symbole de fonction. L'arité de f est désignée par $arity(f)$. Pour le symbole de la paire $<>$, on notera parfois π_1 l'opérateur $<>_1^{-1}$ et π_2 l'opérateur $<>_2^{-1}$.

Les *noms* et les *messages* ont déjà été définis au paragraphe précédent.

Les *expressions publiques* sont de la forme $f(\zeta_1, \dots, \zeta_n)$ si $f \in \mathcal{VF}$, $g^{-1}(\zeta)$ si $g \in \mathcal{IF}$ et $\mathsf{dec}_\eta(\zeta)$. Elles correspondent à la construction des messages en appliquant des fonctions publiques et à la décomposition de messages en utilisant l'inverse des fonctions inversibles ou partiellement inversibles. Ainsi l'expression publique $\mathsf{dec}_\eta(\zeta)$ est évaluée en déchiffrant le message correspondant à ζ à l'aide de la valeur obtenue pour η.

L'ensemble des *formules* logiques est paramétré par un ensemble arbitraire de prédicats V_i tels que leur interprétation soit un langage régulier de termes (c'est-à-dire reconnaissable par un automate d'arbre). On rappelle que les automates d'arbre ont été défini au chapitre 3. Ainsi, on pourra exprimer la capacité des agents à reconnaître certains types de messages comme les clefs, les messages hachés. Notons qu'il s'agit là d'un choix laissé lors de la spécification des capacités des agents et de l'intrus : nos résultats seront valables quel que soit l'ensemble des V_i choisi. Paramétrer ainsi le spi-calcul par un ensemble de prédicats permet en particulier une grande flexibilité sur le choix du typage du protocole.

Exemple 5.1 *Pour exprimer que les agents sont capables de reconnaître les messages hachés, on introduit le prédicat V tel que $S_V = \{h(m)\}$.*

Les *processus* sont construits exactement comme dans [AG97, BDNP99]. Ils permettent en particulier d'exprimer l'émission et la réception de messages sur des canaux particuliers. Ainsi, le processus $a(x) \cdot \overline{b}(h(x))$ représente le processus qui attend un message sur le *canal* a et qui

$$
\begin{array}{lcll}
a, b, \ldots, h, k, \ldots, x, y, z, \ldots & & & \text{noms } \mathcal{N} \\[1em]
M, N, M_1, \ldots, M_n & ::= & a \mid \{M\}_N \mid f(M_1, \ldots, M_n) & \begin{array}{l}\text{messages } \mathcal{M} \\ \forall f \in \mathcal{IF} \uplus \mathcal{OF}\end{array} \\[1em]
 & & & \text{expressions publiques } \mathcal{Z} \\
\zeta, \eta, \zeta_1, \ldots, \zeta_n & ::= & a \mid f(\zeta_1, \ldots, \zeta_n) & \forall f \in \mathcal{VF} \supseteq \{<\_\,,\_>, \{\_\}\_\}, \\
 & & \mid \mathsf{dec}_\eta(\zeta) \mid g_i^{-1}(\zeta) & \forall g \in \mathcal{IF},\ \forall i \leqslant arity(g) \\[1em]
\phi, \psi & ::= & tt \mid [\zeta = \eta] \mid \mathsf{let}\ z = \zeta\ \mathsf{in}\ \phi & \text{formules } \Phi \\
 & & \mid V_1(\zeta) \mid \ldots \mid V_n(\zeta) & V_i \text{ prédicats} \\
 & & \mid \phi \wedge \psi \mid \phi \vee \psi \mid \neg\phi & \\
P, Q & ::= & & \text{processus } \mathcal{P}roc \\
 & & \mathbf{0} & \text{(zéro)} \\
 & \mid & \eta(x).P & \text{(réception)} \\
 & \mid & \overline{\eta}(\zeta).P & \text{(émission)} \\
 & \mid & P + Q & \text{(choix non déterministe)} \\
 & \mid & P \mid Q & \text{(composition parallèle)} \\
 & \mid & (\nu a)P & \text{(restriction)} \\
 & \mid & !P & \text{(réplication)} \\
 & \mid & \phi P & \text{(garde booléenne)} \\
 & \mid & \mathsf{let}\ z = \zeta\ \mathsf{in}\ P & \text{(variable locale)}
\end{array}
$$

FIGURE 5.1 - *Syntaxe du spi-calcul.*

renvoie le message haché sur le canal b. Dans la modélisation des protocoles cryptographiques, on considère en général que tous les canaux sont publics ; aussi dans les exemples qui suivent, nous considérons un unique canal public p.

Par rapport aux travaux de M. Boreale et M. Abadi, nous introduisons la notion de *protocoles* qui sont des processus dans lesquels les messages peuvent être également construits à partir des symboles de fonction privés. Les protocoles sont donc définis exactement de le même manière que les processus excepté pour la définition des expressions publiques, qui sont remplacées par les *expressions* définies de la manière suivante :

$$\zeta, \eta, \zeta_1, \ldots, \zeta_n ::= a \mid f(\zeta_1, \ldots, \zeta_n) \mid \mathsf{dec}_\eta(\zeta) \mid g_i^{-1}(\zeta) \qquad \forall f \in \mathcal{F}, \forall g \in \mathcal{IF},\ \forall i \leqslant arity(g).$$

L'ensemble des expressions est noté $\mathcal{ZP}$. Notons que l'ensemble $\mathcal{ZP}$ contient $\mathcal{Z}$ et que les processus sont des protocoles particuliers.

Cette distinction entre processus et protocole nous permet de distinguer la partie relative aux règles du protocole, pouvant utiliser les constructeurs privés, de la partie qui représente l'intrus utilisant uniquement les constructeurs publics.

Notation : $\tilde{b}$ désigne une suite finie $b_1, \ldots, b_k$ et $(\nu\tilde{b})P$ est une notation qui désigne le protocole $(\nu b_1) \cdots (\nu b_k)P$.

5.1.2 Sémantique opérationnelle

Nous définissons ici les règles d'évaluation des expressions et des formules ainsi que les règles d'inférence des processus.

5.1.2.1 Évaluation des expressions

La fonction d'évaluation des expressions $\widehat{\cdot} : \mathcal{ZP} \to \mathcal{M} \uplus \{\perp\}$ est définie par induction de la manière suivante :

$$\begin{array}{rcl}
\widehat{a} & = & a \\
\widehat{f(\zeta_1,\ldots,\zeta_n)} & = & \left\{\begin{array}{ll} f(M_1,\ldots,M_n) & \text{si } \widehat{\zeta_1} = M_1,\ldots,\widehat{\zeta_n} = M_n,\ f \in \mathcal{VF} \\ \perp & \text{sinon,} \end{array}\right. \\
\widehat{\mathsf{dec}_{\zeta_2}(\zeta_1)} & = & \left\{\begin{array}{ll} M & \text{si } \widehat{\zeta_1} = \{M\}_k \text{ et } \widehat{\zeta_2} = k^{-1}, \\ \perp & \text{sinon,} \end{array}\right. \\
\widehat{f_i^{-1}(\zeta)} & = & \left\{\begin{array}{ll} M_i & \text{si } \widehat{\zeta} = f(M_1,\ldots,M_n),\ f \in \mathcal{IF} \\ \perp & \text{sinon.} \end{array}\right.
\end{array}$$

Comme nous autorisons un nombre arbitraire de symboles de fonctions, nous pensons que les expressions considérées ici sont suffisamment générales pour modéliser la plupart des règles de construction et destruction de la cryptographie. Par contre, nous ne pouvons pas représenter des fonctions comme le « ou » exclusif comme nous le verrons au chapitre 8 et comme cela a été fait dans [CLS03]. Pour de telles fonctions, il faudrait ajouter une théorie équationnelle spécifique comme l'ont fait M. Abadi et C. Fournet [AF01] et nous le faisons pour le modèle sous forme de clauses de Horn, au chapitre 8.

Exemple 5.2 *On considère l'expression* $\zeta = \mathsf{dec}_{<>_1^{-1}(<a,b>)}(\{h(a)\}_a)$*. L'évaluation de* ζ *est* $\widehat{\zeta} = h(a)$*.*

La fonction d'évaluation des formules, $[\![\cdot]\!] : \Phi \to \{\mathit{tt}, \mathit{ff}\}$ est définie par induction. Les seules clauses non standards sont $V_i(\zeta)$ et $\mathsf{let}\, z = \zeta \,\mathsf{in}\, \phi$:

$$\begin{array}{rcl}
[\![V_i(\zeta)]\!] & = & \left\{\begin{array}{ll} \mathit{tt} & \text{si } \widehat{\zeta} \in S_{V_i}, \\ \mathit{ff} & \text{sinon (en particulier si } \widehat{\zeta} = \perp). \end{array}\right. \\
[\![\,\mathsf{let}\, z = \zeta \,\mathsf{in}\, \phi]\!] & = & \left\{\begin{array}{ll} [\![\phi(\zeta/z)]\!] & \text{si } \widehat{\zeta} \neq \perp, \\ \mathit{ff} & \text{sinon.} \end{array}\right.
\end{array}$$

Exemple 5.3 *On considère la formule* $\phi \stackrel{def}{=} \mathsf{let}\, z = \mathsf{dec}_{<>_1^{-1}(<a,b>)}(\{h(a)\}_a) \,\mathsf{in}\, [z = h(a)]$*. Elle est évaluée à vrai :* $[\![\phi]\!] = \mathit{tt}$*.*

Notation : Soit σ une substitution. La propriété $\sigma \models \phi$ est définie par $[\![\phi\sigma]\!] = \mathit{tt}$.

Nous écrirons parfois $[\zeta \neq \perp]$ au lieu de $\mathsf{let}\, z = \zeta \,\mathsf{in}\, \mathit{tt}$ et $[\zeta = \perp]$ au lieu de $\neg(\,\mathsf{let}\, z = \zeta \,\mathsf{in}\, \mathit{tt})$. Les expressions $\mathit{fn}(\zeta)$, $\mathit{fn}(\phi)$ représentent respectivement les variables (ou les noms) libres de ζ et de ϕ.

(INP) $a(x).P \xrightarrow{aM} P[M/x]$ (OUT) $\bar{a}M.P \xrightarrow{\bar{a}<M>} P$

(SUM) $\dfrac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}$ (REP) $\dfrac{!P \xrightarrow{\mu} P'}{P|!P \xrightarrow{\mu} P'}$

(PAR) $\dfrac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q}$ (COM) $\dfrac{P \xrightarrow{(\nu\tilde{b})\bar{a}<M>} P' \quad Q \xrightarrow{aM} Q'}{P|Q \xrightarrow{\tau} (\nu\tilde{b})(P'|Q')}$

(RES) $\dfrac{P \xrightarrow{\mu} P'}{(\nu c)P \xrightarrow{\mu} (\nu c)P'}\ c \notin n(\nu)$ (OPEN) $\dfrac{P \xrightarrow{(\nu\tilde{b})\bar{a}<M>} P'}{(\nu c)P \xrightarrow{(\nu\tilde{b}c)\bar{a}<M>} P'}\ c \neq a, c \in n(M) - \tilde{b}$

(GUARD) $\dfrac{[\![\phi]\!] \quad P \xrightarrow{\mu} P'}{\phi P \xrightarrow{\mu} P'}$ (LET) $\dfrac{\widehat{\zeta} \neq \bot \quad P[\widehat{\zeta}/z] \xrightarrow{\mu} P'}{\mathsf{let}\, z = \zeta\, \mathsf{in}\, P \xrightarrow{\mu} P'}$

FIGURE 5.2 - *Sémantique opérationnelle du spi-calcul (les versions symétriques de* (SUM), (PAR) *et* (COM) *sont omises).*

5.1.2.2 Règles d'inférence

Par convention, on identifie les protocoles et les formules alpha-équivalents, c'est-à-dire qu'on identifie les protocoles ou les formules qui ne diffèrent que par le choix des noms de variables liées. De plus, on suppose que les variables (ou noms) liées par une restriction sont distinctes deux à deux et distinctes des variables libres. On suppose également que les substitutions n'affectent jamais les variables liées.

Les *actions* de protocoles, c'est-à-dire les étiquettes du système de transitions associé aux protocoles, peuvent être de trois formes : τ (action interne), aM (réception du message M sur le canal a) et $\nu(\tilde{b})\bar{a} < M >$ (émission sur le canal a du message M contenant des noms « frais » : $\tilde{b}$). Ces actions sont en général notées μ ou λ. Les actions d'émission et de réception sont dites *visibles*.

La sémantique opérationnelle du spi-calcul est représentée à la figure 5.2. Ainsi, la règle (GUARD) signifie que le processus ϕP se comporte comme le processus P à condition que la formule ϕ soit évaluée à vrai. Dans le cas contraire, le processus ϕP est bloqué. La règle (LET) commence par essayer d'évaluer l'expression ζ : si l'évaluation réussit, elle a alors pour résultat $\widehat{\zeta}$ et le processus $\mathsf{let}\, z = \zeta\, \mathsf{in}\, P$ se comporte comme le processus $P[\widehat{\zeta}/z]$, sinon le processus est bloqué.

Notation : Le symbole s représente une séquence d'actions visibles. La relation $\overset{\epsilon}{\Rightarrow}$ représente la fermeture transitive et réflexive de $\xrightarrow{\tau}$ et la relation $\overset{s}{\Rightarrow}$ est définie par induction par $\Rightarrow\xrightarrow{\mu}\overset{s'}{\Rightarrow}$

si $s = \mu \cdot s'$.

Exemple 5.4 *On considère les protocoles suivants :*

$$\begin{aligned} A(s) &\stackrel{def}{=} \bar{p}(< a, b >) \mid p(x). \mathsf{let}\, z = \mathsf{dec}_{\mathsf{prv}(a)}(x) \,\mathsf{in}\, \bar{p}(\{s\}_z).\mathbf{0}, \\ B &\stackrel{def}{=} p(x). \mathsf{let}\, y = <>_1^{-1}(x) \,\mathsf{in}\, (\nu k)\bar{p}(\{k\}_{\mathsf{pub}(y)}).\mathbf{0}, \\ P(s) &\stackrel{def}{=} A|B. \end{aligned}$$

Intuitivement, le processus A envoie son identité a et celle du processus B pour lui signifier son intention de commencer une communication. Alors (deuxième étape) le processus B peut envoyer à A une nouvelle clef k chiffrée avec la clef publique de a et A répond avec un secret s chiffré avec la clef k qui vient juste d'être reçue (dernière étape). Ces échanges de messages peuvent être formellement représentés par la séquence d'actions suivante :

$$P(s) \xrightarrow{\tau} P'(s)$$

où $P'(s) \stackrel{def}{=} p(x). \mathsf{let}\, z = \mathsf{dec}_{\mathsf{prv}(a)}(x) \,\mathsf{in}\, \bar{p}(\{s\}_z).\mathbf{0} \mid (\nu k)\bar{p}(\{k\}_{\mathsf{pub}(a)}).\mathbf{0}$ *car*

$$A(s) \xrightarrow{\bar{p}<a,b>} A'(s) \quad \textit{et} \quad B \xrightarrow{p(<a,b>)} B',$$

avec $A'(s) \stackrel{def}{=} p(x). \mathsf{let}\, z = \mathsf{dec}_{\mathsf{prv}(a)}(x) \,\mathsf{in}\, \bar{p}(\{s\}_z).\mathbf{0}$ *et* $B' \stackrel{def}{=} (\nu k)\bar{p}(\{k\}_{\mathsf{pub}(a)}).\mathbf{0}$. *Puis*

$$P'(s) \xrightarrow{\tau} P''(s) \xrightarrow{\bar{p}(\{s\}_k)} \mathbf{0},$$

où $P''(s) \stackrel{def}{=} (\nu k)\, \mathsf{let}\, z = \mathsf{dec}_{\mathsf{prv}(a)}(\{k\}_{\mathsf{pub}(a)}) \,\mathsf{in}\, \bar{p}(\{s\}_z).\mathbf{0}$.

5.1.3 Équivalences

Nous rappelons ici la définition classique de l'équivalence structurelle [Mil93, BDNP99] ainsi que les définitions d'équivalence barbue[1] et de testing équivalence, déjà définies dans [AG97, BDNP99].

5.1.3.1 Équivalence structurelle

L'équivalence structurelle est l'équivalence sur les processus la plus simple que l'on puisse attendre.

Définition 5.1 (équivalence structurelle) *L'*équivalence structurelle, *notée* $\equiv$, *est la plus petite (pour l'inclusion) relation d'équivalence sur les processus qui est préservée par la composition parallèle et la restriction, telle que :*

- *pour la composition :* $P|\mathbf{0} \equiv P$, $P|Q \equiv Q|P$ *et* $P|(Q|R) \equiv (P|Q)|R$,
- *pour la restriction :* $(\nu b)\mathbf{0} \equiv \mathbf{0}$, $(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P$ *et* $(\nu a)P|Q \equiv P|(\nu a)Q$ *si* $a \notin fn(P)$,
- *pour la réplication :* $!P \equiv P|!P$.
- *et* $(\mathsf{let}\, z = \eta \,\mathsf{in}\, P) \equiv P[\widehat{\eta}/z]$ *si* $\widehat{\eta} \neq \perp$.

[1] Nous traduisons ainsi le terme de « barbed equivalence » employé en anglais. Cette traduction a déjà été proposée en français.

En particulier, deux processus équivalents peuvent effectuer exactement les mêmes actions. Formellement si $P \equiv Q$ et si $P \xrightarrow{\mu} P'$ alors il existe Q' tel que $Q \xrightarrow{\mu} Q'$ et $P' \equiv Q'$. Aussi on confondra souvent (en particulier dans les exemples) les processus structurellement équivalents.

Les équivalences suivantes concernent plus particulièrement le comportement des protocoles.

5.1.3.2 Testing équivalence

Les *observateurs* sont des processus (et non des protocoles) mis en parallèle avec le protocole testé. Ainsi l'observateur $(\nu k)\bar{p}k.p(x).\,\mathsf{let}\, y = \mathsf{dec}_k(x)[y = s].\bar{a} < a >$, mis un parallèle avec un protocole teste la capacité de ce dernier à recevoir une nouvelle clef k et à envoyer la donnée s chiffrée avec cette nouvelle clef et émet sur le canal a si c'est le cas.

La testing équivalence consiste à évaluer la capacité des protocoles à réussir les tests proposés par les observateurs.

Définition 5.2 *On écrit $P \sqsubseteq_{\sim} Q$ si, pour tout processus O, $P|O \stackrel{w}{\Longrightarrow}$ implique $Q|O \stackrel{w}{\Longrightarrow}$.*

L'équivalence est obtenue par l'intersection du préordre $\sqsubseteq_{\sim}$ et de son inverse : la *testing équivalence*, notée $\simeq$, est définie par $\simeq\, = \,\sqsubseteq_{\sim} \cap \sqsubseteq_{\sim}^{-1}$.

Cette notion d'équivalence ne sera plus utilisée dans la suite de ce chapitre. Nous nous intéresserons uniquement à l'équivalence barbue, basée sur une bisimulation « étape par étape ».

5.1.3.3 Équivalence barbue

On dit qu'un protocole P *se compromet sur* le canal a, ce que l'on note $P \downarrow a$ si $P \xrightarrow{aM} P'$ ou $P \xrightarrow{(\nu\tilde{b})\bar{a}<M>} P'$, pour un certain message M, un ensemble de noms $\tilde{b}$ et un protocole P'. On écrit aussi $P \Downarrow a$ si $P \Longrightarrow P'$ et $P' \downarrow a$, pour un certain protocole P'. Ainsi, $P \Downarrow a$ signifie que le protocole P peut émettre ou recevoir sur a, où a est appelé *barbe*.

Si on interprète le fait de « se compromettre sur un canal » comme « révéler une partie de l'information », alors on dit que deux processus sont en équivalence barbue s'ils révèlent les mêmes informations et s'ils ont le même comportement, quel que soit l'observateur mis en parallèle.

Définition 5.3 (équivalence barbue) *Une relation symétrique $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ est une* bisimulation barbue *si pour tout couple de protocoles P et Q tels que $P\mathcal{S}Q$ on a :*

1. *si P' est un protocole et si $P \xrightarrow{\tau} P'$ alors il existe un protocole Q' tel que $Q \Longrightarrow Q'$ et $P'\mathcal{S}Q'$,*
2. *pour tout nom a, si $P \downarrow a$ alors $Q \Downarrow a$.*

La bisimilarité barbue, *notée $\dot{\cong}$, est la plus grande relation (pour l'inclusion) de bisimulation barbue. Deux protocoles P et Q sont* en équivalence barbue, *noté $P \cong Q$, si pour tout processus R on vérifie $P|R \mathrel{\dot{\cong}} Q|R$.*

Remarque : L'observateur mis en parallèle est un processus et non un protocole car un observateur (ou un intrus) ne peut pas utiliser les constructeurs privés. Si on considère un calcul qui ne comporte pas de symbole de fonction privé (comme le symbole représentant le constructeur des clef privée des agents ou le constructeur des clefs partagées), alors on retrouve ici exactement la notion d'équivalence barbue définie dans [AF01, AG97, BDNP99].

L'équivalence barbue est utilisée pour définir le secret.

5.1.3.4 Secret

Dans [AG97], un protocole P est dit secret pour la variable z si pour tous messages M, M', la relation suivante est vérifiée : $P(M/z) \cong P(M'/z)$.

Une telle propriété ne nous paraît pas pertinente pour modéliser le secret d'un nonce, surtout lorsqu'on permet le chiffrement asymétrique. En effet, considérons le protocole où l'agent A envoie à B sa clef publique et un nonce s chiffré avec sa clef publique :

$$A \to B : \mathsf{pub}(B), \{s\}_{\mathsf{pub}(B)}.$$

Ce protocole peut être modélisé en spi-calcul par $P(s) \stackrel{\text{def}}{=} \bar{p}(< \mathsf{pub}(b), \{s\}_{\mathsf{pub}(b)} >)$ mais les deux protocoles $P(s)$ et $P(s')$ ne sont pas en équivalence barbue. Il suffit en effet de considérer l'observateur $O \stackrel{\text{def}}{=} p(x).[\pi_2(x) = \{s\}_{\pi_1(x)}].\bar{q}(s)$, on obtient alors $P(s) \mid O \Downarrow q$ et $P(s') \mid O \not\Downarrow q$.

Une première solution serait d'interdire aux processus mis en parallèle de contenir les noms s et s' mais alors le protocole qui consiste tout simplement à envoyer le nonce s en clair serait secret, ce qui ne paraît pas raisonnable. Nous distinguons donc deux propriétés de secret suivant que la donnée considérée soit une constante du protocole ou une donnée « fraîche » engendrée au cours d'une session.

Définition 5.4 (secret) *Un protocole P est* secret *pour la variable z représentant une donnée constante si pour tous messages M, M', la relation suivante est vérifiée :* $P(M/z) \cong P(M'/z)$.

Un protocole P est secret *pour la variable z représentant une donnée engendrée au cours d'une session si pour tous messages M, M', la relation suivante est vérifiée :* $(\nu n)P(< M, n > /z) \cong (\nu n)P(< M', n > /z)$.

Nous préciserons à chaque fois quelle est la définition du secret utilisée. On pourra se reporter à [AG97] par exemple pour trouver d'autres notions de secret définies à l'aide d'autres notions d'équivalence.

Exemple 5.5 *Soient s et s' deux noms. On considère les protocoles $P(s), P(s')$, définis dans l'exemple 5.4.*

On peut montrer que $P(s) \dot{\cong} P(s')$. Intuitivement, cela signifie qu'un intrus passif ne peut pas distinguer les deux protocoles $P(s)$ et $P(s')$.

Par contre, $P(s) \not\cong P(s')$. En effet, considérons tout d'abord une attaque "intuitive" qui permet de distinguer les deux protocoles. Un intrus actif peut envoyer à a sa propre clef secrète k', chiffrée avec la clef publique de a. Ensuite l'agent a répond avec le secret s chiffré avec la clef de l'intrus k' au lieu de l'envoyer avec la clef envoyée par b. L'intrus obtient alors le secret. Cela peut se formaliser de la manière suivante : on considère le processus

$$R = p(x).\,\mathsf{let}\, x_1 = <>_1^{-1}(x)\;\mathsf{in}\;(\nu k')\bar{p}(\{k'\}_{\mathsf{pub}(x_1)}).p(y).\,\mathsf{let}\, z = \mathsf{dec}_{k'}(y)\;\mathsf{in}\;[z = s].\bar{k}'(0).\mathbf{0}.$$

Le protocole $P(s)|R$ peut se compromettre sur k' après quelques actions internes, alors que le protocole $P(s')|R$ ne le peut pas.

On montrerait de la même façon que : $(\nu n)P(< s,n >) \dot{\cong} (\nu n)P(< s',n >)$ *et* $(\nu n)P(< s,n >) \not\cong (\nu n)P(< s',n >)$. *Cela montre que P n'est pas secret pour aucune de nos deux définitions.*

5.2 Processus dans un environnement

En général, montrer que deux protocoles sont en équivalence barbue est difficile à cause de la quantification universelle sur l'observateur mis en parallèle des protocoles.

C'est pourquoi nous associons au spi-calcul un modèle de trace avec un environnement explicite de manière à ce que l'équivalence barbue corresponde à une équivalence dans le modèle de trace.

5.2.1 Transitions

Le système de transition décrit ici est identique à celui présenté par M. Boreale.

Les états de notre système de transitions, aussi appelés *configurations*, sont de la forme $\sigma \triangleright P$, où P est un protocole et σ une substitution à valeurs dans les messages qui représente l'environnement. Dans la suite de ce chapitre, on désigne par *substitutions* ou *environnements* uniquement les substitutions et à valeurs dans les messages. Intuitivement, la substitution $\sigma = [t_1/x_1, t_2/x_2 \ldots, t_n/x_n]$ représente un environnement qui possède le message t_1 dans sa première case mémoire, le message t_2 dans sa deuxième case mémoire et ainsi de suite. Toujours intuitivement, un tel environnement pourra tester si $< x_1, x_2 >= x_n$, c'est-à-dire si la paire des contenus de ses deux premières cases mémoires est égal au contenu de sa dernière case mémoire.

Les transitions sont de la forme :

$$\sigma \triangleright P \overset{\mu}{\underset{\delta}{\longmapsto}} \sigma' \triangleright P'.$$

Elles représentent des interactions atomiques entre le protocole P et son environnement σ. Notons que σ n'a pas nécessairement un domaine fini. Comme dans la partie précédente, le symbole μ représente une action de processus (émission, réception ou action interne). Le symbole δ représente une *action d'environnement* qui est complémentaire de la précédente. Elle peut également prendre trois formes (émission, réception ou « pas d'action ») :

$$\delta ::= (\nu\tilde{b})\overline{\eta}(\zeta) \mid \eta(\zeta) \mid -,$$

où ζ et η sont des expressions *publiques* qui n'utilisent donc pas de symboles de fonctions privés.

Les règles d'inférence définissant la relation $\overset{\mu}{\underset{\delta}{\longmapsto}}$ sont décrites à la figure 5.3, qui correspond exactement à la figure 3 de [BDNP99]. Intuitivement, μ représente l'action vue par le protocole tandis que δ représente l'action vue par l'environnement. Ainsi, si $\delta = (\nu\tilde{b})\bar{a} < M >$ alors $\delta = \eta(x)$: lorsque le protocole envoie un message M, l'environnement le stocke dans une nouvelle variable x. Inversement, si $\mu = aM$, alors $\delta = (\nu\tilde{b})\overline{\eta}(\zeta)$: lorsque l'environnement forme un message à partir de manipulations des contenus de ses cases mémoires ($\widehat{\zeta\sigma} = M$) et

On suppose que $n(\eta) \subseteq dom(\sigma)$ et que les noms de $\tilde{b}$ n'apparaissent pas dans σ ni P.

$$\text{E}-\text{OUT}\ \frac{P \xrightarrow{(\nu\tilde{b})\bar{a}<M>} P' \quad \widehat{\eta\sigma} = a}{\sigma \triangleright P \underset{\eta(x)}{\overset{(\nu\tilde{b})\bar{a}<M>}{\longmapsto}} \sigma[M/x] \triangleright P'} \qquad \text{E}-\text{TAU}\ \frac{P \xrightarrow{\tau} P'}{\sigma \triangleright P \underset{-}{\overset{\tau}{\mapsto}} \sigma \triangleright P'}$$

$$\text{E}-\text{INP}\ \frac{P \xrightarrow{aM} P' \quad \widehat{\eta\sigma} = a \quad M = \widehat{\zeta\sigma} \quad \tilde{b} \overset{\text{def}}{=} (n(\zeta) - dom(\sigma))}{\sigma \triangleright P \underset{(\nu\tilde{b})\bar{\eta}(\zeta)}{\overset{aM}{\longmapsto}} \sigma[\tilde{b}/\tilde{b}] \triangleright P'}$$

Figure 5.3 - *Règles d'inférence pour le système de transition.*

l'envoie sur un canal qu'il est également capable de former à partir des contenus de ses cases mémoires, alors le protocole reçoit le message M correspondant.

Notation : La relation $\Longmapsto$ représente la clôture transitive et réflexive de $\underset{-}{\overset{\tau}{\longmapsto}}$ et la relation $\underset{u}{\overset{s}{\Longmapsto}}$ est définie inductivement par $\Longmapsto\underset{\delta}{\overset{\mu}{\longmapsto}}\underset{u'}{\overset{s'}{\Longmapsto}}$ si $s = \mu \cdot s'$ et $u = \delta \cdot u'$, et $\underset{-}{\overset{\tau}{\Longmapsto}} = \Longmapsto$.

Soit $\sigma = [M_i/x_i]_{i\in J}$ une substitution. La notation $range(\sigma)$ représente l'ensemble $\{M_i \mid i \in J\}$ et $dom(\sigma)$ représente le domaine de la substitution σ : $\{x_i \mid i \in J\}$. De plus, on écrit parfois $\sigma = [M_1, \ldots, M_n]$ au lieu de $[M_i/x_i]_{1\leqslant i\leqslant n}$ et $M \in \sigma$ au lieu de $M \in range(\sigma)$.

Considérons un exemple de transitions pour un protocole dans un environnement.

Exemple 5.6 *Nous reprenons la définition du protocole $P(s)$, donnée dans l'exemple 5.4. Soit $\sigma \overset{def}{=} [p/x_1, s/x_2]$. Nous pouvons simuler l'attaque décrite dans l'exemple 5.5 à l'aide de notre nouveau système de transition : $P(s)$ envoie les noms a et b que l'environnement σ apprend (première étape). Puis σ envoie sa propre clef secrète k', chiffrée avec la clef publique de a (deuxième étape). Le protocole P peut alors renvoyer le secret, chiffré avec la clef k' de l'intrus (troisième étape).*

Ces trois étapes peuvent être représentées par les trois transitions suivantes.

Étape 1 : $\sigma \triangleright P(s) \underset{x_1(x_3)}{\overset{\bar{p}<a,b>}{\longmapsto}} \sigma[< a, b > /x_3] \triangleright P'(s)$,

où $P'(s) \overset{def}{=} p(x).\,\mathsf{let}\, z = \mathsf{dec}_{\mathsf{prv}(a)}(x)\,\mathsf{in}\,\bar{p}(\{s\}_z).\mathbf{0} \mid p(x).\,\mathsf{let}\, y = <>_1^{-1}(x)\,\mathsf{in}\,(\nu k)\bar{p}(\{k\}_{\mathsf{pub}(y)}).\mathbf{0}$.

Étape 2 : $\sigma[< a, b > /x_3] \triangleright P'(s) \underset{(\nu k)\bar{x}_1(\{k\}_{\mathsf{pub}(<>_1^{-1}(x_3))})}{\overset{p(\{k\}_{\mathsf{pub}(a)})}{\longmapsto}} \sigma[< a, b > /x_3, k/k] \triangleright P''(s)$

où $P''(s) \overset{def}{=} \mathsf{let}\, z = \mathsf{dec}_{\mathsf{prv}(a)}(\{k\}_{\mathsf{pub}(a)})\,\mathsf{in}\,\bar{p}(\{s\}_z).\mathbf{0} \mid p(x).\,\mathsf{let}\, y = <>_1^{-1}(x)\,\mathsf{in}\,(\nu k)\bar{p}(\{k\}_{\mathsf{pub}(y)}).\mathbf{0}$.

Étape 3 : $\sigma[< a, b > /x_3, k/k] \triangleright P''(s) \underset{x_1(x_4)}{\overset{\bar{p}<\{s\}_k>}{\longmapsto}} \sigma[< a, b > /x_3, k/k, \{s\}_k/x_4] \triangleright P'''(s)$

où $P'''(s) \overset{def}{=} \mathbf{0} \mid p(x).\,\mathsf{let}\, y = <>_1^{-1}(x)\,\mathsf{in}\,(\nu k)\bar{p}(\{k\}_{\mathsf{pub}(y)}).\mathbf{0}$.

5.2.2 Équivalence de traces

Nous définissons dans cette partie une équivalence entre configurations. Pour cela, nous définissons tout d'abord une équivalence entre environnements, que nous allons caractériser afin de montrer la décidabilité de cette équivalence entre environnement. Les définitions d'équivalence entre environnements et entre configurations sont celles de [BDNP99]. Par contre la caractérisation de l'équivalence entre environnements présentée ici diffère complètement de celle donnée par M. Boreale *et al.* Elle est à la base de notre généralisation de leurs résultats.

5.2.2.1 Équivalence entre environnements

Deux environnements sont équivalents si aucune formule du calcul ne peut les distinguer.

Définition 5.5 (équivalence entre environnements [BDNP99]) *Deux substitutions σ et σ', à valeurs dans les messages, sont* équivalentes*, ce que l'on note $\sigma \sim \sigma'$, si $dom(\sigma) = dom(\sigma')$ et pour toute formule ϕ telle que $fn(\phi) \subseteq dom(\sigma)$, la relation $\sigma \models \phi$ est vérifiée si et seulement si $\sigma' \models \phi$ l'est aussi.*

Étant données deux substitutions, il est difficile de décider de manière algorithmique si elles sont équivalents à cause de la quantification universelle sur les formules. En particulier, l'équivalence $\sim$ n'est pas décidable dans [AF01] car ils autorisent des théories équationnelles arbitraires et pour certaines théories équationnelles, même l'égalité entre termes est indécidable. Dans [BDNP99], une caractérisation de $\sim$ à l'aide d'une autre équivalence $\sim'$ est introduite. Ici, nous ne pouvons plus utiliser l'équivalence $\sim'$ car notre modèle est devenu trop général. Nous allons donc caractériser $\sim$ par une approche à la fois plus simple et plus générale.

Tout d'abord, nous allons définir la *connaissance* d'un environnement, c'est-à-dire l'ensemble des messages qu'il peut construire.

Définition 5.6 (connaissance) *Soit W un ensemble de messages. La connaissance engendrée par W, notée $kn(W)$, est le plus petit ensemble de messages tel que :*

1. $W \subseteq kn(W)$
2. $\forall f \in \mathcal{VF}$, *si* $M_1, \ldots, M_n \in kn(W)$, *alors* $f(M_1, \ldots, M_n) \in kn(W)$,
3. $\forall f \in \mathcal{IF}$, $\forall i \leqslant arity(f)$, *si* $f(M_1, \ldots, M_n) \in kn(W)$, *alors* $M_i \in kn(W)$,
4. *si* $\{M\}_k \in kn(W)$ *et* $k^{-1} \in kn(W)$, *alors* $M \in kn(W)$.

Remarque : L'ensemble $kn(W)$ correspond à l'ensemble $\mathsf{fake}(W)$ défini au chapitre 4, mais la définition est maintenant étendue aux nouvelles primitives cryptographiques ajoutées.

Si σ est une substitution, on définit la connaissance $kn(\sigma)$ de la substitution (ou de l'environnement) par $kn(\sigma) = kn(range(\sigma))$. L'ensemble $kn(W)$ est un ensemble infini dès que W n'est pas vide mais il est reconnaissable par un automate d'arbre.

Proposition 5.1 *Soit W un ensemble de messages, tel que W soit effectivement reconnaissable par un automate d'arbre. L'ensemble $kn(W)$ est effectivement reconnaissable par un automate d'arbre.*

Preuve. Nous reprenons les notions de *contraintes ensemblistes* et *contraintes ensemblistes définies* utilisées dans [CP97]. W. Charatonik et A. Podelski ont en particulier montré qu'un ensemble W est reconnaissable par un automate d'arbre si et seulement s'il existe un ensemble de contraintes ensemblistes définies $\mathcal{C}$ tel que S est une variable d'ensemble de $\mathcal{C}$ et $W = \theta(S)$ où θ est la solution minimale de $\mathcal{C}$.

On associe donc à W un ensemble de contraintes ensemblistes $\mathcal{C}$ qui vérifie la propriété décrite ci-dessus. On construit alors un ensemble de contraintes ensemblistes $\mathcal{C}'$ tel que R est une variable d'ensemble de $\mathcal{C}'$ et $W = \theta'(R)$ où θ' est la solution minimale de $\mathcal{C}'$. La caractérisation des ensembles réguliers donnée par W. Charatonik et A. Podelski permet de conclure que l'ensemble $kn(W)$ est reconnaissable par un automate d'arbre. Il reste à construire un tel $\mathcal{C}'$.

On suppose que Msg est une variable d'ensemble qui n'apparaît pas dans $\mathcal{C}$. On pose N_0 l'ensemble des noms apparaissant dans W. L'ensemble N_0 est fini. On considère $\mathcal{C}'$ défini par l'union de $\mathcal{C}$ et des contraintes décrites à la figure 5.4.

La vérification qu'un tel ensemble $\mathcal{C}'$ convient, c'est-à-dire que la solution minimale θ' de $\mathcal{C}'$ vérifie $W = \theta'(R)$, est laissée au lecteur.

$$\begin{array}{ll}
N_0 \subseteq \mathsf{Msg} & f'(\mathsf{Msg}, \ldots, \mathsf{Msg}) \subseteq \mathsf{Msg}, \quad \forall f' \in \mathcal{F} \\[2ex]
S \subseteq R & \{\}_1^{-1}\left(\{\mathsf{Msg}\}_{R \cap f(\mathsf{Msg}, \ldots, \mathsf{Msg})}\right) \subseteq R \\[2ex]
\{\}_1^{-1}\left(\{\mathsf{Msg}\}_{\mathsf{pub}(\mathsf{prv}^{-1}(R))}\right) \subseteq R & \{\}_1^{-1}\left(\{\mathsf{Msg}\}_{\mathsf{prv}(\mathsf{pub}^{-1}(R))}\right) \subseteq R \\[2ex]
h(R, \ldots, R) \subseteq R & g_i^{-1}(R) \subseteq R
\end{array}$$

$$\forall f \in \mathcal{F} \quad f \neq \mathsf{pub}, f \neq \mathsf{prv} \quad \forall h \in \mathcal{VF}, \quad \forall g \in \mathcal{IF}\ \forall i \leqslant arity(g)$$

FIGURE 5.4 - *Contraintes ensemblistes définissant* $kn(W)$.

□

Remarque : Ce résultat est une simple extension des résultats présentés dans [GL00] et [Mon99]. Du point de vue complexité, l'automate reconnaissant $kn(W)$ peut être calculer à partir de l'automate reconnaissant W en temps *a priori* exponentiel.

Nous allons maintenant relier la *connaissance* de l'environnement à l'ensemble des termes qu'il peut construire et donc en particulier envoyer sur le réseau.

Proposition 5.2 *Soit σ une substitution et t un message. Le terme t appartient à $kn(\sigma)$ si et seulement s'il existe une expression publique η telle que $fn(\eta) \subseteq dom(\sigma)$ et $\widehat{\zeta\sigma} = t$.*

Preuve. Supposons $t \in kn(\sigma)$. On définit $P = \{t \mid \exists \eta,\ fn(\eta) \subseteq dom(\sigma) \text{ et } \widehat{\zeta\sigma} = t\}$. Si $t_i/x_i \in \sigma$, alors $t_i \in P$: il suffit de choisir $\eta = x_i$. Donc $range(\sigma) \subseteq P$. De plus, on peut facilement vérifier que P est clos par les règles de construction de la connaissance, donc par définition de $kn(\sigma)$, on déduit $kn(\sigma) \subseteq P$.

Inversement, on montre par induction sur η que s'il existe une expression publique η telle que $fn(\eta) \subseteq dom(\sigma)$ et $\widehat{\zeta\sigma} = t$ alors $t \in kn(\sigma)$. □

5.2.2.2 Caractérisation de l'équivalence

Nous pouvons maintenant donner une définition alternative de l'équivalence entre environnements. Intuitivement, on va associer à chaque message M, un message $\overline{M}^{\sigma}$ dans lequel les parties non décomposables de M ont été remplacées par une variable. Deux parties non décomposables mais identiques sont remplacées par une même variable pour refléter la capacité de l'intrus à tester l'égalité des messages. Deux substitutions seront alors équivalentes si les messages transformés sont égaux.

Pour définir cette transformation formellement, nous commençons par ajouter un symbole de fonction inv d'arité un servant à signaler qu'un message est l'inverse d'un autre message : le terme $\mathsf{inv}(t)$ représente le message qui a pour inverse t. Aussi, nous définissons les *messages étendus* comme les termes définis sur $\mathcal{N} \cup \mathcal{F} \cup \{\mathsf{inv}\}$. Ensuite, nous définissons l'ensemble des *chemins* d'un message étendu comme cela est fait habituellement pour les termes, excepté que l'on ne tient pas compte du symbole inv.

Définition 5.7 (chemin) *L'ensemble des* chemins *d'un message étendu t, noté $path(t)$, est défini inductivement par :*

$$\begin{array}{rcll} path(a) & = & \{1\} & \textit{si } a \textit{ est un nom}, \\ path(f(t_1, \ldots, t_n)) & = & \{\epsilon\} \cup 1.path(t_1) \cup \ldots \cup n.path(t_n) & \textit{si } f \in \mathcal{F} \\ path(\mathsf{inv}t) & = & path(t) & \end{array}$$

Soit $\sigma = [M_i/x_i]_{i \in J}$ une substitution. On définit l'ensemble des chemins de la substitution par $path(\sigma) = \cup_{i \in J} i.path(M_i)$.

Soit t un message étendu, le sous terme de t associé au chemin p de $path(t)$ est défini inductivement par :

$$\begin{array}{rcll} t|_{\epsilon} & = & t & \\ f(t_1, \ldots, t_n)|_{i.p} & = & t_i|_p & \textit{si } f \in \mathcal{F} \\ (t)^{-1}|_p & = & t|_p & . \end{array}$$

Notation : Soit t un message étendu. L'ensemble $leaves(t)$ est l'ensemble des chemins maximaux (pour la relation d'ordre $p \leqslant p'$ si $p' = p{\cdot}p''$) de $path(t)$. La suite des symboles rencontrés le long d'un chemin p est notée $s_t(p)$. La suite $s_t(p)$ est définie inductivement par $s_t(\epsilon) = \epsilon$, $s_{f(t_1,\ldots,t_n)}(i.p') = f.s_{t_i}(p')$ et $s_{(t)^{-1}}(p) = s_t$. La fonction s est étendue aux substitutions de la manière suivante : soit $\sigma = [M_i/x_i]_{i \in J}$ une substitution, $s_\sigma(i.p) = s_{t_i}(p)$.

Nous pouvons maintenant décrire notre transformation qui associe à tout message m un message étendu correspondant à l'information maximale que l'intrus peut obtenir de m.

Définition 5.8 *Soit σ une substitution. L'ensemble Var_σ désigne l'ensemble des variables X_t telles que t est un sous terme d'un message M de σ. La transformation*

$\overline{\cdot}^{\sigma} : \mathcal{T}(\mathcal{F}) \to \mathcal{T}(\mathcal{F} \cup \{\mathsf{inv}\}\, Var_{\sigma})$ *est définie inductivement de la manière suivante (on écrit* $\overline{t}$ *au lieu de* $\overline{t}^{\sigma}$ *lorsqu'il n'y a pas d'ambiguïté sur* σ*) :*

$$\overline{a} = X_a \text{ si } a \text{ est un nom} \tag{5.1}$$

$$\overline{f(t_1,\ldots,t_n)} = f(\overline{t_1},\ldots,\overline{t_n}) \text{ si } f \in \mathcal{IF} \tag{5.2}$$

$$\begin{array}{c}\overline{g(t_1,\ldots,t_n)}\\ g \notin \mathcal{IF},\ g \neq \{\_\}_\_\end{array} = \begin{cases} g(\overline{t_1},\ldots,\overline{t_n}) & \text{si } g \in \mathcal{VF} \text{ et } t_1,\ldots,t_n \in kn(\sigma) \\ X_{g(t_1,\ldots,t_n)} & \text{sinon} \end{cases} \tag{5.3}$$

$$\overline{\{t_1\}_{t_2}} = \begin{cases} \{\overline{t_1}\}_{\mathsf{inv}(\overline{t})} & \text{si } t_2 = t^{-1} \text{ avec } t \in kn(\sigma) \\ \{\overline{t_1}\}_{\overline{t_2}} & \text{si } t_1, t_2 \in kn(\sigma) \\ X_{\{t_1\}_{t_2}} & \text{sinon} \end{cases} \tag{5.4}$$

$$\overline{t} = X_t \text{ sinon} \tag{5.5}$$

La transformation $\overline{\cdot}^{\sigma}$ *est étendue aux substitutions* $\sigma' = [M_i/x_i]_{i\in J}$ *par* $\overline{\sigma'}^{\sigma} = [\overline{M_i}^{\sigma}/x_i]_{i\in J}$*.*

Remarque : On montre facilement que si p est un chemin non vide de $\overline{\sigma'}^{\sigma}$, alors $\sigma|_p \in kn(\sigma)$.

Exemple 5.7 *Étant donnée la substitution* $\sigma_1 = [\{s\}_k/x_1, \{s\}_{\{s\}_k}/x_2, s/x_3]$*, on calcule :* $\overline{\sigma_1}^{\sigma_1} = [X_{\{s\}_k}/x_1, \{X_s\}_{X_{\{s\}_k}}/x_2, X_s/x_3]$*.*

De même si $\sigma_2 = [\{s\}_{\mathsf{prv}(a)}/x_1, \mathsf{pub}(a)/x_2]$ *alors* $\overline{\sigma_2}^{\sigma_2} = [\{X_s\}_{\mathsf{inv}(\mathsf{pub}(X_a))}/x_1, \mathsf{pub}(X_a)/x_2]$*. Intuitivement, l'environnement peut détecter que le premier message est chiffré par une clef dont il connaît l'inverse* ($\mathsf{pub}(a)$) *même s'il ne connaît pas la clef elle-même.*

Deux environnements sont équivalents si les transformées sont égales, au renommage des variables près.

Définition 5.9 (équivalence $\sim'$ entre environnements) *Soit* $\mathcal{P}$ *l'ensemble de chemins de* $\overline{\sigma}^{\sigma}$*. Pour chaque prédicat* V_i *du calcul, on définit l'"ensemble :*

$$A^i_{\sigma} \stackrel{def}{=} \{\zeta \in \mathcal{T}(\mathcal{VF} \cup \{x_p | p \in \mathcal{P}\}) \mid [\![V_i(\zeta[\sigma|_p/x_p]_{p\in\mathcal{P}})]\!] = tt\}.$$

Soient σ *et* σ' *deux substitutions. On dit que* σ *et* σ' *sont équivalentes, ce que l'on note* $\sigma \sim' \sigma'$*, si et seulement si :*

1. *il existe un renommage (c'est-à-dire une fonction bijective)* $\theta : Var_{\sigma} \to Var_{\sigma'}$ *tel que* $\overline{\sigma}^{\sigma}\theta = \overline{\sigma'}^{\sigma'}$ *et* θ préserve les types de clefs *(symétrique ou asymétrique) : si* $\theta(X_t) = X_{t'}$ *alors* t *est sa propre inverse si et seulement si* t' *est également sa propre inverse.*
2. *pour tout entier* i *tel que* $1 \leqslant i \leqslant n$, $A^i_{\sigma} = A^i_{\sigma'}$*.*

L'ensemble A^i_{σ} correspond à l'ensemble des expressions publiques dont l'évaluation par l'intrus est reconnue par le prédicat V_i, c'est-à-dire l'ensemble des expressions publiques dont l'intrus peut détecter qu'elles correspondent à un type particulier de messages.

Exemple 5.8 *On considère à nouveau la substitution* $\sigma_1 = [\{s\}_k/x_1, \{s\}_{\{s\}_k}/x_2, s/x_3]$ *introduite dans l'exemple 5.7. Soit* $\sigma_2 = [\{s\}_k/x_1, \{s\}_{\{s\}_k}/x_1, s'/x_1]$*. Alors :*

$$\begin{aligned}\overline{\sigma_1}^{\sigma_1} &= [X_{\{s\}_k}/x_1, \{X_s\}_{X_{\{s\}_k}}/x_2, X_s/x_3] \\ \overline{\sigma_2}^{\sigma_2} &= [X_{\{s\}_k}/x_1, \{X_s\}_{X_{\{s\}_k}}/x_2, X_{s'}/x_3].\end{aligned}$$

On en déduit que : $\sigma_1 \not\sim' \sigma_2$*. Intuitivement, l'environnement* σ_1 *peut détecter que le contenu de sa troisième variable, chiffré avec le contenu de la première est égal au contenu de la troisième, ce qui n'est pas le cas pour l'environnement* σ_2*. Ainsi, la formule* $\phi = [\{x_3\}_{x_1} = x_2]$ *distingue les deux environnements :* $\sigma_1 \models \phi$ *et* $\sigma_2 \not\models \phi$*. Par conséquent,* $\sigma_1 \not\sim \sigma_2$*.*

Comme annoncé, les deux notions d'équivalence pour les environnements sont identiques.

Théorème 5.1 (coïncidence de $\sim$ et $\sim'$) *Soient* σ *et* σ' *deux substitutions ayant le même domaine :* $\sigma \sim \sigma'$ *si et seulement si* $\sigma \sim' \sigma'$*.*

L'inclusion $\sim \subseteq \sim'$ repose en particulier sur le fait qu'on peut associer à tout chemin p de $\overline{\sigma}^{\sigma}$ une formule et une expression publique qui le caractérisent au sens de la proposition suivante.

Proposition 5.3 *Soit* σ *une substitution. Si* p *est un chemin de* $\overline{\sigma}^{\sigma}$ *alors il existe une formule* ϕ_p *et une expression publique* ζ_p *telles que* $dom(\phi_p), dom(\zeta_p) \subseteq dom(\sigma)$*,* $\widehat{\zeta_p\sigma} = \sigma|_p$ *et* $\sigma \models \phi_p$*.*

De plus, ϕ_p *et* ζ_p caractérisent p *au sens suivant : pour toute substitution* σ'*, si* $dom(\sigma) = dom(\sigma')$ *alors :*

$$\sigma' \models \phi_p \text{ et } \widehat{\zeta_p\sigma'} \neq \bot \Rightarrow \widehat{\zeta_p\sigma'} = \sigma'|_p,\ s_\sigma(p) = s_{\sigma'}(p) \text{ et } p \text{ est un chemin de } \overline{\sigma'}^{\sigma'}.$$

Preuve. Soit $\sigma = [t_i/x_i]_{i\in J}$ une substitution. Nous prouvons cette proposition par induction sur la longueur du chemin p.

Si $p = i$, $\sigma|_i = t_i$, on pose $\zeta = x_i$ et $\phi = tt$. Alors ζ et ϕ caractérisent i.

Supposons maintenant la propriété vraie pour tous les chemins de longueurs plus petites que n. Considérons un chemin p' de $\overline{\sigma}^{\sigma}$, de longueur $n+1$. Alors $p' = p \cdot i$ avec p chemin de $\overline{\sigma}^{\sigma}$, de longueur n. Par hypothèse d'induction, il existe ϕ, ζ tels que $\sigma \models \phi$, $\widehat{\zeta\sigma} = \sigma|_p$ et ϕ et ζ caractérisent p. Différents cas sont à étudier suivant la forme de $\sigma|_p$.

1. Si $\sigma|_p = f(t_1, \ldots, t_n)$ et $f \in \mathcal{IF}$ alors $t|_{p'} = t_i$. On pose $\zeta' = f_i^{-1}(\zeta)$ et $\phi' = \phi$. Montrons que η' et ϕ' conviennent. Soit σ' une substitution telle que $\phi' \models \sigma'$ et $\widehat{\zeta'\sigma'} = t \neq \bot$. Alors $\widehat{\zeta\sigma'} = f(t_1, \ldots, t, \ldots, t_n) = \sigma'|_p$ par hypothèse. Donc $t = \sigma'|_{p.i}$, c'est-à-dire $\widehat{\zeta'\sigma'} = \sigma'|_{p'}$. De plus, $s_\sigma(p') = s_{\sigma'}(p')$ et p' est un chemin de $\overline{\sigma'}$, toutes les conditions sont bien vérifiées.
2. Si $\sigma|_p = g(t_1, \ldots, t_n)$ et $g \in \mathcal{VF}$, $t|_{p'} = t_i$. Comme p' est un chemin de $\overline{\sigma}$, nécessairement $\overline{\sigma}|_p = g(\overline{t_1}, \ldots, \overline{t_n})$, donc $t_1, \ldots, t_n \in kn(\sigma)$. En appliquant la proposition 5.2, on déduit que pour tout j tel que $1 \leqslant j \leqslant n$, il existe ζ_j tel que $\widehat{\zeta_j\sigma} = t_j$. Posons $\zeta' = \zeta_i$ et $\phi' = \phi \wedge [\zeta = g(\zeta_1, \ldots, \zeta_n)]$. Alors $\sigma \models \phi'$ et $\widehat{\zeta'\sigma} = \sigma|_{p'}$.

   Considérons alors une substitution σ' telle que $\sigma' \models \phi'$. En particulier, $\widehat{\zeta\sigma'} = t \neq \bot$. Comme $\sigma' \models \phi$, on déduit $\widehat{\zeta\sigma'} = \sigma'|_p$ en appliquant l'hypothèse d'induction. De plus, comme $\sigma' \models [\zeta = g(\zeta_1, \ldots, \zeta_n)]$, on déduit $\sigma'|_p = g(t_1, \ldots, \widehat{\zeta'\sigma'}. \ldots, t_n)$ d'où $\widehat{\zeta'\sigma'} = \sigma'|_{p'}$, $s_\sigma(p') = s_{\sigma'}(p')$ et p' est un chemin de $\overline{\sigma'}$.
3. Si $t|_p = \{t_1\}_{t_2}$ et $t_2 = t^{-1}$ tel que $t \in kn(\sigma)$. D'après la proposition 5.2, il existe ζ_1 tel que $\widehat{\zeta_1\sigma} = t$. Deux cas sont possibles suivant la valeur de i dans $p = p' \cdot i$. Si $t|_{p'} = t_1$, on pose $\zeta' = \mathrm{dec}_{\zeta_1}(\zeta)$, $\phi' = \phi$. Ils satisfont les propriétés demandées.

   Si $t|_{p'} = t$, on pose $\zeta = \zeta_1$ et $\phi' = \phi \wedge [\mathrm{dec}_{\zeta_1}(\zeta) \neq \bot]$, ils satisfont également les propriétés demandées.

4. Si $t|_p = \{t_1\}_{t_2}$, $t|_{p'} = t_1$ ou t_2 et $t_1, t_2 \in kn(\sigma)$. On conclut à l'aide des mêmes arguments que pour le cas 2.
5. Comme p' est un chemin de $\overline{\sigma}^{\sigma}$, il n'y a pas d'autre cas possible. $\square$

Avant de démontrer le théorème 5.1, nous établissons le lemme suivant qui assure que la fonction d'évaluation $\widehat{\cdot}$ et l'opérateur de transformation $\overline{\cdot}$ commutent. Nous étendons la fonction d´évaluation aux messages étendus en ajoutant la règle : $\widehat{\mathsf{inv}(X_t)} = X_{t^{-1}}$ où t^{-1} désigne l'inverse du terme t.

Lemme 5.1 *Soient σ une substitution et ζ une expression telle que $dom(\zeta) \subset dom(\sigma)$. L'égalité suivante est satisfaite :*

$$\overline{\widehat{\zeta\sigma}}^{\sigma} = \widehat{\zeta\overline{\sigma}^{\sigma}}.$$

Corollaire 5.1 *Soient σ et σ' deux substitutions telles que $\sigma \sim' \sigma'$ et θ une fonction de renommage telle que : $\overline{\sigma}^{\sigma}\theta = \overline{\sigma'}^{\sigma'}$. Alors : $\overline{\widehat{\zeta\sigma}}^{\sigma}\theta = \overline{\widehat{\zeta\sigma'}}^{\sigma'}$.*

De plus, si ζ et η sont deux expressions, alors : $\widehat{\zeta\sigma} = \widehat{\eta\sigma}$ si et seulement si $\widehat{\zeta\sigma'} = \widehat{\eta\sigma'}$.

Preuve. Le lemme 5.1 se montre par induction sur ζ. Nous étudions toutes les formes possibles de ζ.

$\zeta = x$ alors $\widehat{\zeta\sigma} = x\sigma$, donc $\overline{\widehat{\zeta\sigma}} = \overline{x\sigma} = x\overline{\sigma} = \widehat{\zeta\overline{\sigma}}$.

$\zeta = f(\zeta_1, \ldots, \zeta_n)$ avec f dans $\mathcal{VF}$. S'il existe i tel que $\widehat{\zeta_i\sigma} = \bot$, alors $\overline{\widehat{\zeta_i\sigma}} = \bot = \widehat{\zeta_i\overline{\sigma}}$ par hypothèse d'induction. On en déduit l'égalité : $\overline{\widehat{\zeta\sigma}}^{\sigma} = \widehat{\zeta\overline{\sigma}}$. Si pour tout i, $\widehat{\zeta_i\sigma} \neq \bot$, alors $\widehat{\zeta\sigma} = f(\widehat{\zeta_1\sigma}, \ldots, \widehat{\zeta_n\sigma})$ et $\widehat{\zeta_i\sigma} \in kn(\sigma)$ d'après la proposition 5.2. D'où $\overline{\widehat{\zeta\sigma}} = f(\overline{\widehat{\zeta_1\sigma}}, \ldots, \overline{\widehat{\zeta_n\sigma}}) = f(\widehat{\zeta_1\overline{\sigma}}, \ldots, \widehat{\zeta_n\overline{\sigma}}) = \widehat{\zeta\overline{\sigma}}$.

$\zeta = \mathsf{dec}_\eta(\zeta')$. Si $\widehat{\eta\sigma} = \bot$ ou $\widehat{\zeta'\sigma} = \bot$, alors par hypothèse d'induction $\widehat{\eta\overline{\sigma}} = \bot$ ou $\widehat{\zeta'\overline{\sigma}} = \bot$, on en déduit que l'égalité $\overline{\widehat{\zeta\sigma}}^{\sigma} = \widehat{\zeta\overline{\sigma}}$ est satisfaite. Supposons maintenant que $\widehat{\eta\sigma} \neq \bot$ et $\widehat{\zeta'\sigma} \neq \bot$. Deux cas sont possibles : $\widehat{\zeta\sigma} = \bot$ ou $\widehat{\zeta\sigma} \neq \bot$. Supposons $\widehat{\zeta\sigma} = \bot$. Ou bien $\widehat{\zeta'\sigma}$ n'est pas un message chiffré, auquel cas $\widehat{\zeta'\overline{\sigma}} = \overline{\widehat{\zeta'\sigma}}$ n'est pas non plus un message chiffré ; ou bien $\widehat{\zeta'\sigma} = \{M\}_k$ et $\widehat{\eta\sigma} = k'$ avec $k' \neq k^{-1}$. Dans ce dernier cas, $\widehat{\zeta'\overline{\sigma}} = \overline{\widehat{\zeta'\sigma}} = X_{\{M\}_k}$ ou $\{\overline{M}\}_{\overline{k}}$. Par hypothèse d'induction, $\widehat{\eta\overline{\sigma}} = \overline{\widehat{\eta\sigma}} = \overline{k'} \neq (\overline{k})^{-1}$, donc $\widehat{\zeta\overline{\sigma}} = \widehat{\mathsf{dec}_\eta(\zeta')\overline{\sigma}} = \bot$ et l'égalité $\overline{\widehat{\zeta\sigma}}^{\sigma} = \widehat{\zeta\overline{\sigma}}$ est vérifiée. Supposons maintenant que $\widehat{\zeta\sigma} \neq \bot$. Alors $\widehat{\zeta'\sigma} = \{M\}_k$ et $\widehat{\eta\sigma} = k'$ avec $k' = k^{-1}$. En utilisant la proposition 5.2, $k' \in \sigma$, $\overline{\{M\}_k} = \{\overline{M}\}_{(\overline{k'})^{-1}}$. Il vient, en appliquant l'hypothèse d'induction : $\widehat{\zeta\overline{\sigma}} = \overline{M} = \overline{\widehat{\zeta\sigma}}$.

$\zeta = g_i^{-1}(\zeta')$ avec $g \in \mathcal{IF}$. Une même étude de cas permet de conclure : $\overline{\widehat{\zeta\sigma}}^{\sigma} = \widehat{\zeta\overline{\sigma}}$.

La preuve de la première partie du corollaire utilise le fait que $\mathsf{inv}(\theta(X_t)) = \theta(X_{t^{-1}})$ ce qui est assuré car σ respecte le type (symétrique ou asymétrique) des clefs.

La deuxième partie du corollaire est une conséquence de la première partie : soient σ et σ' deux substitutions telles que $\sigma \sim' \sigma'$ et supposons que $\widehat{\zeta\sigma} = \widehat{\eta\sigma}$. Alors $\overline{\widehat{\zeta\sigma}}^{\sigma} = \overline{\widehat{\eta\sigma}}^{\sigma}$. Donc $\overline{\widehat{\zeta\sigma}}^{\sigma}\theta = \overline{\widehat{\eta\sigma}}^{\sigma}\theta$. La première partie du corollaire permet de conclure que : $\overline{\widehat{\zeta\sigma'}}^{\sigma'} = \overline{\widehat{\eta\sigma'}}^{\sigma'}$. Par construction de $\overline{\cdot}^{\sigma'}$, $\overline{t_1}^{\sigma'} = \overline{t_2}^{\sigma'}$ si et seulement si $t_1 = t_2$. On en déduit $\widehat{\zeta\sigma'} = \widehat{\eta\sigma'}$. $\square$

Nous pouvons maintenant prouver le théorème 5.1 qui assure l'équivalence entre les deux notions d'équivalence entre environnements.

Théorème (théorème 5.1) *Soient σ et σ' deux substitutions ayant le même domaine. $\sigma \sim \sigma'$ si et seulement si : $\sigma \sim' \sigma'$.*

Preuve. Soient $\sigma = [M_i/x_i]_{i\in J}$ et $\sigma' = [M'_i/x_i]_{i\in J}$ deux substitutions.
$\boxed{\Rightarrow}$ Supposons $\sigma \not\sim' \sigma'$. Montrons qu'il existe une formule ϕ telle que $\sigma \models \phi$ et $\sigma' \not\models \phi$ (ou $\sigma' \models \phi$ et $\sigma \not\models \phi$).

Supposons tout d'abord qu'il existe i tel que $A^i_\sigma \neq A^i_{\sigma'}$, c'est-à-dire qu'il existe $\zeta \in A^i_\sigma$ tel que $\zeta \notin A^i_{\sigma'}$ (ou symétriquement), d'où $[\![P_i(\widehat{\zeta[\overline{\sigma}|_p/x_p]})]\!] = tt$ et $[\![P_i(\widehat{\zeta[\overline{\sigma'}|_p/x_p]})]\!] = ff$. En appliquant la proposition 5.3, on construit pour chaque chemin p de $\overline{\sigma}^\sigma$, une formule ϕ_p et une expression ζ_p qui caractérisent p. Posons $\zeta' = \zeta[\zeta_p/x_p]$ et $\phi = \bigwedge_{p\in\mathcal{P}} \phi_p \wedge P_i(\zeta)$ où $\mathcal{P}$ désigne l'ensemble des chemins de $\overline{\sigma}^\sigma$. Alors $\sigma \models \phi$ et $\sigma' \not\models \phi$ car $\zeta \notin A^i_{\sigma'}$.

On considère maintenant l'ensemble des positions de $\overline{\sigma}$ et $\overline{\sigma'}$ qui aboutissent à des variables. Ainsi, on définit $I_v = \{p \mid \overline{\sigma}|_p \in Var_\sigma$ et $\overline{\sigma'}|_p \in Var_{\sigma'}\}$, l'ensemble des chemins p tels que $\overline{\sigma}|_p$ et $\overline{\sigma'}|_p$ soient des variables. Pour toute variable $X \in Var_\sigma$, on définit également l'ensemble $P_X = \{p \in I_v \mid \overline{\sigma}|_p = X\}$ et pour toute variable $X \in Var_{\sigma'}$, l'ensemble $P'_X = \{p \in I_v \mid \overline{\sigma'}|_p = X\}$. Alors les P_X et les P'_X définissent une partition de I_v :

$$I_v = \biguplus_{X\in Var_\sigma} P_X = \biguplus_{X\in Var_{\sigma'}} P'_X.$$

Supposons par l'absurde que $\biguplus_{X\in Var_\sigma} P_X$ et $\biguplus_{X\in Var_{\sigma'}} P'_X$ ne définissent pas la même partition de I_v, c'est-à-dire qu'il existe $X \in Var_\sigma$ tel que , pour toute variable $X' \in Var_{\sigma'}$, on ait $P_X \neq P_{X'}$. Comme σ et σ' jouent des rôles symétriques, on peut supposer alors que pour toute variable $X' \in Var_{\sigma'}$, $P_X \not\subseteq P_{X'}$.

Posons $\{p_1, \ldots, p_n\} = P_X$. Comme p_i est un chemin de $\overline{\sigma}$, on choisit ϕ_i et ζ_i caractérisant p_i d'après la proposition 5.3 : $\sigma \models \phi_i$ et $\widehat{\zeta_i\sigma} = \sigma|_{p_i}$. On définit :

$$\phi \stackrel{\text{def}}{=} \bigwedge_{1\leqslant i\leqslant n} \phi_i \wedge [\zeta_1 = \zeta_2][\zeta_2 = \zeta_3] \ldots [\zeta_{n-1} = \zeta_n].$$

Pour un certain sous-terme t de σ, la variable X est égale à X_t et pour tout entier i tel que $1 \leqslant i \leqslant n$, on vérifie $\sigma|_{p_i} = t$, d'où $\sigma \models \phi$. Inversement, supposons que $\sigma' \models \phi$. Alors il existe un terme t' tel que pour tout entier i tel que $\forall 1 \leqslant i \leqslant n$, on a $\widehat{\zeta_i\sigma'} = t'$. Par construction de ϕ_i et ζ_i et comme $\phi_i \models \sigma'$ et $\widehat{\zeta_i\sigma'} \neq \bot$, on déduit $\widehat{\zeta_i\sigma'} = \sigma'|_{p_i} = t'$. Par définition des P_X et P'_X, le fait que $p_i \in P_X$ induit que $\overline{\sigma'}|_{p_i} \in Var_{\sigma'}$, donc $\overline{\sigma'}|_{p_i} = X_{t'}$ pour un certain terme t'. On en déduit : $P_X \subset P'_{X_{t'}}$, contradiction.

Supposons maintenant que $\biguplus_{X\in Var_\sigma} P_X$ et $\biguplus_{X\in Var_{\sigma'}} P'_X$ définissent la même partition mais que cette partition ne respecte pas le type des clefs, c'est à dire qu'il existe deux variables $X_t, X_{t'}$ telles que $P_{X_t} = P_{X_{t'}}$ mais que t est symétrique alors que t' ne l'est pas (ou le contraire). Posons alors $\{p_1, \ldots, p_n\} = P_{X_t}$ choisissons ϕ_1 et ζ_1 caractérisant p_1 d'après la proposition 5.3 : $\sigma \models \phi_1$ et $\widehat{\zeta_1\sigma} = \sigma|_{p_1}$. On définit $\phi = \phi_1 \wedge [\mathsf{dec}_{\zeta_1}(\{\zeta_1\}_{\zeta_1}) = \zeta_1]$. On vérifie alors que : $\sigma \models \phi$ et $\sigma' \not\models \phi$.

On peut donc supposer que $\biguplus_{X \in Var_\sigma} P_X$ et $\biguplus_{X \in Var_{\sigma'}} P'_X$ définissent la même partition I_v et que cette partition respecte le type des clefs et la satisfaction des prédicats. Soit θ une substitution telle que $\forall X \in Var_\sigma, \forall X' \in Var_{\sigma'}$, $P_X = P'_{X'} \Rightarrow X\theta = X'$. Par construction, si $\overline{\sigma}|_p$ et $\overline{\sigma'}|_p$ sont tous les deux des variables alors $\overline{\sigma}|_p\theta = \overline{\sigma'}|_p$. Comme $\sigma \not\sim' \sigma'$, $\overline{\sigma}^\sigma\theta \neq \overline{\sigma'}^{\sigma'}$, il existe i et p tels que $\overline{M_i|_p}^\sigma\theta \neq \overline{M'_i|_p}^{\sigma'}$ et tels que les symboles de tête de $\overline{M_i|_p}^\sigma\theta$ et $\overline{M'_i|_p}^{\sigma'}$ soient distincts. On vient juste de voir que $\overline{M_i|_p}^\sigma\theta$ et $\overline{M'_i|_p}^{\sigma'}$ ne peuvent pas être tous les deux des variables (sinon il y aurait égalité). Supposons (par exemple) que $\overline{M_i|_p}^\sigma\theta$ n'est pas une variable. On considère alors trois cas.

1. Si $\overline{M_i|_p} = f(\overline{t_1}, \ldots, \overline{t_n})$ avec $f \in \mathcal{VF}$ (et éventuellement $f = \{\_\}\_$). Comme $p_j = i.p.j$ (avec $1 \leqslant j \leqslant n$) est également un chemin de $\overline{\sigma}$, on peut définir ϕ_j et ζ_j caractérisant p_j, à l'aide de la proposition 5.3 : $t_j = \widehat{\zeta_j\sigma}$ et $\sigma \models \phi_j$. Soient ϕ, ζ caractérisant $p' = i.p$. On pose
$$\phi' = \phi \wedge \bigwedge_{1 \leqslant j \leqslant n} \phi_i \wedge [\zeta = f(\zeta_1, \ldots, \zeta_n)].$$
Alors $\sigma \models \phi$. Montrons que : $\sigma' \not\models \phi$.
Supposons par l'absurde que $\sigma' \models \phi$. Alors $\sigma'|_{p'} = M'_i|_p = \widehat{\zeta\sigma'} = f(\widehat{\zeta_1\sigma'}, \ldots, \widehat{\zeta_n\sigma'})$, donc $\widehat{\zeta_1\sigma'}, \ldots, \widehat{\zeta_n\sigma'} \in kn(\sigma')$, d'où dans tous le cas : $\overline{\sigma'|_{p'}}^{\sigma'} = \overline{M'_i|_p}^{\sigma'} = f(\overline{\cdot}, \ldots, \overline{\cdot})$, c'est-à-dire $\overline{M_i|_p\theta}^\sigma\theta$ et $\overline{M'_i|_p}^{\sigma'}$ ont le même symbole de tête, contradiction.
2. Si $\overline{M_i|_p} = \{\overline{t_1}\}_{\mathsf{inv}(\overline{t})}$ et $M_i|_p = \{t_1\}_{t_2}$ tel que $t_2 = t^{-1}$. En appliquant une fois de plus la proposition 5.3, on définit $\zeta, \phi, \zeta_1, \phi_1, \zeta_2, \phi_2$ tels que $\widehat{\zeta\sigma} = M_i|_p$, $\widehat{\zeta_1\sigma} = t_1$, $\widehat{\zeta_2\sigma} = t$ et $\sigma \models \phi, \phi_1, \phi_2$. Posons
$$\phi_3 \stackrel{\text{def}}{=} \phi \wedge \phi_1 \wedge \phi_2 \wedge [\zeta_1 = \mathsf{dec}_{\zeta_2}(\zeta)] \wedge \neg[\zeta = \{\zeta_1\}_{\zeta_2}].$$
On obtient alors $\sigma \models \phi$ et $\sigma' \not\models \phi$.
3. Si $\overline{M_i|_p} = f(\overline{t_1}, \ldots, \overline{t_n})$ avec $f \in \mathcal{IF}$. Comme $p_j = i.p.j$ (avec $1 \leqslant j \leqslant n$) est également un chemin de $\overline{\sigma}$, d'après la proposition 5.3, on peut définir ϕ_j et ζ_j caractérisant p_j : $t_j = \widehat{\zeta_j\sigma}$ et $\sigma \models \phi_j$. Soient ϕ et ζ caractérisant $p' = i.p$. On pose
$$\phi' = \phi \wedge \bigwedge_{1 \leqslant j \leqslant n} \phi_i \wedge [f_1^{-1}(\zeta) = \zeta_1] \wedge \ldots \wedge [f_n^{-1}(\zeta) = \zeta_n].$$
On obtient alors $\sigma \models \phi'$ et $\sigma' \not\models \phi'$.

$\boxed{\Leftarrow}$ Étudions la réciproque. On montre par induction sur la construction de la formule ϕ que, étant données deux substitutions σ et σ', elles vérifient $\sigma \sim' \sigma \Rightarrow [\sigma \models \phi \Leftrightarrow \sigma' \models \phi]$.

Soient σ et σ' deux substitutions telles que $\sigma \sim' \sigma$ et θ une fonction de renommage telle que $\overline{\sigma}^\sigma\theta = \overline{\sigma'}^{\sigma'}$.

$\phi = tt$. La propriété est vérifiée.

$\phi = P_i(\zeta)$. Supposons $\sigma \models \phi$. Alors $\widehat{\zeta\sigma} = t$ et la formule $P_i(t)$ est vraie. Montrons par induction sur ζ que :

$$\textit{si } \widehat{\zeta\sigma} \neq \perp \textit{ alors } \exists\zeta' \textit{ t. q. } \widehat{\zeta\sigma} = \zeta'\widehat{[\sigma_p/x_p]_{p\in\mathcal{P}}} \textit{ avec } \zeta' \in \mathcal{T}(\mathcal{VF} \cup \{x_p | p \in \mathcal{P}\}),$$

où $\mathcal{P}$ désigne l'ensemble des chemins de $\overline{\sigma}^\sigma$. Si $\zeta = \mathsf{dec}_\eta(\zeta'')$, deux cas sont possibles. Ou bien $\zeta'' = \{\eta_1\}_{\eta_2}$ et $\widehat{\zeta\sigma} = \widehat{\eta_1\sigma}$. Alors, par hypothèse d'induction, il existe ζ' tel que $\widehat{\eta_1\sigma} = \zeta'\widehat{[\sigma_p/x_p]}_{p\in\mathcal{P}}$. Ou bien $\zeta'' = x_i \in dom(\sigma)$ et $\widehat{\zeta\sigma} = \sigma|_{i\cdot 1}$. Alors $i \cdot 1$ est un chemin de $\overline{\sigma}$, donc $\zeta' = x_{i\cdot 1}$ convient. Les autres cas pour la forme de ζ sont immédiats.

On en déduit qu'il existe ζ' tel que $\zeta'\widehat{[\sigma_p/x_p]}_{p\in\mathcal{P}} = t$ et $\zeta' \in \mathcal{T}(\mathcal{VF} \cup \{x_p | p \in \mathcal{P}\})$ donc $\zeta' \in A^i_\sigma$. Comme $\sigma \sim' \sigma$, on vérifie en particulier que $A^i_\sigma = A^i_{\sigma'}$ et donc $\zeta' \in A^i_{\sigma'}$, d'où $\zeta'\widehat{[\sigma'_p/x_p]}_{p\in\mathcal{P}'} = t'$ avec $t' \in P_i$ et $\mathcal{P}'$ l'ensemble des chemins de $\overline{\sigma'}^{\sigma'}$. On en déduit que $\overline{t}^\sigma\theta = \overline{t'}^{\sigma'}$, c'est-à-dire $\overline{\widehat{\zeta\sigma}}^\sigma\theta = \overline{\zeta'\widehat{[\sigma_p/x_p]}_{p\in\mathcal{P}'}}^{\sigma'}$. En appliquant le corollaire 5.1, on déduit $\widehat{\zeta\sigma'} = \zeta'\widehat{[\sigma_p/x_p]}_{p\in\mathcal{P}'} = t'$ d'où $\sigma' \models \phi$.

$\phi = [\zeta = \eta]$. Supposons $\sigma \models \phi$, c'est-à-dire $\widehat{\zeta\sigma} = \widehat{\eta\sigma}$. Alors, d'après la deuxième partie du corollaire 5.1, $\widehat{\zeta\sigma'} = \widehat{\eta\sigma'}$, d'où : $\sigma' \models \phi$.

$\phi =$ let be z in $=\zeta$ in ϕ'. Supposons $\sigma \models \phi$. On considère : $\sigma_1 = \sigma[\widehat{\zeta\sigma}/z]$ et $\sigma_2 = \sigma'[\widehat{\zeta\sigma'}/z]$. Alors $\sigma \models \phi$ si et seulement si $\sigma_1 \models \phi'$ et $\sigma' \models \phi$ si et seulement si $\sigma_2 \models \phi'$. Pour pouvoir appliquer l'hypothèse d'induction, il nous suffit de montrer que $\sigma_1 \sim' \sigma_2$. Comme $kn(\sigma_1) = kn(\sigma)$ et $kn(\sigma_2) = kn(\sigma')$, il suffit de montrer que $\overline{\widehat{\zeta\sigma}}^\sigma\theta = \overline{\widehat{\zeta\sigma'}}^{\sigma'}$ ce qui est assuré par le lemme 5.1.

$\phi = \phi_1 \vee \phi_2$, $\phi = \phi_1 \wedge \phi_2$ **ou** $\phi = \neg\phi'$. L'étude de ces cas est immédiate. □

Pour montrer que la relation $\sim$ est décidable, il reste à montrer que la deuxième condition $A^i_\sigma = A^i_{\sigma'}$ dans la définition de $\sim'$ est décidable.

Proposition 5.4 *Soient σ une substitution et $\mathcal{P}$ l'ensemble de chemins de $\overline{\sigma}$. Soit P un prédicat tel que son interprétation soit un langage régulier (c'est-à-dire reconnaissable par un automate d'arbre $\mathcal{A}_P$). On pose $A_\sigma \overset{def}{=} \{\zeta \in \mathcal{T}(\mathcal{VF} \cup \{x_p | p \in \mathcal{P}\}) \mid [\![P(\zeta[\sigma|_p/x_p]_{p\in\mathcal{P}})]\!] = tt\}$. Le langage A_σ est régulier.*

Preuve. On construit l'automate $\mathcal{A}$ qui reconnaît le langage A_σ de la manière suivante :
- les transitions $f(q_1, \ldots, q_n) \to q$ de $\mathcal{A}_P$ où f est un symbole visible ($f \in \mathcal{VF}$) sont des transitions de $\mathcal{A}$,
- pour chaque chemin $p \in \mathcal{P}$, si $\sigma|_p$ est accepté par l'automate $\mathcal{A}_P$ dans un état q alors la transition $x_p \to q$ est ajoutée aux transitions de $\mathcal{A}$.

On vérifie alors que $\mathcal{L}(\mathcal{A}) = A_\sigma$. □

À l'aide du théorème 5.1, de la proposition 5.4 et en utilisant le fait que la décision de l'égalité de deux langages réguliers est décidable et EXPTIME-complet [CDG+97], on déduit le résultat suivant.

Corollaire 5.2 *Soient σ et σ' deux substitutions ayant le même domaine. Le problème $\sigma \sim \sigma'$ est décidable et EXPTIME-complet.*

5.2.2.3 Bisimulation

À l'aide de la notion d'équivalence entre environnements définies dans les paragraphes précédents, nous pouvons introduire une notion d'équivalence entre configurations pour le système de transition défini au paragraphe 5.2.1. Cette notion d'équivalence est identique à celle introduite dans [BDNP99].

Notation : La transition $\sigma \triangleright P \overset{\widehat{\mu}}{\underset{\delta}{\Longmapsto}} \sigma' \triangleright P'$ représente $\sigma \triangleright P \overset{\mu}{\underset{\delta}{\Longmapsto}} \sigma' \triangleright P'$ si $\mu \neq \tau$ et $\sigma \triangleright P \overset{\epsilon}{\Longmapsto} \sigma' \triangleright P'$ si $\mu = \tau$. La notation $\widehat{\cdot}$ n'a bien sûr rien à voir avec la fonction d'évaluation $\overline{\cdot}$ définie au paragraphe 5.1.2.1.

On dit qu'une relation binaire $\mathcal{R}$ est *compatible* si elle ne contient que des paires de configurations de la forme $(\sigma_1 \triangleright P,\ \sigma_2 \triangleright Q)$ avec $\sigma_1 \sim \sigma_2$. Étant donnée une relation binaire $\mathcal{R}$, on écrit $(\sigma_1, \sigma_2) \vdash P\mathcal{R}Q$ si la paire $(\sigma_1 \triangleright P, \sigma_2 \triangleright Q)$ est dans $\mathcal{R}$.

Définition 5.10 (bisimulation faible) *Une relation compatible $\mathcal{R}$ est une* bisimulation faible *si pour tous environnements σ_1, σ_2, pour tous protocoles P, Q tels que $(\sigma_1, \sigma_2) \vdash P\mathcal{R}Q$ et $\sigma_1 \triangleright P \overset{\mu}{\underset{\delta}{\longmapsto}} \sigma_1' \triangleright P'$, il existe μ', σ_2' et Q' tels que : $\sigma_2 \triangleright Q \overset{\widehat{\mu'}}{\underset{\delta}{\Longmapsto}} \sigma_2' \triangleright Q'$ et $(\sigma_1', \sigma_2') \vdash P'\mathcal{R}Q'$ et symétriquement en échangeant les rôles de Q et P. La* bisimilarité faible, *notée $\approx$, est la plus grande relation (pour l'inclusion) de bisimulation faible.*

Exemple 5.9 *Nous reprenons une nouvelle fois les processus $P(s)$ et $P(s')$ définis dans l'exemple 5.4. Considérons $\sigma_1 = \sigma_2 = [a/a, b/b, s/s, s'/s']$. Alors $(\sigma_1, \sigma_2) \vdash P(s) \not\approx P(s')$. En effet, on vérifie comme dans l'exemple 5.4 que $\sigma_1 \triangleright P(s) \overset{\mu}{\underset{\delta}{\Longmapsto}} \sigma_1' \triangleright P'''(s)$ où $P'''(s)$ est défini dans l'exemple 5.6 et $\sigma_1' = \sigma_1[< a, b > /x_3, k/k, \{s\}_k/x_4]$. Alors, en particulier pour la formule $\phi \overset{def}{=} [\mathsf{dec}_k(x_4) = s]$, la relation $\sigma_1' \models \phi$ est vérifiée.*

Cette séquence de transition ne peut pas être simulée par $\sigma_2 \triangleright P(s')$ car aucun des σ_2' accessibles ne vérifie la relation $\sigma_2' \models \phi$.

5.3 Équivalence des deux calculs

L'objet de cette partie est de montrer la correspondance entre la bisimilarité faible et l'équivalence barbue. Une partie de la preuve de l'équivalence entre les deux notions est une adaptation simple de la preuve donnée par M. Boreale.

5.3.1 Équivalence barbue étendue

La bisimilarité faible est définie pour des protocoles dans un contexte. Nous commençons donc par étendre l'équivalence barbue aux protocoles *dans un contexte.*

Définition 5.11 *Soient σ_1 et σ_2 deux substitutions équivalentes. Rappelons que l'ensemble $leaves(\overline{\sigma_1}^{\sigma_1})$ désigne l'ensemble des chemins maximaux de $\overline{\sigma_1}^{\sigma_1}$.*

*Une relation binaire S entre protocoles est une (σ_1, σ_2)-*bisimulation barbue *si pour tous protocoles P, Q tels que PSQ, les conditions suivantes sont vérifiées :*

- *pour tout protocole P', si $P \overset{\tau}{\longrightarrow} P'$ alors il existe un protocole Q' tel que $Q \Longrightarrow Q'$ et $P'SQ'$;*
- *pour tout chemin $p \in leaves(\overline{\sigma_1}^{\sigma_1})$, si $P \downarrow \sigma_1|_p$, alors $Q \downarrow \sigma_2|_p$*

et symétriquement en échangeant les rôles de Q et P.

Deux protocoles P et Q sont en (σ_1, σ_2)-bisimilarité barbue, *noté* $(\sigma_1, \sigma_2) \vdash P \dot{\cong} Q$, *si* (P, Q) *appartient à la plus grande relation de* (σ_1, σ_2)*-bisimulation barbue.*

Deux protocoles P et Q sont en (σ_1, σ_2)-équivalence barbue, *noté* $(\sigma_1, \sigma_2) \vdash P \cong Q$, *si pour tout* processus *R tel que $fn(R) \subseteq dom(\sigma_1)$, la relation $(\sigma_1, \sigma_2) \vdash P|R\sigma_1 \dot{\cong} Q|R\sigma_2$ est vérifiée.*

Cette définition est très proche de celle donnée par M. Boreale. La seule différence est que les barbes sont maintenant testées relativement aux expressions irréductibles connues de l'environnement et non plus relativement aux seuls noms connus de l'environnement.

L'équivalence barbue définie précédemment est reliée à l'équivalence barbue étendue aux protocoles dans un environnement de la manière suivante. Les protocoles P et Q sont en équivalence barbue ($P \cong Q$) si et seulement si les protocoles P et Q sont en équivalence barbue dans l'environnement qui connaît toutes les variables libres des protocoles P et Q ($(\epsilon_V, \epsilon_V) \vdash P \cong Q$ où V est un ensemble de noms contenant $fn(P, Q)$ et ϵ_V représente la substitution $[v/v]_{v \in V}$).

La bisimilarité faible est correcte vis-à-vis de l'équivalence barbue.

Théorème 5.2 (correction de la bisimilarité faible) *Soient P et Q deux protocoles et σ_1 et σ_2 deux substitutions équivalentes. Si $(\sigma_1, \sigma_2) \vdash P \approx Q$, alors $(\sigma_1, \sigma_2) \vdash P \cong Q$.*

La preuve de la correction ($\approx \subseteq \cong$) est exactement la même que celle présentée dans [BDNP99], nous ne la referons donc pas ici.

Pour la complétude ($\cong \subseteq \approx$) et comme dans [BDNP99], on se limite à des protocoles ayant une « image finie ».

Définition 5.12 (protocole d'image structurellement finie) *On dit qu'un protocole P est d'*image structurellement finie *si pour toute trace visible s, l'ensemble des classes d'équivalence $\{P' \mid P \stackrel{s}{\Longrightarrow} P'\}/\equiv$ est fini où $\equiv$ est l'équivalence structurelle définie au paragraphe 5.1.3.1.*

Nous verrons au paragraphe 5.4 que cette limitation n'est pas gênante pour les applications car nous nous ramenons à des protocoles sans réplication et ces derniers sont toujours d'image structurellement finie.

Théorème 5.3 (complétude de bisimilarité faible) *Considérons P et Q deux protocoles d'image structurellement finie et σ_1 et σ_2 deux substitutions équivalentes, de domaine fini. Si $(\sigma_1, \sigma_2) \vdash P \cong Q$, alors $(\sigma_1, \sigma_2) \vdash P \approx Q$.*

Remarque : La restriction aux substitutions de domaine fini n'est à nouveau pas une limitation concernant les applications développées au paragraphe 5.4. En effet, pour montrer que deux protocoles sont en équivalence barbue, il suffira de montrer qu'ils sont en bisimilarité faible pour un environnement contenant toutes les variables libres des deux protocoles considérés. Un tel environnement peut facilement être choisi de domaine fini.

L'objet de la partie suivante est de prouver le théorème 5.3.

5.3.2 Éléments de preuve

Le principal argument de la preuve de complétude dans [BDNP99] est l'existence, pour tout environnement σ, d'une *formule caractéristique* ϕ_σ qui vérifie :

$$\forall \sigma' \quad \sigma' \models \phi_\sigma \Leftrightarrow \sigma \sim \sigma'. \tag{5.6}$$

Nous allons voir qu'il n'est plus possible de construire de telles formules car, intuitivement, des formules arbitrairement grandes peuvent être nécessaires pour distinguer deux environnements.

Aussi, nous définissons une nouvelle notion de *formule caractéristique* vérifiant une propriété plus faible que la propriété 5.6, puis une notion de *processus caractéristique* qui caractérise exactement les substitutions équivalentes.

5.3.2.1 Formule caractéristique

Même si nous ne pouvons plus construire des formules vérifiant la propriété 5.6, nous pouvons associer à tout environnement σ, une *formule caractéristique* ϕ_σ qui reconnaît toutes les substitutions qui ont la même « structure » que σ.

Définition 5.13 (formule caractéristique) *Soit σ une substitution de domaine fini. Pour tout chemin p de $\overline{\sigma}^\sigma$, on note ϕ_p et ζ_p, la formule et l'expression publique caractérisant p, comme définies dans la proposition 5.3.*

La formule caractéristique *de σ est :*

$$\phi_\sigma \stackrel{def}{=} \bigwedge_{\substack{p \,\in\, leaves(\overline{\sigma}^\sigma) \\ f \,\in\, \mathcal{VF}}} \left[\phi_p \wedge [f^{-1}(\zeta_p) = \bot] \wedge \epsilon_\sigma[\mathsf{dec}_{\zeta_p}(\{\zeta_p\}_{\zeta_p}) = \bot] \wedge \bigwedge_{p' \in leaves(\overline{\sigma}^\sigma)} \epsilon_\sigma[\zeta_p = \zeta_{p'}] \right].$$

La formule $\epsilon_\sigma \phi$ représente ϕ si $\sigma \models \phi$ et $\neg\phi$ sinon.

Intuitivement, la formule ϕ_σ reconnaît toutes les substitutions σ' qui « étendent » σ : les substitutions σ et σ' vérifient les mêmes égalités et tout chemin de $\overline{\sigma}^\sigma$ est un chemin de $\overline{\sigma'}^{\sigma'}$.

Proposition 5.5 *Soient σ et σ' deux substitutions de domaines finis tels que $dom(\sigma) = dom(\sigma')$. Si $\sigma' \models \phi_\sigma$, alors il existe une substitution injective $\theta : Var_\sigma \to \mathcal{T}(\mathcal{F} \cup Var_{\sigma'})$ telle que :*

1. *$\overline{\sigma'}^{\sigma'} = \overline{\sigma}^\sigma \theta$;*
2. *θ préserve les types des clefs : si $\theta(X_t) = X_{t'}$ alors $t = t^{-1}$ si et seulement si $t' = t'^{-1}$, si $\theta(X_t) = f(t')$ alors $t = t^{-1}$ si et seulement si $f \neq$ pub et $f \neq$ prv.*

Preuve. Soient σ et σ' deux substitutions de domaines finis tels que $dom(\sigma) = dom(\sigma')$. Supposons que $\sigma' \models \phi_\sigma$.

Alors, pour tout chemin p de $leaves(\overline{\sigma}^\sigma)$, $\sigma' \models \phi_p$. D'après les propriétés de ϕ_p et ζ_p, on déduit que p est un chemin de $\overline{\sigma'}^{\sigma'}$ et $s_\sigma(p) = s_{\sigma'}(p)$.

Considérons θ telle que $\theta(\overline{\sigma}|_p) = \overline{\sigma'}|_p$ pour tout chemin p de $leaves(\overline{\sigma}^\sigma)$. Comme $\sigma' \models \phi_\sigma$, on déduit $\sigma|_p = \sigma|_{p'} \Rightarrow \sigma'|_p = \sigma'|_{p'}$, ce qui assure que θ est une fonction. Inversement la propriété $\sigma|_p \neq \sigma|_{p'} \Rightarrow \sigma'|_p \neq \sigma'|_{p'}$ est également vérifiée, ce qui assure que θ est une substitution injective vérifiant : $\overline{\sigma'}^{\sigma'} = \overline{\sigma}^{\sigma}\theta$.

Montrons que θ préserve la symétrie : si $\theta(X_t) = X_{t'}$ (avec $t = \sigma|_p = \widehat{\zeta_p\sigma}$ et $t' = \sigma'|_p = \widehat{\zeta_p\sigma'}$ pour un certain chemin p) alors t est symétrique si et seulement si $[\![[\mathsf{dec}_{\zeta_p}(\{\zeta_p\}_{\zeta_p}) = \bot]\sigma]\!] = ff$, donc t est symétrique si et seulement si t' est symétrique. Même chose lorsque $\theta(X_t) = f(t')$. $\square$

Que nous manque-t-il pour obtenir une unique formule satisfaisant 5.6 ? Intuitivement, la formule ϕ_σ vérifie les propriétés positives de σ : le terme contenu dans la première variable de σ est décomposable jusqu'à une certaine profondeur et la décomposition a une certaine forme. Mais une unique formule ne permet pas de vérifier des propriétés négatives comme : « le terme contenu dans la première variable n'est pas décomposable ».

Exemple 5.10 *Supposons que le spi-calcul considéré ne contienne aucun prédicat V_i et considérons les substitutions $\sigma_1 = [k/x_1, k'/x_2]$ et $\sigma_n = [k/x_1.\{k'\}_{k^n}]$ où k^n, $n \geqslant 2$ est défini par $k^2 =< k, k >$ et $k^{n+1} =< k, k^n >$. Alors $\phi_{\sigma_1} = \neg[x_1 = x_2]$ et $\sigma_n \models \phi_{\sigma_1}$ mais pourtant $\sigma_1 \not\sim \sigma_n$. Pour caractériser toutes les substitutions σ' telles que $\sigma_1 \sim \sigma'$, il faudrait exprimer que les contenus des mémoires x_1 et x_2 ne peuvent pas être analysés avec la connaissance de σ'. Il faudrait ainsi vérifier que $[\mathsf{dec}_{x_1}(x_2) = \bot]$, $[\mathsf{dec}_{<x_1,x_1>}(x_2) = \bot]$, et ainsi de suite pour une infinité de formules.*

5.3.2.2 Processus caractéristique

En fait, pour caractériser toutes les substitutions σ' telles que $\sigma \sim \sigma'$, il faudrait pouvoir en particulier exprimer qu'une expression publique ζ ne peut pas être reconstruite par la substitution σ', c'est-à-dire que pour tout symbole visible $f \in \mathcal{VF}$, pour toutes expressions publiques $\zeta_1, \ldots, \zeta_k$, $\widehat{\zeta\sigma'} \neq f(\widehat{\zeta_1\sigma'}, \ldots, \widehat{\zeta_k\sigma'})$. De même, il faudrait pouvoir exprimer qu'une expression publique ne peut pas être déchiffrée par une substitution σ'. Comme ce type de propriétés ne peut être exprimé par une unique formule, nous introduisons un processus caractéristique qui va tester l'appartenance à une classe d'équivalence pour la relation $\sim$.

Théorème 5.4 *Soit σ une substitution de domaine fini. Il existe un processus R^b_σ et une formule ϕ_σ tels que pour toute substitution σ' vérifiant $dom(\sigma') = dom(\sigma)$, les substitutions σ et σ' sont équivalentes si et seulement si : $\sigma' \models \phi_\sigma$ et $R^b_\sigma\sigma' \not\Downarrow b$.*

Pour démontrer ce théorème, nous introduisons deux symboles de relations binaires vis et enc qui n'appartiennent pas à notre syntaxe. La relation $vis(\sigma, \zeta)$ sera interprétée à vraie si l'environnement σ peut détecter que $\widehat{\zeta\sigma}$ a pour symbole de tête un symbole visible. La relation $enc(\sigma, \zeta)$ sera interprétée à vraie si l'environnement σ peut détecter que $\widehat{\zeta\sigma}$ est un message chiffré en calculant explicitement l'inverse de la clef.

$$[\![vis(\sigma,\zeta)]\!] = \begin{cases} tt & \text{s'il existe } f \in \mathcal{VF},\ \zeta_1, \ldots, \zeta_n \text{ tels que } \widehat{\zeta\sigma} = f(\widehat{\zeta_1\sigma}, \ldots, \widehat{\zeta_n\sigma}), \\ ff & \text{sinon.} \end{cases}$$

$$[\![enc(\sigma,\zeta)]\!] = \begin{cases} tt & \text{s'il existe } \zeta' \text{ tel que } \widehat{\mathsf{dec}_{\zeta'}(\zeta)} \neq \bot, \\ ff & \text{sinon.} \end{cases}$$

Exemple 5.11 *Considérons les substitutions $\sigma_1 = [k/x_1, k'/x_2]$ et $\sigma_n = [k/x_1.\{k'\}_{k^n}]$ définies dans l'exemple 5.10. Alors : $[\![enc(\sigma_1, x_2)]\!] = ff$ et $[\![enc(\sigma_n, x_2)]\!] = tt$.*

De plus, on dit que σ satisfait la formule $vis(\cdot, \zeta)$ si $[\![vis(\sigma, \zeta)]\!] = tt$ et symétriquement pour la formule $enc(\cdot, \zeta)$.

À l'aide de ces deux relations, nous définissons une nouvelle formule associée à une substitution σ :

$$\phi_\sigma^{char} \stackrel{\text{def}}{=} \phi_\sigma \wedge \bigwedge_{p \in leaves(\overline{\sigma}^\sigma)} \neg vis(\cdot, \zeta_p) \wedge \neg enc(\cdot, \zeta_p).$$

La proposition 5.5 nous permet de déduire que la formule ϕ_σ^{char} caractérise presque entièrement la classe d'équivalence de σ.

Proposition 5.6 *Soient σ et σ' deux substitutions de domaine fini telles que $dom(\sigma) = dom(\sigma')$. Alors les deux substitutions sont équivalentes si et seulement si $\sigma' \models \phi_\sigma^{char}$ et pour tout entier i tel que $1 \leqslant i \leqslant n$, les ensembles A_σ^i et $A_{\sigma'}^i$ sont égaux.*

La preuve du théorème 5.4 se ramène maintenant à la construction de deux processus. D'une part, on construit un processus caractéristique $R_{\zeta,\sigma}^b$ tel que $R_{\zeta,\sigma}^b\sigma'$ se compromet sur b si et seulement si σ' ne satisfait pas la formule $\neg vis(\cdot, \zeta) \wedge \neg enc(\cdot, \zeta)$. D'autre part, on construit $R_{i,\sigma}^b$ tel que $R_{i,\sigma}^b\sigma'$ se compromet sur b si et seulement si $A_\sigma^i \neq A_{\sigma'}^i$. Ces deux constructions font l'objet des deux paragraphes suivants. Il suffit ensuite de prendre pour R_σ^b la composition parallèle des processus $R_{\zeta_p,\sigma}^f$ et $R_{i,\sigma}^b$ pour tout chemin p de $leaves(\overline{\sigma}^\sigma)$ et pour tout i entier tel que $1 \leqslant i \leqslant n$.

Caractérisation du caractère visible ou chiffré d'une expression

Soient $\sigma = [M_i/x_i]_{i \leqslant n}$ une substitution de domaine fini, b un nom et ζ une expression publique. Nous définissons le processus $R_{\zeta,\sigma}^b$ de la manière suivante :

$$\begin{array}{rcl}
R_{\zeta,\sigma}^b \stackrel{\text{def}}{=} & (\nu a)(!\bar{a}(x_1) \mid \ldots \mid !\bar{a}(x_n) & \\
& \mid a(y_1). \ldots .a(y_m).!\bar{a}(f(y_1, \ldots, y_m)) & \forall f \in \mathcal{VF} \\
& \mid a(x).a(y).!\bar{a}(\mathsf{dec}_x(y)) & \\
& \mid a(x).!\bar{a}(g_i^{-1}(x)) & \forall g \in \mathcal{IF}, \forall i \leqslant arity(g) \\
& \mid a(y_1). \ldots .a(y_m).[f(y_1, \ldots, y_m) = \zeta]\bar{b}(\zeta) & \forall f \in \mathcal{VF} \\
& \mid a(x).a(y).[\mathsf{dec}_x(y) = \zeta]\bar{b}(\zeta)) &
\end{array}$$

Soit $\sigma' = [M_i'/x_i]_{i \leqslant n}$ une substitution de même domaine que σ. Intuitivement, le processus $R_{\zeta,\sigma}^b\sigma'$ envoie les messages M_i' un nombre arbitraire de fois sur un canal privé a. De plus, il calcule tous les messages M constructibles par l'environnement σ', c'est-à-dire les termes M tel que $M = \widehat{\zeta'\sigma'}$ pour une certaine expression publique ζ'. Enfin, pour tous termes $M_1, \ldots, M_m$ ($M_i = \widehat{\zeta_i\sigma'}$) ainsi obtenus, le processus $R_{\zeta,\sigma}^b\sigma'$ teste si $\widehat{\zeta\sigma'} = f(M_1, \ldots, M_m)$ ou $\widehat{\zeta\sigma'} = \mathsf{dec}_{M_1}(M_2)$ et se compromet sur b si c'est le cas. Formellement, on obtient le lemme suivant.

Lemme 5.2 *Soient $\sigma = [M_i/x_i]_{i \leqslant n}$ une substitution de domaine fini, b un nom, ζ une expression publique et $R_{\zeta,\sigma}^b$ le processus défini ci-dessus. Soit σ' substitution telle que $dom(\sigma') = dom(\sigma)$. Les propriétés suivantes sont vérifiées :*

- *Il existe une suite d'actions μ telle que $R_\zeta^f \stackrel{\mu}{\Rightarrow} R' \xrightarrow{\bar{a}<M>} R''$ si et seulement s'il existe une expression publique ζ telle que $\widehat{\zeta\sigma} = M$.*

– *$R^f_\zeta \overset{\mu}{\Rightarrow} R' \xrightarrow{\bar{f}<M>} R''$ pour un certain message M et une certaine suite d'actions μ si et seulement l'une des relations suivantes est vérifiée : $vis(\sigma', \zeta_p)$ ou $enc(\sigma', \zeta_p)$.*

Preuve. La première propriété de ce lemme se vérifie par induction sur la longueur de la suite d'action μ. La deuxième propriété est une conséquence directe de la première. □

Caractérisation de l'appartenance d'une expression à un langage régulier

Soient $\sigma = [M_i/x_i]_{i \leqslant n}$ une substitution de domaine fini, b un nom et i un entier tel que $1 \leqslant i \leqslant n$ où n est le nombre de prédicats du spi-calcul considéré. Il reste à construire un processus $R^b_{i,\sigma}$ tel que $R^b_{i,\sigma}$ se compromet sur b si et seulement si $A^i_\sigma \neq A^i_{\sigma'}$. On rappelle que : $A^i_\sigma = \{\zeta \in \mathcal{T}(\mathcal{VF} \cup \{x_p | p \in \mathcal{P}\}) \,|\, [\![V_i(\zeta[\sigma|_p/x_p]_{p \in \mathcal{P}})]\!] = tt\}$ où $\mathcal{P}$ est l'ensemble des chemins de $\overline{\sigma}^\sigma$. On suppose définis, pour chaque chemin p de $\mathcal{P}$, l'expression ζ_p et la formule ϕ_p caractérisant p selon la proposition 5.3. D'après la proposition 5.4, l'ensemble A^i_σ est régulier. Soit $\mathcal{A}$ l'automate d'arbre qui le reconnaît. L'automate $\mathcal{A}$ a un ensemble Q d'états, un ensemble Q_f d'états finals et un ensemble T de transition. En utilisant les règles de l'automate $\mathcal{A}$, on commence par construire un processus $R^{i,1}_\sigma$ tel que $R^{i,1}_\sigma$ se compromet sur b si et seulement s'il existe une expression ζ telle que $\zeta \in A^i_\sigma$ et $\zeta \notin A^i_{\sigma'}$.

Tout d'abord, pour chaque chemin p de $\mathcal{P}$ et pour chaque état q de Q, on pose :

$$P_{p,q} = \begin{cases} !(\bar{a} < \zeta_p, q >) & \text{si } x_p \to q_p \in T \\ \mathbf{0} & \text{sinon} \end{cases}.$$

De plus, pour chaque transition t de T, où $t = f(q_1, \ldots, q_k) \to q$, on définit :

$$P_t \overset{\text{def}}{=} !(a(x_1).\, \mathsf{let}\, x_1 =< x'_1, q_1 > \,\mathsf{in}\, (a(x_2).\, \mathsf{let}\, x_2 =< x'_2, q_2 > \,\mathsf{in}\, \cdots \\ (a(x_k).\, \mathsf{let}\, x_k =< x'_k, q_k > \,\mathsf{in}\, \bar{a} < f(x'_1, \ldots, x'_k), q >)$$

où $\mathsf{let}\, x_i =< x'_i, q_i > \,\mathsf{in}\, R$ représente le processus $[q_i =<>^{-1}_2 x_i]\, \mathsf{let}\, x'_i =<>^{-1}_1 x_i \,\mathsf{in}\, R$.

Le processus P_t simule la transition t de l'automate $\mathcal{A}$. Il reste à construire un processus qui teste si les expressions acceptées dans un état final satisfont le prédicat V_i. Soit q_f un état final de $\mathcal{A}$, on définit :

$$P_{q_f} \overset{\text{def}}{=} !(a(x).\, \mathsf{let}\, x =< x', q_f > \,\mathsf{in}\, \neg P_i(x').\bar{b}).$$

On construit alors le processus $R^{i,1}_\sigma$ de la manière suivante :

$$R^{i,1}_\sigma \overset{\text{def}}{=} (\nu a) \mid_{\left(\substack{p \in \mathcal{P} \\ q \in Q}\right)} \phi_p P_{p,q} \mid_{t \in T} P_t \mid_{q \in Q_f} P_{q_f},$$

où $|_{i \in J} P_i$ représente la composition parallèle des processus P_i pour i dans J.

Inversement, on construit un processus $R^{i,2}_\sigma$ tel que $R^{i,2}_\sigma$ se compromet sur b si et seulement s'il existe une expression ζ telle que $\zeta \notin A^i_\sigma$ et $\zeta \in A^i_{\sigma'}$. Pour cela, on considère $\mathcal{A}^c$ l'automate d'arbre reconnaissant le complémentaire de A^i_σ. On pose Q^c l'ensemble de ses états, Q^c_f l'ensemble de ses états finals et T^c l'ensemble de ses transitions. Pour tout état final q_f de Q^c_f, on définit

$$P'_{q_f} \overset{\text{def}}{=} !(a(x).\, \mathsf{let}\, x =< x', q_f > \,\mathsf{in}\, P_i(x').\bar{b}).$$

La définition des autres processus intermédiaires ($P_{p,q}$ et P_t) est inchangée. Le processus $R_\sigma^{i,2}$ est construit de la manière suivante :

$$R_\sigma^{i,1} \stackrel{\text{def}}{=} (\nu a) \mid_{\left(\begin{smallmatrix} p \in \mathcal{P} \\ q \in Q^c \end{smallmatrix}\right)} \phi_p P_{p,q} \mid_{t \in T^c} P_t \mid_{q \in Q_f^c} P'_{q_f}.$$

Les processus $R_\sigma^{i,1}$ et $R_\sigma^{i,2}$ vérifient les propriétés demandées.

Lemme 5.3 *Soient $\sigma = [M_i/x_i]_{i \leqslant n}$ une substitution de domaine fini, b un nom , i un entier tel que $1 \leqslant i \leqslant n$ où n est le nombre de prédicats du spi-calcul considéré et $R_\sigma^{i,1}$ et $R_\sigma^{i,2}$ les processus décrits précédemment. Pour toute substitution σ' telle que $dom(\sigma') = dom(\sigma)$, les propriétés suivantes sont vérifiées :*
- *$R_\sigma^{i,1}\sigma' \Downarrow f$ si et seulement s'il existe une expression ζ telle que : $\zeta \in A_\sigma^i$ et $P_i(\zeta[\zeta_p/x_p]_{p \in \mathcal{P}}\sigma') = \mathit{ff}$,*
- *$R_\sigma^{i,2}\sigma' \Downarrow f$ si et seulement s'il existe une expression ζ telle que : $\zeta \notin A_\sigma^i$ et $P_i(\zeta[\zeta_p/x_p]_{p \in \mathcal{P}}\sigma') = \mathit{tt}$.*

On en déduit que $A_\sigma^i = A_{\sigma'}^i$ si et seulement si $R_\sigma^{i,1}\sigma' \not\Downarrow f$ et $R_\sigma^{i,2}\sigma' \not\Downarrow f$.

Ce lemme est vérifié par induction sur ζ.

On pose alors $R_{i,\sigma}^b = R_\sigma^{i,1} \mid R_\sigma^{i,2}$. Le processus $R_{i,\sigma}^b$ satisfait les propriétés demandées, ce qui termine la démonstration du théorème 5.4.

La suite de la preuve du théorème 5.3 de complétude est une adaptation simple de la preuve présentée par M. Boreale *et al.* dans [BDNP99].

5.3.2.3 Complétude de la bisimilarité faible vis-à-vis de l'équivalence barbue

Nous récrivons la preuve donnée par M. Boreale *et al.* dans [BDNP99], adaptée à notre nouvelle syntaxe. Afin d'alléger la preuve, nous réutiliserons les définitions et notations de M. Boreale sans rappeler celles qui ne changent pas. Par conséquent, la preuve présentée ici n'est pas complètement lisible sans la preuve donnée dans [BDNP99]. Nous avons choisi cette forme de présentation car, une fois le théorème 5.4 démontré sur les processus caractéristiques, l'apport personnel dans la preuve de complétude est minime.

Ainsi, nous supposons définie la chaîne de relations $\approx_i$, $i \geqslant 0$ présentée dans [BDNP99]pour approcher la relation de bisimilarité $\approx$ sur les configurations. Nous définissons le *rang* d'une substitution σ par $rk(\sigma) = |\phi_\sigma|$, où $|E|$ désigne la taille syntaxique d'un terme E, c'est à dire le nombre d'occurrences de symboles dans E. On pose $\approx_w \stackrel{\text{def}}{=} \cap_{i \geqslant 0} \approx_i$.

Les lemmes 4.20 et 4.22 de [BDNP99] sont toujours vérifiés.

Lemme 5.4 (Lemme 4.20) *Soient P et Q deux processus d'image structurellement finie. La relation $(\sigma_1, \sigma_2) \vdash P \approx Q$ est vérifiée si et seulement si la relation $(\sigma_1, \sigma_2) \vdash P \approx_w Q$ l'est également.*

Lemme 5.5 (Lemme 4.22) *Soient σ et σ' deux substitutions de même domaine (fini). Si $\sigma \sim \sigma'$ alors $rk(\sigma) = rk(\sigma')$.*

En fait, si σ et σ' deux substitutions équivalentes, elles vérifient une propriété plus forte que celle du lemme 5.5 : $\phi_\sigma = \phi_{\sigma'}$ par construction des formules ϕ_σ.

Nous allons maintenant montrer que la relation $\cong$ implique la relation $\approx_w$, ce qui montrera le théorème de complétude. Pour cela, nous redéfinissons la classe des contextes canoniques $R_{i,\sigma}$ dépendants d'un entier $i \geqslant 0$ et d'une substitution σ.

Notation : La notation $\tau.P$ représente le processus $(\nu c)(c(x).P \mid \bar{c} < c >)$ où c et x ne sont pas des noms libres de P : $c, x \notin fn(P)$.

Soient $P_1, \ldots, P_k$ des processus en nombre fini, $\sum\{P_1, \ldots, P_k\}$ représente la somme non déterministe des processus : $P_1 + \cdots + P_k$.

Définition 5.14 (contexte canonique) *Les contextes canoniques $R_{i,\sigma}$ pour $i \geqslant 0$ sont définis par induction sur i de la manière suivante : $R_{0,\sigma} \stackrel{def}{=} \mathbf{0}$ et pour tout entier i tel que $i > 0$:*

$$
\begin{array}{rcl}
R_{n,\sigma} & \stackrel{def}{=} & \sum\{R_\eta^{inp} + R_\eta^{out} : n(\eta) \subseteq dom(\sigma), \widehat{\eta\sigma} \in \mathcal{N} \text{ et } |\eta| < i\} + R_\epsilon + \bar{e}_i \text{ où :} \\
R_\eta^{inp} & \stackrel{def}{=} & \eta(x). \sum\{\phi_{\sigma'}(R_{\sigma'}^{h_{\eta,\sigma',i}} \mid \bar{f}_{\eta,\phi_{\sigma'},i} + \tau.R_{i-1,\sigma'}) : \\
 & & \qquad \text{il existe } M \text{ t.q. } \sigma' = \sigma[M/x], \text{ et } rk(\sigma') \geqslant i\} \\
R_\eta^{out} & \stackrel{def}{=} & \sum\{(\nu\tilde{b})\overline{\eta}(\zeta).(\bar{g}_{\eta,\zeta,i} + \tau.R_{i-1,\sigma'}) : fn((\nu\tilde{b})\overline{\eta}(\zeta)) \subseteq dom(\sigma), \sigma' = \sigma[\tilde{b}/\tilde{b}] \text{ et } |\zeta| < i\} \\
R_\epsilon & \stackrel{def}{=} & \tau.(\bar{h}_i) + \tau.R_{i-1,\sigma}
\end{array}
$$

où les noms e_j, $f_{\eta,\phi_{\sigma'},i}$, $g_{\eta,\zeta,i}$ et h_j pour j entier tel que $1 \leqslant j \leqslant i$ sont tous distincts entre eux et distincts des noms de σ.

De la même manière que dans [BDNP99], la somme R_η^{inp} est finie car, pour $\tilde{x}$ et i fixés, il n'y a qu'un nombre fini de formules phi_σ et de processus $R_{i,\sigma}$ tels que $dom(\sigma) = \tilde{x}$ et $rk(\sigma) \leqslant i$ (on le prouve formellement par induction sur i). De même, la somme R_η^{out} est finie car les actions $(\nu\tilde{b})\overline{\eta}(\zeta)$ sont considérées à alpha-équivalence près. De plus, $fn(R_{i,\sigma}) \subseteq dom(\sigma) \cup \tilde{l}$, où $\tilde{l}$ est l'ensemble des noms $e_j, h_{\eta,\sigma',j}, f_{\eta,\phi_{\sigma'},j}, g_{\eta,\zeta,j}$ et h_j $(0 \leqslant j \leqslant i)$ apparaissant dans $R_{i,\sigma}$.

Supposons maintenant que $(\sigma_1, \sigma_2) \vdash P \cong Q$ et montrons que $(\sigma_1, \sigma_2) \vdash P \approx Q$. D'après le lemme 4.20, il suffit de montrer que pour tout entier i tel que $i \geqslant 0$ la propriété $(\sigma_1, \sigma_2) \vdash P \approx_i Q$ est vérifiée. Considérons le processus canonique R_{i,σ_1}. On pose $\rho_1 = \sigma_1[\tilde{l}/\tilde{l}]$, où $\tilde{l}$ est l'ensemble des noms (distincts de ceux apparaissant déjà dans σ_1) $e_j, h_{\eta,\sigma',j}, f_{\eta,\phi_{\sigma'},j}, g_{\eta,\zeta,j}$ et h_j $(0 \leqslant j \leqslant i)$ apparaissant dans R_{i,σ_1}. Montrons par induction sur i que si $(\rho_1, \rho_2) \vdash P|R_{i,\sigma_1}\sigma_1 \cong Q|R_{i,\sigma_1}\sigma_2$, alors $(\sigma_1, \sigma_2) \vdash P \approx_i Q$.

Le cas $i = 0$ est immédiat. Considérons $i > 0$. Nous considérons uniquement le cas d'une transition (E-INP) car les autres cas sont semblables ou plus faciles. La transition a la forme suivante :

$$\sigma_1 \triangleright P \xmapsto[\eta(x)]{(\nu\tilde{b})\bar{a}<M>} \sigma_1' \triangleright P' \tag{5.7}$$

avec $|\eta(x)| \leqslant i, \sigma_1' = \sigma_1[M/x]$ et $rk(\sigma1') \leqslant i$. Nous allons montrer l'existence d'une transition pour la configuration $\sigma_2 \triangleright Q$ qui simule la première. D'après l'équation (5.7) ci-dessus, on déduit que :

$$P \mid R_{i,\sigma_1}\sigma_1 \xrightarrow{\tau} (\nu\tilde{b})(P' \mid \phi_{\sigma_1'}(R_{\sigma_1'}^{h_{\eta,\sigma_1',i}} \mid \bar{f}_{\eta,\phi_{\sigma_1'},i} + \tau.R_{i-1,\sigma_1'})\sigma_1') \stackrel{\text{def}}{=} A.$$

Comme $(\rho_1, \rho_2) \vdash P|R_{i,\sigma_1}\sigma_1 \stackrel{\cdot}{\cong} Q|R_{i,\sigma_1}\sigma_2$ par hypothèse et comme $A \downarrow f_{\eta,\phi_{\sigma_1'},i}$, on déduit l'existence d'une transition

$$Q \mid R_{i,\sigma_1}\sigma_2 \Rightarrow (\nu\tilde{b})(Q' \mid \phi_{\sigma_1'}(R_{\sigma_1'}^{h_{\eta,\sigma_1',i}}\sigma_2' \mid \bar{f}_{\eta,\phi_{\sigma_1'},i} + \tau.R_{i-1,\sigma_1'})\sigma_2') \stackrel{\text{def}}{=} B.$$

avec $(\rho_1, \rho_2) \vdash A \dot{\cong} B$ et $B \downarrow f_{\eta,\phi_{\sigma'_1},i}$. De plus, la substitution σ'_2 vérifie $\sigma'_2 = \sigma_2[M'/x]$ avec $Q \xRightarrow{(\nu\tilde{b'})\overline{a'}<M'>} Q'$ $(a' = \widehat{\eta\sigma_2})$. D'où

$$\sigma_2 \triangleright Q \xRightarrow[\eta(x)]{(\nu\tilde{b'})\overline{a'}<M'>} \sigma'_2 \triangleright Q'. \tag{5.8}$$

Comme $B \downarrow f_{\eta,\phi_{\sigma'_1},i}$, la propriété $\sigma'_2 \models \phi_{\sigma'_1}$ est vérifiée. En outre, comme $R_{\sigma'_1}^{h_{\eta,\sigma'_1,i}} \sigma'_1 \not\Downarrow h_{\eta,\sigma'_1,i}$ et $(\rho_1, \rho_2) \vdash A \dot{\cong} B$, il vient $R_{\sigma'_1}^{h_{\eta,\sigma'_1,i}} \sigma'_2 \not\Downarrow h_{\eta,\sigma'_1,i}$ donc, en appliquant le théorème 5.4, les substitutions σ'_1 et σ'_2 sont équivalents : $\sigma'_1 \sim \sigma'_2$.

Maintenant, comme $A \xrightarrow{\tau}\equiv (\nu\tilde{b})((P' \mid R_{i-1,\sigma'_1})\sigma'_1) \stackrel{\text{def}}{=} A'$, on déduit que $B \Longrightarrow B'$ avec $(\rho_1, \rho_2) \vdash A' \dot{\cong} B'$. Comme $A' \downarrow e_{i-1}$, le processus B' doit également vérifier que $B' \Downarrow e_{i-1}$, donc nécessairement $B' \equiv (\nu\tilde{b'})((Q'' \mid R_{i-1,\sigma'_1})\sigma'_2)$ avec $Q' \Longrightarrow Q''$. En retirant les restrictions $(\nu\tilde{b})$ et $(\nu\tilde{b'})$, on obtient :

$$(\rho_1, \rho_2) \vdash P' \mid R_{i-1,\sigma'_1})\sigma'_1 \dot{\cong} Q'' \mid R_{i-1,\sigma'_1})\sigma'_2.$$

En appliquant l'hypothèse d'induction, on déduit que $(\sigma'_1, \sigma'_2) \vdash P' \approx_{i-1} Q''$. Comme $Q' \Longrightarrow Q''$ et d'après l'équation 5.8, on déduit :

$$\sigma_2 \triangleright Q \xRightarrow[\eta(x)]{(\nu\tilde{b'})\overline{a'}<M'>} \sigma'_2 \triangleright Q'.$$

Cette transition simule bien la transition 5.7 car $\sigma'_1 \sim \sigma'_2$ et $(\sigma'_1, \sigma'_2) \vdash P' \approx_{i-1} Q''$, ce qui termine la preuve du théorème 5.3.

5.4 Application à la preuve du secret

Nous montrons dans cette partie comme appliquer le théorème d'équivalence entre bisimilarité faible et équivalence observationnelle à la preuve du secret de protocoles. Nous commençons par établir des propriétés utiles pour prouver que deux protocoles sont en bisimilarité faible puis nous appliquons ces résultats à la preuve du secret du protocole de Needham-Schroeder-Lowe, pour un nombre non borné de sessions et sans typage de ses composantes.

5.4.1 Préliminaires

Dans cette partie, nous allons introduire une nouvelle notion de bisimilarité entre protocoles puis établir des propriétés de clôture et de composition pour cette nouvelle bisimilarité.

5.4.1.1 Bisimilarité faible restreinte

Notre technique pour montrer le secret d'un protocole P est de construire un environnement σ tel que, quelle que soit l'évolution de P, la connaissance de σ n'augmente pas. Pour cela, il faut au moins empêcher l'environnement de créer des nouveaux nonces « en cours de route » et lui permettre plutôt d'en créer autant que nécessaire au départ. Aussi, nous définissons une nouvelle bisimilarité entre processus, notée $\approx'$. La relation $\approx'$ est définie de la même manière que $\approx$ à l'exception du fait que les actions $\sigma_1 \triangleright P \underset{\delta}{\overset{\mu}{\longmapsto}} \sigma_1' \triangleright P'$ sont de la forme $(\mathsf{E}-\mathsf{OUT})$, $(\mathsf{E}-\mathsf{TAU})$ et $(\mathsf{E}-\mathsf{INP}')$ où l'action $(\mathsf{E}-\mathsf{INP}')$ est définie de la même manière que l'action $(\mathsf{E}-\mathsf{INP})$ mais l'environnement ne peut plus créer de nouveaux nonces.

Définition 5.15 (bisimilarité faible restreinte) *Supposons pour cette définition que les actions $\underset{\delta}{\overset{\mu}{\longmapsto}}$ de configurations sont de la forme (E-OUT), (E-TAU) (actions définies à la figure 5.3) et (E-INP') où (E-INP') est défini de la manière suivante :*

$$(E-INP')\frac{P \xrightarrow{aM} P' \quad \widehat{\eta\sigma} \quad M=\widehat{\zeta\sigma} \quad n(\zeta)\subset dom(\sigma)}{\sigma\triangleright P \underset{\overline{\eta}(\zeta)}{\overset{aM}{\longmapsto}} \sigma\triangleright P'}.$$

La bisimilarité faible restreinte, *notée $\approx'$, est alors la plus grande relation binaire compatible $\mathcal{R}$ sur les configurations vérifiant que pour toutes substitutions σ_1 et σ_2, pour tous protocoles P et Q tels que $(\sigma_1,\sigma_2)\vdash P\mathcal{R}Q$, si $\sigma_1 \triangleright P \underset{\delta}{\overset{\mu}{\longmapsto}} \sigma_1' \triangleright P'$ pour une certaine action $\underset{\delta}{\overset{\mu}{\longmapsto}}$ alors il existe μ',σ_2' et Q' tels que $\sigma_2\triangleright Q \underset{\delta}{\overset{\widehat{\mu'}}{\Longmapsto}} \sigma_2'\triangleright Q'$ et $(\sigma_1',\sigma_2')\vdash P'\mathcal{R}Q'$ et symétriquement en inversant les rôles de P et Q.*

Comme annoncé, les relations $\approx$ et $\approx'$ sont équivalentes à condition de permettre à l'environnement de posséder un nombre arbitraire de noms au départ.

Proposition 5.7 *Soient σ_1 et σ_2 deux substitutions équivalentes et P et Q deux protocoles. La relation $(\sigma_1,\sigma_2)\vdash P\approx Q$ est vérifiée si et seulement si pour toutes suites finies de noms $\tilde{h}$ et $\tilde{k}$ telles que $|\tilde{h}|=|\tilde{k}|$ et $\tilde{h}\cap dom(\sigma_1)=\tilde{k}\cap dom(\sigma_2)=\emptyset$, la relation $(\sigma_1[\tilde{h}/\tilde{h}],\sigma_2[\tilde{k}/\tilde{k}])\vdash P\approx' Q$ est vérifiée.*

Preuve. Soient σ_1 et σ_2 deux substitutions équivalentes et P et Q deux processus. On considère la relation binaire compatible $\mathcal{R}$ définie par $(\sigma_1,\sigma_2)\vdash P\mathcal{R}Q$ si et seulement si toutes suites de noms $\tilde{h}$ et $\tilde{k}$ telles que $|\tilde{h}|=|\tilde{k}|$ et $\tilde{h}\cap dom(\sigma_1)=\tilde{k}\cap dom(\sigma_2)=\emptyset$, la relation $(\sigma_1[\tilde{h}/\tilde{h}],\sigma_2[\tilde{k}/\tilde{k}])\vdash P\approx' Q$ est vérifiée. Montrons que $\mathcal{R}$ est une bisimulation faible.

On suppose que $(\sigma_1,\sigma_2)\vdash P\mathcal{R}Q$ et que $\sigma_1\triangleright P \underset{\delta}{\overset{\mu}{\longmapsto}} \sigma_1'\triangleright P'$ pour une certaine action $\underset{\delta}{\overset{\mu}{\longmapsto}}$. Supposons que cette action soit de la forme E-INP (les autres cas sont immédiats) :

$$\sigma_1\triangleright P \underset{(\nu\tilde{b})\overline{\eta}(\zeta)}{\overset{aM}{\longmapsto}} \sigma_1[\tilde{b}/\tilde{b}]\triangleright P', \tag{5.9}$$

avec $P\xrightarrow{aM}P'$, $\widehat{\eta\sigma}=a$, $M=\widehat{\zeta\sigma}$, $\tilde{b}=(n(\zeta)-dom(\sigma))$ et $\tilde{b}\cap fn(P)=\emptyset$. On choisit $\tilde{k}$ telle que $|\tilde{b}|=|\tilde{k}|$, $\tilde{k}\cap dom(\sigma_2)=\emptyset$ et $\tilde{k}\cap fn(Q)=\emptyset$. De la transition 5.9, on déduit :

$$\sigma_1[\tilde{b}/\tilde{b}]\triangleright P \underset{\overline{\eta}(\zeta)}{\overset{aM}{\longmapsto}} \sigma_1[\tilde{k}/\tilde{k}]\triangleright P'.$$

Cette nouvelle transition est cette fois de la forme (E-INP'). De plus, $(\sigma_1[\tilde{b}/\tilde{b}],\sigma_2[\tilde{k}/\tilde{k}])\vdash P\approx' Q$ par hypothèse. On déduit qu'il existe μ',σ_2' et Q' tels que $\sigma_2[\tilde{k}/\tilde{k}]\triangleright Q \underset{\overline{\eta}(\zeta)}{\overset{\widehat{\mu'}}{\Longmapsto}} \sigma_2'\triangleright Q'$ et $(\sigma_1',\sigma_2')\vdash P'{\approx'}Q'$. D'où $\sigma_2\triangleright Q \underset{(\nu\tilde{k})\overline{\eta}(\zeta)}{\overset{\widehat{\mu'}}{\Longmapsto}} \sigma_2'\triangleright Q'$. De plus, on vérifie aisément que pour toutes suites $\tilde{h'}$ et $\tilde{k'}$ telles que $|\tilde{h'}| = |\tilde{k'}|$ et $\tilde{h'} \cap dom(\sigma'_1) = \tilde{k'} \cap dom(\sigma'_2) = \emptyset$, la relation $(\sigma'_1[\tilde{h'}/\tilde{h'}], \sigma'_2[\tilde{k'}/\tilde{k'}]) \vdash P' \approx' Q'$ est vérifiée. On en déduit que $(\sigma'_1, \sigma'_2) \vdash P'\mathcal{R}Q'$ et donc $\mathcal{R}$ est une bisimulation faible. □

5.4.1.2 Propriétés de clôture

Nous donnons ici quatre propriétés de « clôture » pour la bisimulation faible et pour la bisimulation faible restreinte. Ainsi, la propriété d'*affaiblissement* assure que si deux protocoles sont en bisimilarité faible pour deux environnements, ils sont également en bisimilarité faible pour des environnements plus petits. La propriété de *restriction* assure que si deux protocoles P et Q sont en bisimilarité faible pour deux environnements, les protocoles restreints $(\nu\tilde{h})P$ et $(\nu\tilde{k})Q$ ont également en bisimilarité faible pour les mêmes environnements pourvu que les noms de $\tilde{h}$ et $\tilde{k}$ n'apparaissent pas dans les environnements considérés.

Proposition 5.8 *Soient P et Q deux protocoles, σ_1 et σ_2 des substitutions équivalentes. Les propriétés suivantes sont vérifiées.*

Affaiblissement *Si $(\sigma_1[M_i/x_i]_{i\in J}, \sigma_2[N_i/x_i]_{i\in J}) \vdash P \approx Q$ alors $(\sigma_1, \sigma_2) \vdash P \approx Q$, où les M_i et les N_i sont des messages.*

Contraction *Soit ζ une expression publique telle que : $n(\zeta) \subseteq dom(\sigma_1)$ et $\widehat{\zeta\sigma_1} \neq \perp$. Si $(\sigma_1, \sigma_2) \vdash P \approx Q$ alors $(\sigma_1[\widehat{\zeta\sigma_1}/x], \sigma_2[\widehat{\zeta\sigma_2}/x]) \vdash P \approx Q$.*

Composition *Soit R un processus tel que : $fn(R) \subseteq dom(\sigma_1)$. Si $(\sigma_1, \sigma_2) \vdash P \approx Q$ alors $(\sigma_1, \sigma_2) \vdash P|R\sigma_1 \approx Q|R\sigma_2$.*

Restriction *Soient $\tilde{h}$ et $\tilde{k}$ tels que $\tilde{k} \cap n(\sigma_1) = \emptyset$ et $\tilde{h} \cap n(\sigma_2) = \emptyset$. Si $(\sigma_1, \sigma_2) \vdash P \approx Q$ alors $(\sigma_1, \sigma_2) \vdash (\nu\tilde{k})P \approx (\nu\tilde{h})Q$.*

De plus, la relation $\approx'$ vérifie les mêmes propriétés.

Nous ne ferons pas la preuve de ces propriétés car elle est identique à celle présentée par M. Boreale *et al.* dans [BDNP99].

5.4.1.3 σ-sécurité

Comme M. Boreale, nous introduisons une notion de σ-sécurité qui indique qu'un protocole n'augmente pas la connaissance de l'environnement σ.

Définition 5.16 (σ-sécurité) *Soient P un protocole et σ une substitution. Le protocole est dit* σ-sauf *si pour toute suite d'actions s telle que $\sigma \triangleright R \overset{s}{\underset{u}{\Longmapsto}} \sigma' \triangleright R'$ et $R' \xrightarrow{(\nu\tilde{b})\overline{a}<M>} R''$, alors $a, M \in kn(\sigma)$.*

On dit également qu'un protocole est σ-sauf* *si pour toute suite d'actions s* prises dans (E − OUT), (E − TAU) *ou* (E − INP′), *si $\sigma \triangleright R \overset{s}{\underset{u}{\Longmapsto}} \sigma' \triangleright R'$ et $R' \xrightarrow{(\nu\tilde{b})\overline{a}<M>} R''$, alors $a, M \in kn(\sigma)$.*

L'intérêt des protocoles σ-saufs et σ-saufs* est qu'ils sont en bisimilarité faible avec eux-mêmes et qu'ils peuvent se composer.

Proposition 5.9 *Soient σ et σ' deux substitutions équivalentes et P un protocole σ et σ'-sauf (resp. σ et σ'-sauf*). Le protocole P est en bisimilarité faible (resp. en bisimilarité faible restreinte) avec lui-même pour les substitutions σ et σ' : $(\sigma, \sigma') \vdash P \approx P$ (resp. $(\sigma, \sigma') \vdash P \approx' P$).*

Soient σ_1 et σ_2 deux substitutions équivalentes et Q_1, Q_2, R_1, R_2 des protocoles. Supposons que $(\sigma_1, \sigma_2) \vdash Q_1 \approx Q_2$ (resp. $(\sigma_1, \sigma_2) \vdash Q_1 \approx' Q_2$) et que $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$ (resp. $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$). Supposons de plus que pour i entier valant 1 ou 2, les protocoles Q_i et R_i sont σ_i-saufs. (resp. σ_i-saufs). Alors la relation $(\sigma_1, \sigma_2) \vdash Q_1|R_1 \approx Q_2|R_2$ (resp. $(\sigma_1, \sigma_2) \vdash Q_1|R_1 \approx' Q_2|R_2$) est vérifiée.*

Preuve. La preuve de cette proposition repose sur le même principe que la preuve de la proposition 5.7. Nous montrons chacune des deux propriétés pour la relation de bisimilarité faible $\approx$, la preuve étant identique pour $\approx'$.

Montrons tout d'abord la deuxième partie de la propriété. Considérons la relation binaire $\mathcal{R}$ définie de la manière suivante : $(\sigma_1, \sigma_2) \vdash (Q_1 \mid R_1)\mathcal{R}(Q_2 \mid R_2)$ si et seulement si :

1. $(\sigma_1, \sigma_2) \vdash Q_1 \approx Q_2$, $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$, et
2. pour i entier tel que $i = 1, 2$, les protocoles Q_i et R_i sont σ_i-saufs.

Montrons que $\mathcal{R}$ est une bisimulation faible. Supposons que : $(\sigma_1, \sigma_2) \vdash (Q_1 \mid R_1)\mathcal{R}(Q_2 \mid R_2)$. Nous considérons uniquement le cas le plus délicat, à savoir lorsque les protocoles Q_1 et R_1 interagissent entre eux. Supposons donc que :

$$\sigma_1 \triangleright Q_1 \mid R_1 \xmapsto[-]{\tau} \sigma_1 \triangleright (\nu\tilde{h})(Q'_1 \mid R'_1), \tag{5.10}$$

avec $Q_1 \xrightarrow{(\nu\tilde{h})\bar{a}<M>} Q'_1$ et $R_1 \xrightarrow{aM} R'_1$ (les autres cas sont symétriques). Comme Q_1 est σ_1-sauf, les messages a, M vérifient $a, M \in kn(\sigma_1)$, donc $a = \widehat{\eta\sigma_1}$ pour une certaine expression publique η (proposition 5.2). En fait, comme les messages M et a sont dans la connaissance de σ_1, la suite $\tilde{h}$ est nécessairement vide. On en déduit la transition : $\sigma_1 \triangleright Q_1 \xmapsto[\eta(x)]{(\nu\tilde{h})\bar{a}<M>} \sigma'_1 \triangleright Q'_1$ avec $\sigma'_1 = \sigma_1[M/x]$. De plus, la relation $(\sigma_1, \sigma_2) \vdash Q_1 \approx Q_2$ est vérifiée par hypothèse, on en déduit que : $\sigma_2 \triangleright Q_2 \xmapsto[\eta(x)]{(\nu\tilde{k})\bar{a'}<M'>} \sigma'_2 \triangleright Q'_2$, pour certains messages a' ($a' = \widehat{\eta\sigma_2}$) et M' et pour un certain protocole Q'_2, tel que $(\sigma'_1, \sigma'_2) \vdash Q'_1 \approx Q'_2$. De même, $\tilde{k}$ est nécessairement vide et $\sigma'_2 = \sigma_2[M'/x]$ car $a', M' \in kn(\sigma_2)$.

Maintenant, comme $M \in kn(\sigma_1)$, il existe une expression publique ζ telle que $n(\zeta) \subseteq dom(\sigma_1)$ et $M = \widehat{\zeta\sigma_1}$ (proposition 5.2). D'autre part, comme $\sigma'_1 = \sigma_1[M/x]$ et $M \in kn(\sigma_1)$, on déduit que $kn(\sigma'_1) = kn(\sigma_1)$ donc les applications $\overline{\cdot}^{\sigma'_1}$ et $\overline{\cdot}^{\sigma_1}$ sont identiques. De la même manière, $\overline{\cdot}^{\sigma'_2} = \overline{\cdot}^{\sigma_2}$. Enfin, les substitutions σ'_1 et σ'_2 sont équivalentes, donc $\overline{M}^{\sigma'_1}\theta = \overline{M'}^{\sigma'_2}$ pour une certaine substitution θ. On en déduit l'égalité : $\overline{M}^{\sigma_1}\theta (= \overline{M}^{\sigma'_1}\theta) = \overline{M'}^{\sigma_2}$. En appliquant le corollaire 5.1, il vient : $\overline{M}^{\sigma_1}\theta = \overline{\widehat{\zeta\sigma_1}}^{\sigma_1}\theta = \overline{\widehat{\zeta\sigma_2}}^{\sigma_2}$. Comme l'application $\overline{\cdot}^{\sigma_2}$ est injective, on déduit que $M' = \widehat{\zeta\sigma_2}$.

Considérons maintenant ce qui se passe pour le protocole R_1 : comme $R_1 \xrightarrow{aM} R'_1$, on peut effectuer la transition $\sigma_1 \triangleright R_1 \xmapsto[\overline{\eta}\zeta]{aM} \sigma_1 \triangleright R'_1$. De plus, comme $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$, on déduit $\sigma_2 \triangleright R_2 \stackrel{a''M''}{\underset{\overline{\eta}\zeta}{\Longmapsto}} \sigma_2 \triangleright R'_2$, avec $(\sigma_1, \sigma_2) \vdash R'_1 \approx R'_2$ et $a'' = \widehat{\eta\sigma_2}$, $M'' = \widehat{\zeta\sigma_2}$. D'après ce qui précède, $a' = a''$ et $M' = M''$. En combinant les relations $Q_2 \xRightarrow{\bar{a}'<M'>} Q_2'$ et $R_2 \xRightarrow{a'M'} R_2'$, on déduit $Q_2 \mid R_2 \Rightarrow Q_2' \mid R_2'$, et donc :

$$\sigma_2 \triangleright Q_2 \mid R_2 \underset{-}{\overset{\tau}{\longmapsto}} \sigma_2 \triangleright (Q_2' \mid R_2').$$

Il reste à vérifier $(\sigma_1, \sigma_2) \vdash Q_1' \mid R_1' \mathcal{R} Q_2' \mid R_2'$. Vérifions donc les conditions 1 et 2 de la définition de $\mathcal{R}$. De $(\sigma_1', \sigma_2') \vdash Q_1' \approx Q_2'$, on déduit par affaiblissement (proposition 5.8) que $(\sigma_1, \sigma_2) \vdash Q_1' \approx Q_2'$. On a déjà montré que $(\sigma_1, \sigma_2) \vdash R_1' \approx R_2'$ dons la première condition est vérifiée. D'autre part, les protocoles Q_i' et R_i' sont σ_i-saufs car les protocoles Q_i et R_i le sont, ce qui montre la deuxième condition.

Montrons maintenant la première partie de la propriété.

On considère la relation $\mathcal{R}$ définie par $(\sigma_1, \sigma_2) \vdash P\mathcal{R}Q$ si et seulement si $\sigma_1 \sim \sigma_2$ et il existe un protocole A tel que A est σ_1 et σ_2-sauf et tel que $P = A[\widehat{\zeta_i \sigma_1}/x_i]_{i \in J}$ et $Q = A[\widehat{\zeta_i \sigma_2}/x_i]_{i \in J}$.

On montre que $\mathcal{R}$ est une bisimulation faible en utilisant les mêmes techniques que pour la première partie de la preuve. □

Remarque : La définition de la σ-sécurité proposée par M. Boreale *et al.* ne permet en fait pas d'assurer la deuxième partie de cette proposition. En effet, rappelons la définition donnée dans [BDNP99] :

> un protocole R est σ-sauf si pour toute action s telle que $\sigma \triangleright R \underset{u}{\overset{s}{\Longmapsto}} \sigma' \triangleright R'$ $\underset{\eta(x)}{\overset{(\nu\tilde{b}\bar{a}<M>)}{\longmapsto}} R''$, alors $M \in kn(\sigma)$.

La différence avec la définition que nous proposons tient dans « $\eta(x)$ ». Intuitivement, un protocole R est σ-sauf pour cette nouvelle définition si, quel que soit le message M que le protocole peut envoyer sur un canal connu de l'environnement, ce message M est déjà connu de l'environnement. Mais ce protocole peut éventuellement se compromettre sur des canaux non connus de l'environnement.

Cela permet de construire un contre-exemple pour la propriété 5.9 (proposition 5.6 de [BDNP99]) si on considère la définition de la σ-sécurité rappelée ci-dessus.

On définit $\sigma_1 \stackrel{\text{def}}{=} \sigma_2 \stackrel{\text{def}}{=} [p/p]$. (Les environnements σ_1 et σ_2 ne connaisse qu'un canal public p.)

Soit Q_1 le protocole défini par $Q_1 \stackrel{\text{def}}{=} \bar{a} < a > .\bar{p} < a > .\mathbf{0}$: le protocole Q_1 commence par envoyer le message a sur le canal a puis envoie le message s sur le canal p. On considère également les protocoles suivants : $Q_2 \stackrel{\text{def}}{=} \mathbf{0}$, $R_1 \stackrel{\text{def}}{=} R_2 \stackrel{\text{def}}{=} a(x).\mathbf{0}$.

Alors R_1 est σ_1-sauf et Q_2, R_2 sont σ_2-saufs. De plus, la relation $(\sigma_1, \sigma_2) \vdash R_1 \approx R_2$ est vérifiée. D'autre part, comme il n'y a pas de transitions possibles pour la configuration $\sigma_1 \triangleright P_1$, le protocole Q_1 est σ_1-sauf et $(\sigma_1, \sigma_2) \vdash P_1 \approx P_2$.

Par contre, on ne peut composer ces protocoles en préservant la bisimilarité faible : $(\sigma_1, \sigma_2) \vdash Q_1 | R_1 \not\approx Q_2 | R_2$.

5.4.2 Exemple : le protocole de Needham-Schroeder-Lowe

Dans cette partie nous montrons que le protocole de Needham-Schroeder-Lowe préserve le secret du nonce du deuxième participant pour une propriété de secret exprimée sous forme d'équivalence observationnelle et pour un nombre arbitraire de sessions du protocole. Nous allons commencer par décrire le protocole en spi-calcul, puis la démonstration du secret se décompose en trois grandes étapes. Tout d'abord nous nous ramenons à la preuve de la bisimilarité faible restreinte en appliquant les théorèmes d'équivalence. Puis la partie principale de la démonstration consiste à générer un environnement dont la connaissance n'augmente plus quel que soit le déroulement du protocole. Enfin, on termine la preuve à l'aide des propositions démontrées au paragraphe 5.4.1.

5.4.2.1 Présentation du protocole

Nous rappelons la description informelle du protocole de Needham-Schroeder-Lowe à clefs publiques :

$$\begin{array}{lll} A & \to B: & \{A, N_a\}_{\mathsf{pub}(B)} \\ B & \to A: & \{N_a, N_b, B\}_{\mathsf{pub}(A)} \\ A & \to B: & \{N_b\}_{\mathsf{pub}(B)} \end{array}$$

Nous considérons une version du protocole avec deux participants honnêtes a et b et un participant malhonnête i. Nous montrons au chapitre 6 que pour une classe très large de protocoles et pour des propriétés de secret exprimées sous forme d'accessibilité, deux agents suffisent pour trouver une attaque (l'un honnête, l'autre malhonnête). Aussi, nous pensons que se limiter à deux agents honnêtes et un malhonnête n'est pas une réelle restriction. Nous aurions même pu nous limiter à un agent honnête et un malhonnête mais nous avons choisi la première solution pour garder une présentation plus lisible.

Le protocole associé au protocole de Needham-Schroeder-Lowe se décompose en plusieurs protocoles représentant chacun le rôle d'un des participants. Ainsi, nous définissons :

$$\begin{array}{rcl} I_{A,B}(n) & = & \overline{p}(\{n, A\}_{\mathsf{pub}(B)}).p(x).\, \mathsf{let}\, (x_{n_1}, x_{n_2}, x_2) = \mathsf{dec}_{\mathsf{prv}(A)}(x)\, \mathsf{in} \\ & & \qquad [x_{n_1} = n][x_2 = B]\, \overline{p}(\{x_{n_2}\}_{\mathsf{pub}(B)}).\mathbf{0} \\ R_{B,A}(n) & = & p(x).\, \mathsf{let}\, (x_{n_1}, x_1) = \mathsf{dec}_{\mathsf{prv}(B)}(x)\, \mathsf{in}\, [x_1 = A]\, \overline{p}(\{x_{n_1}, n, B\}_{\mathsf{pub}(A)}).\mathbf{0} \end{array}$$

Le protocole $I_{a,i}(n)$ décrit le rôle de a en tant qu'initiateur du protocole, parlant à l'intrus i pour une session où le nonce n va être engendré par a. Ainsi le protocole commence par envoyer le message $\{n, a\}_{\mathsf{pub}(i)}$ puis attend un message de la forme $\{n, m, i\}_{\mathsf{pub}(a)}$ où m est un message quelconque et répond par le message $\{m\}_{\mathsf{pub}(b)}$. Le protocole $R_{b,a}(n)$ décrit le rôle de b répondant à a pour une session où b engendre le nonce n. Ainsi le protocole $R_{b,a}(n)$ attend un message de la forme $\{a, m\}_{\mathsf{pub}(b)}$ où m est un message quelconque et répond par le message $\{m, n, b\}_{\mathsf{pub}(a)}$.

Le protocole associé au protocole de Needham-Schroeder-Lowe s'obtient par composition des réplications de protocoles correspondant à chacun des rôles pour chaque identité :

$$\begin{array}{rcl} P(n_b) & = & !(\nu n_1^{ab}) I_{a,b}(n_1^{ab}) \;|!(\nu n_1^{ai}) I_{a,i}(n_1^{ai}) \;|!(\nu n_1^{ba}) I_{b,a}(n_1^{ba}) \;|!(\nu n_1^{bi}) I_{b,i}(n_1^{bi}) \\ & & |\; R_{b,a}(n_b) \;|!(\nu n_2^{ab}) R_{a,b}(n_2^{ab}) \;|!(\nu n_2^{ai}) R_{a,i}(n_2^{ai}) \;|!(\nu n_2^{ba}) R_{b,a}(n_2^{ba}) \;|!(\nu n_2^{bi}) R_{b,i}(n_2^{bi}) \\ & & |\; \overline{p}(\mathsf{prv}(i)).\mathbf{0} \end{array}$$

Le protocole $P(n_b)$ représente le protocole de Needham-Schroeder-Lowe avec un nombre arbitraire de sessions et trois participants a, b et i. De plus, on a ajouté un processus qui révèle la clef privée de i. C'est pourquoi ce dernier sera considéré comme malhonnête. C'est aussi pourquoi nous ne décrivons pas les protocoles associés à i parlant à a ou b : ces protocoles peuvent être représentés par des processus. Intuitivement, cela correspond au fait que le rôle de i peut être simulé par l'intrus. Enfin, nous avons particularisé un des protocoles correspondant au rôle de b répondant à a : il s'agit du protocole $R_{b,a}(n_b)$. Le but de cette partie est de montrer que ce nonce reste secret, c'est-à-dire que pour tous messages M et M', les protocoles $(\nu n)P(< M, n > /n_b)$ et $P(\nu n)(< M', n > /n_b)$ sont en équivalence barbue : $P(\nu n)(< M, n > /n_b) \cong P(\nu n)(< M', n > /n_b)$.

Notation : Dans la suite, on écrira $P(t)$ au lieu de $P(t/n_b)$.

5.4.2.2 Application des théorèmes d'équivalence

Tout d'abord, on a vu au paragraphe 5.3.1 que les protocoles $(\nu n)P(< M, n >)$ et $(\nu n)P(< M', n >)$ sont en équivalence barbue si et seulement si : $(\epsilon_V, \epsilon_V) \vdash (\nu n)P(< M, n >) \cong (\nu n)P(< M', n >)$ où V contient les noms libres de $(\nu n)P(< M, n >)$ et $P(\nu n)P(< M', n >)$. On choisit $V = \{a, b, i\} \cup \mathit{fn}(M) \cup \mathit{fn}(M')$.

On commence par supprimer les réplications des protocoles $(\nu n)P(< M, n >)$ et $(\nu n)P(< M', n >)$. En utilisant les mêmes techniques de preuve que pour les propriétés 5.7 et 5.9, on montre facilement qu'il suffit de prouver que pour tout entier q strictement positif, la relation $(\epsilon_V, \epsilon_V) \vdash (\nu n)P_q(< M, n >) \cong (\nu n)P_q(< M', n >)$ est vérifiée où P_q est le protocole P dans lequel les processus avec réplication sont répliqués exactement q fois.

$$\begin{array}{rcl} P_q(M) & \stackrel{\text{def}}{=} & R_{ba}(M) \mid \overline{p}(\mathsf{prv}(i)).\mathbf{0} \\ & & \mid (\nu n^{ab}_{1,1}) I_{a,b}(n^{ab}_{1,1}) \mid \cdots \mid (\nu n^{ab}_{1,q}) I_{a,b}(n^{ab}_{1,q}) \mid (\nu n^{ai}_{1,1}) I_{a,i}(n^{ai}_{1,1}) \mid \cdots \mid (\nu n^{ai}_{1,q}) I_{a,i}(n^{ai}_{1,q}) \\ & & \mid (\nu n^{ba}_{1,1}) I_{b,a}(n^{ba}_{1,1}) \mid \cdots \mid (\nu n^{ba}_{1,q}) I_{b,a}(n^{ba}_{1,q}) \mid (\nu n^{bi}_{1,1}) I_{b,i}(n^{bi}_{1,1}) \mid \cdots \mid (\nu n^{bi}_{1,q}) I_{b,i}(n^{bi}_{1,q}) \\ & & \mid (\nu n^{ab}_{2,1}) R_{a,b}(n^{ab}_{2,1}) \mid \cdots \mid (\nu n^{ab}_{2,q}) R_{a,b}(n^{ab}_{2,q}) \mid (\nu n^{ai}_{2,1}) R_{a,i}(n^{ai}_{2,1}) \mid \cdots \mid (\nu n^{ai}_{2,q}) R_{a,i}(n^{ai}_{2,q}) \\ & & \mid (\nu n^{ba}_{2,1}) R_{b,a}(n^{ba}_{2,1}) \mid \cdots \mid (\nu n^{ba}_{2,q}) R_{b,a}(n^{ba}_{2,q}) \mid (\nu n^{bi}_{2,1}) R_{b,i}(n^{bi}_{2,1}) \mid \cdots \mid (\nu n^{bi}_{2,q}) R_{b,i}(n^{bi}_{2,q}) \end{array}$$

D'après le théorème de complétude 5.3, il suffit de montrer que, pour tout entier q, la relation $(\epsilon_V, \epsilon_V) \vdash (\nu n)P_q(< M, n >) \approx (\nu n)P_q(< M', n >)$ est vérifiée. On vérifie en effet que ϵ_V est de domaine fini et que $(\nu n)P_q(< M, n >)$ et $(\nu n)P_q(< M', n >)$ sont d'image structurellement finie.

En appliquant maintenant la proposition 5.7, il suffit de montrer que les protocoles sont en bisimilarité faible restreinte : il suffit de montrer que pour tout entier q, pour toutes suites finies de noms $\tilde{h}$ et $\tilde{k}$ telles que $|\tilde{h}| = |\tilde{k}|$ et $\tilde{h} \cap dom(\sigma_1) = \tilde{k} \cap dom(\sigma_2) = \emptyset$, la relation $(\epsilon_V[\tilde{h}/\tilde{h}], \epsilon_V[\tilde{k}/\tilde{k}]) \vdash (\nu n)P_q(< M, n >) \approx' (\nu n)P_q(< M', n >)$ est vérifiée.

5.4.2.3 Génération d'un invariant

La principale étape de la preuve est de trouver un environnement σ_M contenant $\epsilon_V[\tilde{h}/\tilde{h}]$ tel que chacune des composantes du protocole $(\nu n)P_q(< M, n >)$ soit σ-sauf*, c'est-à-dire tel que la connaissance de l'environnement n'augmente plus. Nous pourrons ensuite le recomposer à l'aide de la proposition 5.9. On commence par donner à l'environnement l'ensemble des messages qu'il pourrait obtenir par un jeu honnête des protocoles. On considère :

$$\begin{aligned}
S_{M,\tilde{h}} \stackrel{\text{def}}{=} V \quad \cup \quad \tilde{h} \cup & \bigcup_{\substack{1 \leqslant m \leqslant q,\\ (A,B) \in \{(a,b),(b,a)\}}} \{n^{Ai}_{1,m}, n^{Ai}_{2,m}, \{n^{AB}_{1,m}, A\}_{\mathsf{pub}(B)}, \{n^{BA}_{2,m}\}_{\mathsf{pub}(B)}, \{< M, n >\}_{\mathsf{pub}(B)}\} \\
\cup \quad & \bigcup_{\substack{1 \leqslant m, m' \leqslant q,\\ (A,B) \in \{(a,b),(b,a)\}}} \{n^{AB}_{1,m}, n^{BA}_{2,m'}, B\}_{\mathsf{pub}(A)}, \{n^{Ai}_{1,m}, n^{BA}_{2,m'}, B\}_{\mathsf{pub}(A)}\} \\
\cup \quad & \bigcup_{\substack{1 \leqslant m \leqslant q,\\ (A,B) \in \{(a,b),(b,a)\}}} \{\{n^{AB}_{1,m}, < M, n >, B\}_{\mathsf{pub}(A)}, \{n^{Ai}_{1,m}, < M, n >, B\}_{\mathsf{pub}(A)}\}
\end{aligned}$$

et on pose $\sigma_{M,\tilde{h}} = \epsilon_{S_{M,\tilde{h}}}$. Un tel environnement ne connaît encore pas assez de messages. En effet, pour tout message t, l'environnement peut construire le message $\{t, a\}_{\mathsf{pub}(b)}$ auquel b va répondre par $\{t, n^{ab}_m, b\}_{\mathsf{pub}(a)}$ pour un certain entier m tel que $m \leqslant q$ et la connaissance de l'environnement augmente. On définit alors une fonction sur les messages F_M qui simule les réponses possibles de b ou a jouant le rôle du deuxième participant.

$$\begin{aligned}
F_M(S) \quad \stackrel{\text{def}}{=} \quad & \bigcup_{1 \leqslant m \leqslant q} \left\{\{t, n^{ba}_{2,m}, b\}_{\mathsf{pub}(a)}, \{t, M, b\}_{\mathsf{pub}(a)} \mid \{t, a\}_{\mathsf{pub}(b)} \in kn(S)\right\} \\
& \cup \bigcup_{1 \leqslant m \leqslant q} \left\{\{t, n^{ab}_{2,m}, a\}_{\mathsf{pub}(b)}, \{t, M, a\}_{\mathsf{pub}(b)} \mid \{t, b\}_{\mathsf{pub}(a)} \in kn(S)\right\}.
\end{aligned}$$

On considère alors la fonction sur les messages $\overline{F_M}$ qui associe à S le plus petit point fixe de F_M contenant S : $\overline{F_M}(S) = \cup_{i \geqslant 0} F^i_M(S)$ où F^i_M désigne l'application de F_M répétée i fois. On construit la substitution $\sigma'_{M,\tilde{h}}$ de la manière suivante :

$$\sigma'_{M,\tilde{h}} \stackrel{\text{def}}{=} [t/x_t]_{t \in \overline{F_M}(range(\sigma_{M,\tilde{h}}))}.$$

On constate alors que $range(\sigma'_{M,\tilde{h}}) = range(\overline{F_M}(range(\sigma_{M,\tilde{h}})))$ et $range(\overline{F_M}(range(\sigma_{M',\tilde{k}}))) = range(\overline{F_M}(range(\sigma_{M,\tilde{h}})))[M \mapsto M', h_i \mapsto k_i]_{1 \leqslant i \leqslant |\tilde{h}|}$. On pose :

$$\sigma'_{M',\tilde{k}} \stackrel{\text{def}}{=} [t[M \mapsto M', h_i \mapsto k_i]_{1 \leqslant i \leqslant |\tilde{h}|}/x_t]_{t \in \overline{F_M}(range(\sigma_{M,\tilde{h}}))}.$$

Ces nouveaux environnements (infinis) conviennent : chacune des composantes des protocoles $(\nu n)P_q(< M, n >)$ et $(\nu n)P_q(< M', n >)$ est $\sigma'_{M,\tilde{h}}$ et/ou $\sigma'_{M',\tilde{k}}$-sauve*.

Lemme 5.6 *Soient m un entier tel que $\forall 1 \leqslant m \leqslant q$, et A et B des noms pris dans l'ensemble $\{a, b\}$ tels que $A \neq B$. Les protocoles $I_{A,B}(n^{AB}_{1,m})$, $I_{A,i}(n^{Ai}_{1,m})$, $R_{A,B}(n^{AB}_{2,m})$ et $R_{A,i}(n^{Ai}_{2,m})$ sont $\sigma'_{M,\tilde{h}}$ et $\sigma'_{M',\tilde{k}}$-saufs*.*
De plus, le protocole $R_{b,a}(M)$ est $\sigma'_{M,\tilde{h}}$-sauf et le protocole $R_{b,a}(M')$ est $\sigma'_{M',\tilde{k}}$-sauf*.*

En outre, les substitutions $\sigma'_{M,\tilde{h}}$ et $\sigma'_{M',\tilde{k}}$ sont équivalentes.

Lemme 5.7 $\sigma'_{M,\tilde{h}} \sim \sigma'_{M',\tilde{k}}$.

Enfin, les protocoles élémentaires $R_{b,a}(< M, n >)$ et $R_{b,a}(< M', n >)$ sont en bisimilarité faible restreinte.

Lemme 5.8 $(\sigma'_{M,\tilde{h}}, \sigma'_{M',\tilde{k}}) \vdash R_{b,a}(< M, n >) \approx' R_{b,a}(< M', n >)$.

Afin de ne pas perdre le fil de la démonstration, nous repoussons la preuve de ces lemmes au dernier paragraphe de cette partie.

5.4.2.4 Fin de la preuve

Soient m un entier tel que $1 \leqslant m \leqslant q$, et A et B des noms pris dans l'ensemble $\{a, b\}$ tels que $A \neq B$. En utilisant les lemmes 5.6 et 5.7 et la première partie de la proposition 5.9, on en déduit que la relation

$$(\sigma'_{M,\tilde{h}}, \sigma'_{M',\tilde{k}}) \vdash I_{A,B}(n^{AB}_{1,m}) \approx' I_{A,B}(n^{AB}_{1,m})$$

est vérifiée ainsi que les relations correspondantes pour les protocoles $I_{A,i}(n^{Ai}_{1,m})$, $R_{A,B}(n^{AB}_{2,m})$ et $R_{A,i}(n^{Ai}_{2,m})$. En utilisant les lemmes 5.8 et 5.7 et la deuxième partie de la proposition 5.9, nous pouvons recomposer les protocoles $P_q(< M, n >)$ et $P_q(< M', n >)$:

$$\begin{aligned}(\sigma'_{M,\tilde{h}}, \sigma'_{M',\tilde{k}}) \vdash R_{b,a}(< M, n >) \mid \mathop{\Big|}_{\substack{1 \leqslant m \leqslant q \\ (A,B)=(a,b) \\ (A,B)=(b,a)}} & I_{A,B}(n^{AB}_{1,m}) \mid I_{A,i}(n^{Ai}_{1,m}) \mid R_{A,B}(n^{AB}_{2,m}) \mid R_{A,i}(n^{Ai}_{2,m}) \\ \approx' R_{b,a}(< M', n >) \mid \mathop{\Big|}_{\substack{1 \leqslant m \leqslant q \\ (A,B)=(a,b) \\ (A,B)=(b,a)}} & I_{A,B}(n^{AB}_{1,m}) \mid I_{A,i}(n^{Ai}_{1,m}) \mid R_{A,B}(n^{AB}_{2,m}) \mid R_{A,i}(n^{Ai}_{2,m}).\end{aligned}$$

En appliquant la propriété d'*affaiblissement* de la proposition 5.8, il vient :

$$\begin{aligned}(\epsilon_V[\tilde{h}/\tilde{h}], \epsilon_V[\tilde{k}/\tilde{k}]) \vdash R_{b,a}(<M, n>) \mid \mathop{\Big|}_{\substack{1 \leqslant m \leqslant q \\ (A,B)=(a,b) \\ (A,B)=(b,a)}} & I_{A,B}(n^{AB}_{1,m}) \mid I_{A,i}(n^{Ai}_{1,m}) \mid R_{A,B}(n^{AB}_{2,m}) \mid R_{A,i}(n^{Ai}_{2,m}) \\ \approx' R_{b,a}(< M', n >) \mid \mathop{\Big|}_{\substack{1 \leqslant m \leqslant q \\ (A,B)=(a,b) \\ (A,B)=(b,a)}} & I_{A,B}(n^{AB}_{1,m}) \mid I_{A,i}(n^{Ai}_{1,m}) \mid R_{A,B}(n^{AB}_{2,m}) \mid R_{A,i}(n^{Ai}_{2,m}).\end{aligned}$$

Enfin, en appliquant la propriété de *restriction* de la proposition 5.8, il vient la relation attendue : $(\epsilon_V, \epsilon_V) \vdash P_q(< M, n >) \approx' P_q(< M', n >)$, ce qui termine la démonstration.

5.4.2.5 Preuve des lemmes

Il reste à vérifier les lemmes 5.6, 5.7 et 5.8.

Remarquons tout d'abord que les clefs privées de a et b ($\mathsf{prv}(a)$ et $\mathsf{prv}(b)$) ne sont pas dans la connaissance des substitutions $\sigma'_{M,\tilde{h}}$ et $\sigma'_{M',\tilde{k}}$. En effet, les messages $\mathsf{prv}(a)$ et $\mathsf{prv}(b)$ n'apparaissent jamais comme sous-termes de messages de $\sigma'_{M,\tilde{h}}$ ou $\sigma'_{M',\tilde{k}}$ et le symbole de fonction prv est un symbole de fonction privé.

Nous pouvons maintenant faire la preuve du lemme 5.7. Posons $\theta \stackrel{\text{def}}{=} [M \mapsto M', h_i \mapsto k_i]_{1 \leqslant i \leqslant |\tilde{h}|}$. On rappelle que $\sigma'_{M,\tilde{h}} = [t/x_t]_{t \in \overline{F_M}(range(\sigma_{M,\tilde{h}}))}$ et $\sigma'_{M',\tilde{k}} = [t\theta/x_t]_{t \in \overline{F_M}(range(\sigma_{M,\tilde{h}}))}$. Comme $\mathsf{prv}(a)$ et $\mathsf{prv}(b)$ ne sont pas dans la connaissance de $kn(\sigma'_{M,\tilde{h}})$ ni de $kn(\sigma'_{M,\tilde{h}})$ et que les messages des deux substitutions sont des noms ou des messages chiffrés avec $\mathsf{prv}(a)$ ou $\mathsf{prv}(b)$, on déduit que $\overline{\sigma'_{M,\tilde{h}}} = [X_t/x_t]_{t\in\overline{F_M}(range(\sigma_{M,\tilde{h}}))}$ et $\overline{\sigma'_{M',\tilde{k}}} = [X_{t\theta}/x_t]_{t\in\overline{F_M}(range(\sigma_{M,\tilde{h}}))}$. On pose $\theta'(X_t) = X_{t\theta}$, on obtient alors $\overline{\sigma'_{M,\tilde{h}}}\theta' = \overline{\sigma'_{M',\tilde{k}}}$ donc d'après le théorème 5.1, les deux substitutions sont équivalentes.

Nous allons maintenant faire la preuve du lemme 5.6. Pour cela la proposition suivante nous sera utile.

Proposition 5.10 *Soit $\{t\}_k$ un message chiffré tel que $\{t\}_k \in kn(\sigma)$, avec $\sigma = \sigma'_{t,\tilde{h}}$ avec $\sigma'_{t',\tilde{k}}$. Alors ou bien $\{t\}_k \in \sigma$ ou bien $t \in kn(\sigma)$.*

Preuve. En effet, comme $\{t\}_k \in kn(\sigma)$, d'après la proposition A.1, il existe une expression publique telle que : $\{t\}_k = \widehat{\zeta\sigma}$, d'où $\overline{\{t\}_k} = \overline{\widehat{\zeta\sigma}} = \widehat{\zeta\overline{\sigma}}$. Le cas $\overline{\{t\}_k} = X_{\{t\}_k}$ correspond au cas $\{t\}_k \in kn(\sigma)$ et le cas $\overline{\{t\}_k} = \{\overline{t}\}_{\overline{k}}$ ou $\overline{\{t\}_k} = \{\overline{t}\}_{(\overline{k'})^{-1}}$ correspond au cas $t \in kn(\sigma)$. □

Lemme (lemme 5.6) *Soient m un entier tel que $1 \leqslant m \leqslant q$, et A et B des noms pris dans l'ensemble $\{a, b\}$ tels que $A \neq B$. Les protocoles $I_{A,B}(n^{AB}_{1,m})$, $I_{A,i}(n^{Ai}_{1,m})$, $R_{A,B}(n^{AB}_{2,m})$ et $R_{A,i}(n^{Ai}_{2,m})$ sont $\sigma'_{M,\tilde{h}}$ et $\sigma'_{M',\tilde{k}}$-saufs*.*
De plus, le protocole $R_{b,a}(< M, n >)$ est $\sigma'_{M,\tilde{h}}$-sauf et le protocole $R_{b,a}(< M', n >)$ est $\sigma'_{M',\tilde{k}}$-sauf*.*

Preuve. Nous montrons seulement que $I_{a,b}(n^{ab}_{1,m})$ et $I_{a,i}(n^{ai}_{1,m})$ sont $\sigma'_{M,\tilde{h}}$-saufs* car les autres cas sont similaires ou plus simples.

Supposons : $\sigma'_{M,\tilde{h}} \triangleright I_{a,b}(n^{ab}_{1,m}) \Longrightarrow\xmapsto{\bar{p}<N>}$. Montrons que $N \in kn(\sigma'_{M,\tilde{h}})$. Ou bien $N = \{n^{ab}_{1,m}, a\}_{\mathsf{pub}(b)}$ et alors $N \in \sigma_{M,\tilde{h}}$, ce qui implique : $N \in kn(\sigma'_{M,\tilde{h}})$. Ou bien N est de la forme $N = \{N'\}_{\mathsf{pub}(b)}$ avec $\{n^{ab}_{1,m}, N', b\}_{\mathsf{pub}(a)} \in kn(\sigma'_{M,\tilde{h}})$. En appliquant la proposition 5.10, il vient que : $\{n^{ab}_{1,m}, N', b\}_{\mathsf{pub}(a)} \in \sigma'_{M,\tilde{h}}$ ou $N' \in kn(\sigma'_{M,\tilde{h}})$.

- Si $\{n^{ab}_{1,m}, N', b\}_{\mathsf{pub}(a)} \in \sigma'_{M,\tilde{h}}$, alors par construction de $\sigma'_{M,\tilde{h}}$, le message N' vérifie $N' = n^{ba}_{2,m'}$ pour un certain entier m' tel que $1 \leqslant m' \leqslant q$. Comme $\{n^{ba}_{2,m'}\}_{\mathsf{pub}(b)} \in \sigma_{M,\tilde{h}}$, il vient $N \in kn(\sigma'_{M,\tilde{h}})$.
- Si $N' \in kn(\sigma'_{M,\tilde{h}})$, alors $N = \{N'\}_{\mathsf{pub}(b)} \in kn(\sigma'_{M,\tilde{h}})$.

Nous pouvons conclure dans tous les cas que $I_{a,b}(n^{ab}_{1,m})$ est $\sigma'_{M,\tilde{h}}$-sauf*.

Supposons maintenant que : $\sigma'_{M,\tilde{h}} \triangleright I_{a,i}(n^{ai}_{1,m}) \Longrightarrow\xmapsto{\bar{p}<N>}$. Montrons que : $N \in kn(\sigma'_{M,\tilde{h}})$. Ou bien $N = \{n^{ai}_{1,m}, a\}_{\mathsf{pub}(i)}$ et alors $N \in \sigma_{M,\tilde{h}}$, ce qui implique $N \in kn(\sigma'_{M,\tilde{h}})$. Ou alors N est de la forme $N = \{N'\}_{\mathsf{pub}(i)}$ avec $\{n^{ai}_{1,m}, N', i\}_{\mathsf{pub}(a)} \in kn(\sigma'_{M,\tilde{h}})$. En appliquant la proposition 5.10, il vient que : $\{n^{ai}_{1,m}, N', i\}_{\mathsf{pub}(a)} \in \sigma'_{M,\tilde{h}}$ ou $N' \in kn(\sigma'_{M,\tilde{h}})$.

- Le cas $\{n^{ai}_{1,m}, N', i\}_{\mathsf{pub}(a)} \in \sigma'_{M,\tilde{h}}$ est impossible par construction de $\sigma'_{M,\tilde{h}}$.
- Si $N' \in kn(\sigma'_{M,\tilde{h}})$ alors $N = \{N'\}_{\mathsf{pub}(i)} \in kn(\sigma'_{M,\tilde{h}})$.

Nous pouvons conclure dans tous les cas que $I_{a,i}(n^{ai}_{1,m})$ est $\sigma'_{M,\tilde{h}}$-sauf*. □

Il reste à montrer que $R_{b,a}(< M, n >)$ et $R_{b,a}(< M', n >)$ sont en bisimilarité faible restreinte pour $\sigma'_{M,\tilde{h}}$ et $\sigma'_{M',\tilde{k}}$.

Lemme (lemme 5.8) $(\sigma'_{M,\tilde{h}}, \sigma'_{M',\tilde{k}}) \vdash R_{b,a}(< M, n >) \approx' R_{b,a}(< M', n >)$.

Preuve. La seule transition possible pour $\sigma'_{M,\tilde{h}} \triangleright R_{b,a}(< M, n >)$ est :

$$\sigma'_{M,\tilde{h}} \triangleright R_{b,a}(< M, n >) \xmapsto[\bar{p}<\zeta>]{p(N_1)} \sigma'_{M,\tilde{h}} \triangleright R'_{b,a}(< M, n >)[N_1/x]$$

avec $N_1 = \widehat{\zeta\sigma'_{M,\tilde{h}}}$ et $R'_{b,a}(M) \stackrel{\text{def}}{=} \mathsf{let}\,(x_{n_a}, x_a) = \mathsf{dec}_{\mathsf{prv}b}(x)\,\mathsf{in}\,[x_a = a]\,\overline{p}(\{x_{n_a}, M, b\}_{\mathsf{pub}(a)}).\mathbf{0}$. Alors

$$\sigma'_{M',\tilde{k}} \triangleright R_{b,a}(< M', n >) \xmapsto[\bar{p}<\zeta>]{p(N_2)} \sigma'_{M',\tilde{k}} \triangleright R'_{b,a}(< M', n >)[N_2/x]$$

avec $N_2 = \widehat{\zeta\sigma'_{M',\tilde{k}}}$. Montrons que : $(\sigma'_{M,\tilde{h}}, \sigma'_{M',\tilde{k}}) \vdash R'_{b,a}(<M, n>)[N_1/x] \approx' R'_{b,a}(<M', n>)[N_2/x]$. La seule transition possible pour $\sigma'_{M,\tilde{h}} \triangleright R'_{b,a}(< M, n >)[N_1/x]$ est :

$$\sigma'_{M,\tilde{h}} \triangleright R'_{b,a}(< M, n >)[N_1/x] \xmapsto[p(z)]{\bar{p}<\{N'_1, <M,n>, b\}_{\mathsf{pub}(a)}>} \sigma'_{M,\tilde{h}}[\{N'_1, < M, n >, b\}_{\mathsf{pub}(a)}/z] \triangleright \mathbf{0}$$

pour N'_1 vérifiant $N_1 = \{N'_1, a\}_{\mathsf{pub}(b)}$. Considérons $\overline{N_1}^{\sigma'_{M,\tilde{h}}}$. Le corollaire 5.1 assure que si σ et σ' sont deux substitutions telles que $\sigma \sim' \sigma'$ et θ une fonction de renommage telle que $\overline{\sigma}^{\sigma}\theta = \overline{\sigma'}^{\sigma'}$ alors : $\overline{\widehat{\zeta\sigma}}^{\sigma}\ \theta = \overline{\widehat{\zeta\sigma'}}^{\sigma'}$. Comme $N_1 = \widehat{\zeta\sigma'_{M,\tilde{h}}}$, il vient : $\overline{N_1}^{\sigma'_{M,\tilde{h}}}\theta = \overline{N_2}^{\sigma'_{M',\tilde{k}}}$. Il nous faut considérer deux cas.

- Si $\overline{N_1}^{\sigma'_{M,\tilde{h}}} = X_{N_1}$, alors $N_1 \in \sigma'_{M,\tilde{h}}$, d'où $N'_1 = n^{ab}_{1,m}$ pour un certain entier m tel que $1 \leqslant m \leqslant q$. Les égalités suivantes sont vérifiées : $\overline{N_2}^{\sigma'_{M',\tilde{k}}} = X_{N_1}\theta = X_{\{n^{ab}_{1,m}, a\}_{\mathsf{pub}(b)}}\theta = X_{\{n^{ab}_{1,m}, a\}_{\mathsf{pub}(b)}}$. On en déduit que : $N_2 = \{n^{ab}_{1,m}, a\}_{\mathsf{pub}(b)}$ et

$$\sigma'_{M',\tilde{k}} \triangleright R'_{b,a}(< M', n >)[N_2/x] \xmapsto[p(z)]{\bar{p}<\{n^{ab}_{1,m}, <M',n>, b\}_{\mathsf{pub}(a)}>} \sigma'_{M',\tilde{k}}[\{n^{ab}_{1,m}, < M', n >, b\}_{\mathsf{pub}(a)}/z] \triangleright \mathbf{0}.$$

  Comme $\sigma'_{M,\tilde{h}}[\{n^{ab}_{1,m}, < M, n >, b\}_{\mathsf{pub}(a)}/z] \sim \sigma'_{M',\tilde{k}}[\{n^{ab}_{1,m}, < M', n >, b\}_{\mathsf{pub}(a)}/z]$, la relation $(\sigma'_{M,\tilde{h}}[\{n^{ab}_{1,m}, < M, n >, b\}_{\mathsf{pub}(a)}/z], \sigma'_{M',\tilde{k}}[\{n^{ab}_{1,m}, < M', n >, b\}_{\mathsf{pub}(a)}/z]) \vdash \mathbf{0} \approx' \mathbf{0}$ est vérifiée.
- Si $\overline{N_1}^{\sigma'_{M,\tilde{h}}} = \{\overline{N'_1}, a\}_{\mathsf{pub}(a)}$, alors $\overline{N_2}^{\sigma'_{M',\tilde{k}}} = \{\overline{N'_1}\theta, a\}_{\mathsf{pub}(b)}$, donc $N_2 = \{N'_2, a\}_{\mathsf{pub}(b)}$ pour un certain message N'_2 tel que : $\overline{N'_2} = \overline{N'_1}\theta$. On en déduit :

$$\sigma'_{M',\tilde{k}} \triangleright R'_{b,a}(< M', n >)[N_2/x] \xmapsto[p(z)]{\bar{p}<\{N'_2, <M',n>, b\}_{\mathsf{pub}(a)}>} \sigma'_{M',\tilde{k}}[\{N'_2, < M', n >, b\}_{\mathsf{pub}(a)}/z] \triangleright \mathbf{0}.$$

  Les substitutions $\sigma'_{M,\tilde{h}}[\{N'_1, < M, n >, b\}_{\mathsf{pub}(a)}/z]$ et $\sigma'_{M',\tilde{k}}[\{N'_2, < M', n >, b\}_{\mathsf{pub}(a)}/z]$ restent équivalentes donc la relation $(\sigma'_{M,\tilde{h}}[\{N'_1, < M, n >, b\}_{\mathsf{pub}(a)}/z], \sigma'_{M',\tilde{k}}[\{N'_2, < M', n >, b\}_{\mathsf{pub}(a)}/z]) \vdash \mathbf{0} \approx' \mathbf{0}$ est vérifiée. $\square$

5.5 Comparaisons

Les résultats présentés ici sont une extension des travaux de M. Boreale *et al.* [BDNP99]. D'autre part, ils sont assez proches de ceux de M. Abadi et C. Fournet [AF01]. Nous allons comparer plus précisément nos résultats aux leurs ainsi qu'à d'autres travaux un peu plus éloignés.

5.5.1 Travaux de M. Boreale *et al.*

Le résultat d'équivalence entre l'équivalence observationnelle barbue et la bisimilarité faible est une généralisation importante des travaux de M. Boreale *et al.* [BDNP99] qui nous permet de traiter des protocoles cryptographiques plus réalistes.

- Nous traitons le cas du chiffrement à clefs publiques, qui est utilisé dans de très nombreux protocoles [CJ97].
- Nous permettons également le chiffrement à clefs composées, qui est utilisé dans beaucoup de protocoles réels comme SSL [MSS98]. Ajouter le chiffrement à clefs composées augmente considérablement la difficulté des techniques de décision : de nombreux résultats de décidabilité ne sont plus valables dès qu'on autorise le chiffrement à clefs composées [CS01].
- Nous avons ajouté un nombre arbitraire de symboles de fonctions avec différentes propriétés : inversibles ou non, publiques ou privés.
- Le spi-calcul est paramétré par un nombre arbitraire de prédicats dont la sémantique est un langage régulier. Ces prédicats permettent de modéliser qu'un agent peut reconnaître par exemple des clefs, des noms ou des messages hachés. Cela permet en particulier de typer les messages. À nouveau, permettre que les messages soient typés ou non a des conséquences importantes : certains résultats de décidabilité sont valables uniquement si l'on suppose que les messages sont fortement typés (aucune variable de type message) [Low98, CCM01]. M. Boreale n'autorise lui qu'un seul prédicat qui reconnaît exactement l'ensemble des noms.

Cette généralisation repose sur une nouvelle caractérisation de l'équivalence $\sim$ entre environnements à l'aide d'automates d'arbres. L'idée d'associer à un message le terme correspondant au message vu par l'intrus, avait déjà été introduite dans [AR00] mais dans un contexte moins général : pas de clefs composées, pas de tests d'égalité...

D'autre part, en partant de la notion de σ-sécurité introduite par M. Boreale, nous avons développé une méthode de preuve pour les protocoles avec réplication ce qui n'était pas le cas dans [BDNP99]. Plus généralement, c'est à notre connaissance la première fois que le secret du protocole de Needham-Schroeder-Lowe est vérifié dans le contexte du spi-calcul pour un nombre arbitraire de sessions.

5.5.2 Travaux de M. Abadi et C. Fournet

M. Abadi et C. Fournet ont montré l'équivalence entre l'équivalence barbue et une équivalence de traces similaire à la nôtre dans un contexte plus général que le nôtre. En effet, ils ajoutent au spi-calcul une théorie équationnelle arbitraire permettant en particulier de modéliser les propriétés algébriques des primitives cryptographiques. Dans notre modèle, la théorie équationnelle est implicitement fixée par les propriétés (inversibles ou non) des symboles fonctionnels. Le résultat d'équivalence entre équivalence barbue et bisimilarité faible peut donc être considéré comme une conséquence de leur propre résultat d'équivalence. Mais notre cadre moins général permet de préserver la décidabilité de l'équivalence $\sim$ entre environnements qui est le premier pas vers l'automatisation des preuves. En particulier, M. Abadi et C. Fournet ne développent aucun système de preuve pour l'équivalence barbue.

D'autre part, nous avons introduit la distinction entre symboles de fonctions publiques et privés. Dans [AF01], tous les symboles de fonctions sont publiques. Pour modéliser le chiffrement asymétrique, ils ont alors besoin d'introduire des nonces « frais » dans les symboles de fonctions prv et pub et d'envoyer à chaque fois explicitement les clefs publiques de chacun des agents. Un tel codage ne permet pas de faire l'hypothèse courante que « l'intrus connaît explicitement toutes les clefs publiques et ne connaît pas les clefs privées des agents honnêtes ». Il faut en effet toujours démontrer d'abord pour ce codage que le protocole vérifie bien ces hypothèses en vérifiant en quelque sorte le système de distribution des clefs.

Enfin, nous introduisons des prédicats qui permettent aux agents de reconnaître des ensembles réguliers de messages. De ce point de vue, notre modèle est une petite extension de celui de M. Abadi et C. Fournet.

5.5.3 Autres travaux

M. Abadi et A. Gordon [AG98] ont également proposé une équivalence entre processus dans un environnement, appelée *framed bisimilarity* et très proche de celle utilisée par M. Abadi et C. Fournet. Ils ont montré que la framed bisimilarity implique la testing équivalence mais que la réciproque est fausse.

H. Huttel [Hut02] a montré que la framed bisimilarity était décidable pour un fragment du calcul qui ne contient pas la réplication, ni la composition non déterministe ni les clefs composées.

M. Abadi et B. Blanchet [AB02] ont montré que les protocoles exprimés en spi-calcul peuvent se traduire en un système de type tel que qu'un protocole est secret en spi-calcul si et seulement si le système correspondant est bien typé. Cependant, la propriété de secret considérée n'est pas une propriété d'équivalence observationnelle mais une propriété d'accessibilité. Le principal intérêt de ce résultat est son utilisation possible par les outils de vérification comme celui de B. Blanchet [Bla01] par exemple.

5.5.4 Perspectives

Nous pensons que la méthode de preuve de l'équivalence barbue présentée ici est une méthode de preuve qui peut-être réutilisée au moins « à la main » pour prouver le secret d'autres protocoles dans le cadre du spi-calcul. En effet, le point principal de la preuve se ramène à la recherche d'un ensemble de messages invariant par application des règles du protocole, ce qui est une propriété plus intuitive que l'équivalence observationnelle entre protocoles. Notons que cette recherche d'invariants fait partie des méthodes classiques de preuves du secret de protocoles dans des modèles de traces [Mea00a, HS00, Coh00], où le secret est exprimé sous forme d'accessibilité. Il serait intéressant d'automatiser cette méthode de preuve. La décidabilité de l'équivalence entre environnements nous permet de faire le premier pas mais il reste à chercher une automatisation de la recherche d'un invariant, au moins pour une classe restreinte de protocoles.

D'autre part, l'existence d'un invariant apparaît comme une condition *suffisante* pour l'équivalence observationnelle de deux protocoles. Qu'en est-il de la réciproque : lorsque deux protocoles sont équivalents, existe-t-il toujours des environnements invariants quelle que soit l'évolution de chacun des protocoles ? De plus, nous nous sommes limités à l'étude de l'équivalence barbue mais nous pensons pouvoir obtenir comme M. Boreale des résultats similaires pour la testing équivalence définie au paragraphe 5.1.3.2.

Enfin, le spi-calcul est un cadre naturel pour exprimer l'anonymat [SS96, SH02] ou le secret d'un vote (dans ce cas, le secret est pris dans un ensemble fini de messages, connu de l'intrus). En effet, l'intrus connaissant les noms a et b peut distinguer par comparaison les messages $\{a\}_{\mathsf{pub}(b)}$ et $\{b\}_{\mathsf{pub}(b)}$ même s'il ne connaît pas la clef privée $\mathsf{prv}(b)$ de a : les environnements $[a, b, \{a\}_{\mathsf{pub}(b)}]$ et $[a, b, \{b\}_{\mathsf{pub}(b)}]$ ne sont pas équivalents. Intuitivement, l'environnement peut construire les messages $\{a\}_{\mathsf{pub}(b)}$ et $\{b\}_{\mathsf{pub}(b)}$ et les comparer au troisième message de sa mémoire. Ainsi, le protocole naïf où l'agent a signifie à b son intention de lui parler en lui envoyant son nom chiffré avec la clef publique de b :

$$A \to B : \{a\}_{\mathsf{pub}(b)}$$

n'est pas anonyme. V. Shmatikov et D.J.D Hughes ont en particulier commencé à définir l'anonymat sous forme d'équivalence observationnelle [SH02].

Deuxième partie

Vérification pratique des protocoles cryptographiques

CHAPITRE 6

Réduction à un nombre fixé d'agents

La vérification des protocoles cryptographiques est difficile car de nombreux paramètres ne sont pas bornés : nombre arbitraire de sessions et de participants, taille arbitraire des messages... Dans ce chapitre nous démontrons un résultat de réduction sur le nombre de participants. Plus précisément, nous montrons que, sous des hypothèses très générales, deux identités distinctes d'agents suffisent pour trouver une attaque.

Un tel résultat a des applications autant pratiques et théoriques. D'un point de vue théorique, borner le nombre d'agents permet de simplifier l'ensemble des traces à considérer. En outre, ce résultat justifie les abstractions faites dans certains modèles pour faciliter la preuve du secret [Bla01, Pau97a, BLP03]. D'un point de vue pratique, ce résultat peut être utilisé dans des logiciels comme Casrul [JRV00], Aviss [ABB$^+$02] ou Casper [Low97a] pour limiter la recherche d'attaques en n'utilisant qu'au plus deux identités.

Ce chapitre a fait l'objet d'un article [CLC03b] et d'un rapport de recherche [CLC02]. Dans la première partie, nous montrons que deux agents suffisent pour trouver une attaque puis nous augmentons l'expressivité du modèle de manière à pouvoir interdire à un agent de se parler à lui-même. Nous montrons alors que $k+1$ agents suffisent pour trouver une attaque, où k est le nombre de rôles du protocole considéré. Dans tous les cas, nous bornons donc *a priori* le nombre de participants et la borne reste faible : deux ou trois en général. Nous en déduisons un résultat de décidabilité pour un intrus passif. Nous avons essayé d'étendre ce résultat de réduction au nombre de nonces nécessaire à une attaque, ce qui nous permettrait d'en déduire un résultat de décidabilité. Nous verrons à la fin de ce chapitre quelles sont les difficultés pour obtenir un tel résultat de réduction sur les nonces.

6.1 Réduction du nombre de participants

Pour montrer ce résultat de réduction, nous nous plaçons dans le modèle sous forme de clauses de Horn, décrit au chapitre 3. Nous reprenons en particulier toutes les notations et définitions introduites.

Nous avons choisi de montrer ce résultat de réduction dans le cadre du modèle sous forme de clauses car celui-ci est très général. Nous pensons ainsi que la preuve donnée peut être adaptée à tous les modèles de trace existants à ce jour. En particulier, comme nous avons montré que le modèle Millen-Rueß se réduit au modèle sous forme de clauses de Horn (théorème 4.2 du chapitre 4), nous pouvons déduire comme simple corollaire du résultat montré dans ce chapitre que deux agents suffisent pour trouver une attaque dans le modèle Millen-Rueß. Par contre, il ne semble pas que notre résultat puisse s'appliquer directement pour l'équivalence observationnelle de protocoles en spi-calcul (définis au chapitre 5) car nous n'avons pas exprimé la bisimulation en clauses de Horn.

6.1.1 Deux agents suffisent

Nous allons montrer que l'on peut se restreindre à deux agents par une simple projection : étant donnée une attaque avec n agents, nous projetons toutes identités honnêtes sur une unique identité honnête et toutes identités malhonnêtes sur une unique identité malhonnête. La trace obtenue reste une attaque valide ce qui démontre le résultat. Cependant, cette projection nécessite qu'un agent puisse se parler à lui-même ce qui est le cas dans la plupart de modèles de protocoles [MR00, THG99, HLS00, Bla01, CDL$^+$99, AL00]. Mais un résultat similaire est aussi obtenu lorsqu'on interdit à un agent à se parler à lui-même (paragraphe 6.1.2).

6.1.1.1 Définitions

Nous avons vu au chapitre 3 qu'un protocole peut être représenté par un ensemble de clauses de Horn contraintes, où le système de contraintes est $\mathcal{S}$, le système décrit au paragraphe 3.3.1. En particulier, toutes les formules contraintes considérées par la suite dans ce chapitre sont des formules telles que les contraintes sont exprimées dans $\mathcal{S}$. On note $\mathcal{I}_\mathcal{S}$ l'interprétation associée à $\mathcal{S}$.

Nous commençons par restreindre légèrement l'ensemble des clauses possibles pour représenter un protocole : nous spécifions que la description ne doit pas tenir compte de la représentation interne des agents.

Définition 6.1 *Une clause contrainte $\phi \mid K$ est* admissible *si ϕ n'utilise pas les symboles s_h ou s_d (symboles constructeurs des agents honnêtes et malhonnêtes).*

Un ensemble de clauses contraintes est admissible si chacune des clauses contraintes est admissible.

Exemple 6.1 *L'ensemble des clauses contraintes décrit dans la partie 3.4 du chapitre 3 pour représenter le pouvoir de l'intrus et les règles du protocole de Yahalom forme un ensemble de clauses contraintes admissible.*

Il est à noter que cette définition ne restreint pas la classe des protocoles considérés car les descriptions des protocoles n'utilisent pas la représentation interne des agents.

Soit $\mathcal{C}$ un ensemble de clauses contraintes. Nous rappelons que $\mathcal{H}_\mathcal{C}$ désigne le plus petit modèle de Herbrand de $\mathcal{C}$. De plus (définition 3.9), une attaque sur un ensemble de clauses contraintes $\mathcal{C}$ pour la propriété $\phi \mid K$ *avec n agents* est un sous-ensemble fini $\mathcal{H}_0 \subset \mathcal{H}_\mathcal{C}$ tel que $\mathcal{H}_0 \not\models [\![\phi \mid K]\!]$ et tel que $\mathcal{H}_0$ contient exactement n termes distincts de sorte Agent.

6.1.1.2 Réduction

Le théorème de réduction peut maintenant être énoncé formellement.

Théorème 6.1 (Réduction à deux agents) *Soit $\mathcal{C}_P$ un ensemble de clauses contraintes admissibles. Soit $\phi \mid K$ une clause contrainte admissible purement négative. S'il existe* *une attaque sur $\mathcal{C}_P$ pour $\phi \mid K$ avec n agents, alors il existe une attaque sur $\mathcal{C}_P$ pour $\phi \mid K$ avec (au plus) deux agents.*

Remarque : ce résultat de réduction est valable pour des ensembles de clauses admissibles et des propriétés de sécurité purement négatives. Nous avons vu précédemment que la première condition n'est pas une réelle restriction car les clauses représentant des protocoles cryptographiques sont toujours des clauses admissibles. Mais la seconde condition est nettement plus restrictive : si les propriétés de secret s'expriment facilement sous forme de clauses purement négatives, ce n'est pas le cas des propriétés d'authentification décrites au paragraphe 3.4.2.2 du chapitre 3. Nous verrons cependant au paragraphe 6.1.3 comment se ramener, en particulier pour les propriétés d'authentification, à des clauses purement négatives.

Preuve. Soient $\mathcal{C}_P$ un ensemble de clauses contraintes admissibles et $\phi \mid K$ une clause contrainte admissible purement négative. Supposons qu'il existe une attaque sur $\mathcal{C}_P$ pour ϕ avec n agents, c'est-à-dire qu'il existe $\mathcal{H}_0 \subset \mathcal{H}_{\mathcal{C}_P}$ tel que $\mathcal{H}_0 \not\models [\![\phi \mid K]\!]$ et tel que $\mathcal{H}_0$ contient exactement n termes distincts de sorte Agent.

Comme annoncé en introduction, nous allons obtenir une attaque avec deux agents en projetant les termes représentant des agents honnêtes sur un unique terme h et en projetant les termes représentant des agents malhonnêtes sur un unique terme d.

Pour cela, nous précisons comment est construit $\mathcal{H}_{\mathcal{C}_P}$, aussi noté $\mathcal{H}_P$. Soient $\mathcal{M}$ l'ensemble des messages, $\mathcal{T}$ l'ensemble des littéraux clos, positifs et Σ_g l'ensemble des substitutions de l'ensemble des variables dans l'ensemble des messages.

Soit c une clause de Horn contrainte :

$$c = B_1(\vec{x}), ..., B_n(\vec{x}) \Rightarrow A(\vec{x}) \mid K$$

où $B_1(\vec{x}), \dots, B_n(\vec{x}), A(\vec{x})$ sont des littéraux positifs dont les variables libres sont contenues dans $\vec{x}$. Soit S un sous-ensemble de $\mathcal{T}$, on définit $c(S)$ de la manière suivante :

$$c(\mathcal{S}) \stackrel{\text{def}}{=} \{A(\vec{x})\theta\sigma \mid \mathcal{I}_{\mathcal{S}}, \theta \models K, \sigma \in \Sigma_g, \forall i, B_i(\vec{x})\theta\sigma \in \mathcal{S}\}.$$

Alors l'opérateur de conséquence immédiate $F_{\mathcal{C}}$ est une fonction de $2^{\mathcal{T}}$ dans $2^{\mathcal{T}}$ définie par :

$$F_{\mathcal{C}}(S) \stackrel{\text{def}}{=} S \cup \bigcup_{c \in \mathcal{C}} c(S).$$

Pour simplifier, on note F_P l'opérateur $F_{\mathcal{C}_P}$. En reprenant les résultats de [Apt90], l'ensemble des littéraux positifs de $\mathcal{H}_P$, noté $\mathcal{H}_P^+$ est le plus petit point fixe de F_P :

$$\mathcal{H}_P^+ = \bigcup_{k=1}^{+\infty} F_P^k(\emptyset).$$

Pour tout littéral $L \in \mathcal{H}_P^+$, il existe un entier minimal n_L tel que $L \in F_P^{n_L}$.

Nous pouvons maintenant introduire la fonction de projection, notée $\overline{\cdot}$ qui, à tout littéral clos L, associe le littéral $\overline{L}$, dans lequel chaque sous-terme maximal de sorte Ha est remplacé par h et chaque sous-terme de sorte Da est remplacé par d :

$$\begin{array}{rcll}
\overline{f(t_1, \dots, t_n)} & \stackrel{\text{def}}{=} & f(\overline{t_1}, \dots, \overline{t_n}) & \text{si } f \notin \{s_h, s_d\} \\
\overline{s_h(t)} & \stackrel{\text{def}}{=} & h & \\
\overline{s_d(t)} & \stackrel{\text{def}}{=} & d &
\end{array}$$

et $\overline{P(t)} = P(\overline{t})$. De la même façon, la fonction de projection est étendue sur les substitutions par $\overline{\theta}(x) = \overline{\theta(x)}$.

Soit $P \in \{\mathsf{Num}, \mathsf{Agent}, \mathsf{Ha}, \mathsf{Da}, \mathsf{Liste}, \mathsf{Message}, \mathsf{Event}, \mathsf{Trace}\}$ un des prédicats qui peut être utilisé dans les contraintes et soit t un terme de $\mathcal{M}$. Par récurrence sur la profondeur de t, on montre que si $P(t) \in \mathcal{I}_\mathcal{S}$ alors $P(\overline{t}) \in \mathcal{I}_\mathcal{S}$. On en déduit que si $\mathcal{I}_\mathcal{S}, \theta \models K$ pour une contrainte K de $\mathcal{S}$, alors $\mathcal{I}_\mathcal{S}, \overline{\theta} \models K$.

Montrons alors, par récurrence sur n, que si L est un littéral positif de $\mathcal{H}_P$ tel que $n_L = n$, alors $\overline{L}$ est un littéral de $\mathcal{H}_P$.
Si $n_L = 0$, il n'y a pas de littéral L tel que $n_L = n$ donc il n'y a rien à démontrer.
Supposons la propriété vraie pour $n_L \leqslant n$. Soit L littéral positif de $\mathcal{H}_P$ tel que $n_L = n + 1$. Alors il existe une clause contrainte c_L et des littéraux positifs $L_1, \ldots, L_k \in \mathcal{H}_P^+$, antécédents de L par c_L, c'est-à-dire $L \in c_L(\{L_1, \ldots, L_k\})$ avec $n_{L_i} \leqslant n$ pour tout $1 \leqslant i \leqslant k$. Par hypothèse de récurrence, $\overline{L_1}, \ldots, \overline{L_k} \in \mathcal{H}_P^+$.

$$c_L = B_1(\vec{x}), \ldots, B_k(\vec{x}) \Rightarrow A(\vec{x}) \mid K$$

et $L = A(\vec{x})\theta\sigma$, $L_i = B_1(\vec{x})\theta\sigma$ avec $\mathcal{I}_\mathcal{S}, \theta \models K$. Comme c_L est une clause contrainte admissible, elle ne contient pas les symboles s_h et s_d et donc $\overline{L} = A(\vec{x})\overline{\theta}\overline{\sigma}$, $\overline{L_i} = B_1(\vec{x})\overline{\theta}\overline{\sigma}$. De plus, d'après ce qui précède, si $\mathcal{I}_\mathcal{S}, \theta \models K$, alors $\mathcal{I}_\mathcal{S}, \overline{\theta} \models K$. D'où $\overline{L} \in c_L(\{\overline{L_1}, \ldots \overline{L_k}\})$, et donc $\overline{L} \in \mathcal{H}_P^+$.

On peut maintenant conclure : comme $\phi \mid K$ est purement négative, on peut supposer que $\mathcal{H}_0$ ne contient que des littéraux positifs. On pose $\mathcal{H}_1 = \overline{\mathcal{H}_0}$. L'ensemble $\mathcal{H}_1$ est fini et d'après ce qui précède, $\mathcal{H}_1 \subset \mathcal{H}_P$. Il reste à montrer $\mathcal{H}_1 \not\models \phi$. Soit $\phi\theta\sigma$ une instance de ϕ falsifiée par $\mathcal{H}_0$ et telle que : $\mathcal{I}_\mathcal{S}, \theta \models K$. $\overline{\phi\theta\sigma}$ est falsifiée par $\mathcal{H}_1$ et $\mathcal{I}_\mathcal{S}, \overline{\theta} \models K$. À nouveau, comme $\phi \mid K$ est une clause contrainte admissible, elle ne contient pas les symboles s_h et s_d et donc $\overline{\phi\theta\sigma} = \phi\overline{\theta}\overline{\sigma}$, d'où $\mathcal{H}_1 \not\models [\![\phi \mid K]\!]$. $\square$

6.1.2 $k+1$ agents suffisent

Le résultat démontré dans la partie précédente est énoncé dans un modèle très général. Cependant, en projetant tous les agents honnêtes sur une même identité honnête, nous avons implicitement utilisé le fait qu'un agent peut se parler à lui-même. Nous allons donc à nouveau établir ce résultat de réduction en ajoutant la possibilité d'interdire à un agent de se parler à lui-même.

6.1.2.1 Un agent peut-il se parler à lui-même ?

Même si de très nombreux modèles autorisent un agent à se parler à lui-même, certains interdisent cette possibilité [Pau98, Low98]. Le résultat de réduction à deux agents ne peut donc pas s'appliquer pour ces modèles. De plus, permettre à un agent de se parler à lui-même peut introduire des attaques qui n'apparaissent que pour des sessions où un agent se parle réellement à lui-même. Or un tel comportement est peu réaliste : pourquoi un agent honnête s'enverrait-il un secret à lui-même à travers le réseau ?

Exemple 6.2 *On considère un protocole construit pour l'occasion et composé d'une seule règle :*

$$A \rightarrow B : \{A, B, N_a\}_{\mathsf{pub}(B)}, \{secret\}_{\{A,A,N_a\}_{\mathsf{pub}(B)}}.$$

L'agent B peut construire la clef composée $\{A, A, N_a\}_{\mathsf{pub}(B)}$ et obtenir le secret. Si B est un agent honnête, l'intrus ne peut pas connaître N_a et ne peut donc pas construire

$\{A, A, N_a\}_{\mathsf{pub}(B)}$. Le seul message contenant N_a et connu de l'intrus est $\{A, B, N_a\}_{\mathsf{pub}(B)}$ donc si A et B sont distincts, le secret n'est pas connu de l'intrus. Par contre si $A = B$, alors l'intrus peut déchiffrer la deuxième partie du message et obtenir le secret.

Conclusion : le secret est préservé si et seulement si aucun agent honnête ne peut se parler à lui-même.

On ajoute donc un symbole de prédicat Distinct permettant de distinguer les agents honnêtes. Ainsi, on pourra ajouter dans les prémisses des règles la condition que les agents honnêtes doivent être distincts. Un agent malhonnête pourra par contre toujours se comporter comme il le souhaite, c'est-à-dire éventuellement se parler à lui-même, mais ce dernier cas n'augmente en général pas la connaissance de l'intrus.

$$\mathcal{C}_{\neq} \stackrel{\text{def}}{=} \left\{ \begin{array}{rcll} & \Rightarrow & \mathsf{Distinct}(h, s_h(x)) & \mid \mathsf{Ha}(x) \\ & \Rightarrow & \mathsf{Distinct}(x, y) & \mid \mathsf{Ha}(x), \mathsf{Da}(y) \\ & \Rightarrow & \mathsf{Distinct}(x, y) & \mid \mathsf{Da}(x), \mathsf{Da}(y) \\ \mathsf{Distinct}(x, y) & \Rightarrow & \mathsf{Distinct}(s_h(x), s_h(y)) & \mid \mathsf{Ha}(x), \mathsf{Ha}(y) \\ \mathsf{Distinct}(x, y) & \Rightarrow & \mathsf{Distinct}(y, x) & \end{array} \right.$$

L'interprétation de Distinct dans le plus petit modèle de Herbrand de $\mathcal{C}_{\neq}$ est l'ensemble des paires $(s_h^k(h), s_d^m(d))$, $(s_d^m(d), s_h^k(h))$, $(s_d^m(d), s_d^k(d))$ et $(s_h^i(h), s_h^j(h))$ avec $i \neq j$ et $i, j, k, m \in \mathbb{N}$.

Remarque : On pourrait aussi choisir de prendre pour prédicat Distinct, le prédicat qui distingue tous les agents. De cette manière, aucun participant du protocole ne pourra s'envoyer un message à lui-même en suivant une règle du protocole, qu'il soit honnête ou malhonnête. Pour ce choix du prédicat Distinct, on obtient un résultat de réduction identique, à condition d'expliciter une condition toujours vérifiée par les protocoles réels : lorsqu'un agent malhonnête joue une règle du protocole, la connaissance de l'intrus n'augmente pas. Cette propriété est bien vérifiée pour les protocoles provenant de spécifications réelles : la connaissance de l'intrus étant supérieure à celles des agents malhonnêtes, tout ce qu'un agent malhonnête peut construire, l'intrus pour le faire également.

On étend la définition des clauses admissibles aux clauses qui contiennent le prédicat Distinct.

Définition 6.2 *Soit $\phi \mid K$ une clause contrainte. $\phi \mid K$ est* admissible *si :*

1. *ϕ ne contient pas les symboles s_h et s_d,*
2. *et pour tout littéral L de ϕ tel que $L = \pm\mathsf{Distinct}(t(\vec{x}))$, L est un littéral négatif : $L = \neg\mathsf{Distinct}(t(\vec{x}))$.*

On peut ainsi représenter un protocole sous forme de clauses admissibles, en interdisant (ou pas) à un agent de se parler à lui-même : il suffit d'ajouter le prédicat Distinct dans les prémisses des clauses représentant les règles du protocole.

Exemple 6.3 *On reprend l'exemple du protocole de Yahalom mais on interdit désormais, et pour chaque règle, qu'un agent se parle à lui-même. Le pouvoir de l'intrus est inchangé. Les nouvelles clauses correspondant aux règles du protocole sont décrites à la figure 6.1.*

Toutes les clauses qui suivent sont contraintes par la clause :

$$K_0 = \mathsf{Agent}(a) \wedge \mathsf{Agent}(b) \wedge \mathsf{Message}(x) \wedge \mathsf{Message}(y) \wedge \mathsf{Message}(z) \wedge \mathsf{Num}(s) \wedge \mathsf{Trace}(t).$$

$$\mathsf{Fresh}(t,s), T(t) \Rightarrow \left\{ \begin{array}{ll} T(& [st(a,1,1,<a,b,srv>),s] \\ & \cdot[st(b,2,1,<b,srv>),s] \\ & \cdot[st(srv,3,1,srv),s] \cdot t) \end{array} \right.$$

$$\left. \begin{array}{r} \mathsf{NotPlayed}(a,1,2,s,t), T(t), \\ \mathsf{Distinct}(a,b), \\ \mathsf{In}([st(a,1,1,<a,b,srv>),s],t) \end{array} \right\} \Rightarrow \left\{ \begin{array}{ll} T(& [<a, n_1([a;b],s)>, s] \\ & \cdot[st(a,1,2,<a,b,srv,n_1([a;b],s)>),s] \\ & \cdot t) \end{array} \right.$$

$$\left. \begin{array}{r} \mathsf{NotPlayed}(b,2,2,s,t), \mathsf{Distinct}(a,b), \\ T(t), I(<a,x>,t), \\ \mathsf{In}([st(b,2,1,<b,srv>),s],t) \end{array} \right\} \Rightarrow \left\{ \begin{array}{ll} T(& [<b, \{a,x,n_2([a;b],s)\}_{\mathsf{shr}(b)}>, s] \\ & \cdot[st(b,2,2,<b,srv,a,n_2([a;b],s)>),s] \\ & \cdot t) \end{array} \right.$$

$$\left. \begin{array}{r} I(<b,\{a,x,y\}_{\mathsf{shr}(b)}>,t), \\ \mathsf{Distinct}(a,b), \\ \mathsf{NotPlayed}(srv,3,2,s,t), T(t), \\ \mathsf{In}([st(srv,3,1,srv),s],t) \end{array} \right\} \Rightarrow \left\{ \begin{array}{ll} T(& [m,s] \\ & \cdot[st(srv,3,2,srv),s] \cdot t) \end{array} \right.$$

$$\left. \begin{array}{r} T(t), \mathsf{Distinct}(a,b), \mathsf{In}([u_1,s],t), \\ \mathsf{NotPlayed}(a,1,3,s,t), \\ I(<\{b,x,n_1([a;b],s),y\}_{\mathsf{shr}(a)}, z>, t) \end{array} \right\} \Rightarrow T([<z,\{y\}_x>,s] \cdot [u_2,s] \cdot t)$$

$$\begin{array}{lrcl} \text{où} & m & = & <\{b,n_3([a;b],s),x,y\}_{\mathsf{shr}(a)}, \{a,n_3([a;b],s)\}_{\mathsf{shr}(b)}>, \\ & u_1 & = & st(a,1,2,<a,b,srv,n_1([a;b],s)>), \text{ et} \\ & u_2 & = & st(a,1,3,<a,b,srv,n_1([a;b],s),x,y>). \end{array}$$

FIGURE 6.1 - *Protocole de Yahalom : un agent ne peut pas se parler à lui-même.*

6.1.2.2 Réduction

Notre résultat de réduction dépend maintenant de la propriété de sécurité $\phi \mid K$ considérée : si la clause ϕ utilise k variables distinctes représentant des agents honnêtes, et s'il y a une attaque, alors il y a une attaque avec (au plus) $k+1$ agents. Précisons tout d'abord ce que signifie qu'une variable *représente un agent honnête.*

Définition 6.3 *Soit $\phi \mid K$ une clause contrainte. Une variable x de ϕ* est de sorte Ha *s'il existe une contrainte K' telle que $K = \mathsf{Ha}(x) \wedge K'$.*

Théorème 6.2 (réduction à $k+1$ agents) *Soit $\mathcal{C}_P$ un ensemble de clauses contraintes admissibles, qui ne contient pas de variables de sortes* Ha. *Soit $\phi \mid K$ une clause admissible purement négative.*

S'il existe une attaque sur $\mathcal{C}_P \cup \mathcal{C}_{\neq}$ pour la propriété $\phi \mid K$ avec n agents, alors il existe une attaque sur $\mathcal{C}_P$ pour la propriété $\phi \mid K$ avec (au plus) $k+1$ agents, où k est le nombre de variables de ϕ de sorte Ha.

Remarque : par rapport au théorème précédent, nous interdisons que $\mathcal{C}_P$ contienne des variables de sortes Ha. Cette condition est en fait vérifiée par les clauses représentant des protocoles. En effet, la spécification du protocole lui-même ne doit pas distinguer les agents honnêtes des agents malhonnêtes. D'autre part, le pouvoir de l'intrus ne doit pas dépendre des agents honnêtes : ou bien une donnée est connue pour tous les agents (comme les identités des agents ou leurs clefs publiques), ou bien, elle est connue seulement pour les agents malhonnêtes (comme les clefs compromises).

La seule réelle restriction reste que la propriété considérée doit être une clause purement négative mais, comme annoncé précédemment, cette limitation sera partiellement éliminée au paragraphe 6.1.3.

Preuve. Soient $\mathcal{C}_P$ un ensemble de clauses contraintes admissibles qui ne contient pas de variables de sortes Ha, et $\phi \mid K$ une clause admissible purement négative qui contient exactement k variables de sorte Ha : $x_1, \ldots x_k$.

Nous reprenons les notations définies dans la preuve du théorème 6.1. $\mathcal{H}_P$ désigne maintenant le plus petit modèle de Herbrand de $\mathcal{C}_P \cup \mathcal{C}_{\neq}$. On considère à nouveau $\mathcal{H}_0$, sous ensemble fini de $\mathcal{H}_P$ qui forme une attaque sur $\mathcal{C}_P \cup \mathcal{C}_{\neq}$ pour la propriété $\phi \mid K$ avec n agents : $\mathcal{H}_0 \not\models [\![\phi \mid K]\!]$. Comme ϕ est purement négative, on peut également supposer que $\mathcal{H}_0$ ne contient que des littéraux positifs.

Soit $\phi\theta\sigma$ une instance de ϕ, falsifiée par $\mathcal{H}_0$ telle que $\mathcal{I}_{\mathcal{S}}, \theta \models K$. Soient $s_h^{m_1}(h), \ldots, s_h^{m_p}(h)$ les éléments de l'ensemble $\{x_1\theta, \ldots, x_k\theta\}$ avec $m_1 < \cdots < m_p$ ($p \leqslant k$). L'idée de la preuve est de garder ces p agents comme des agents honnêtes distincts et d'envoyer tous les autres agents (honnêtes ou malhonnêtes) sur une unique identité malhonnête. On définit alors la fonction de projection de la manière suivante :

$$\begin{cases} \overline{f(t_1, \ldots, t_n)} & \stackrel{\text{def}}{=} & f(\overline{t_1}, \ldots, \overline{t_n}) & \text{si } f(t_1, \ldots, t_n) \text{ n'est pas dans l'interprétation de } \mathsf{Ha} \text{ ou } \mathsf{Da}, \\ \overline{s_h^{m_i}(h)} & \stackrel{\text{def}}{=} & s^{i-1}(h) & \text{pour } 1 \leqslant i \leqslant p, \\ \overline{t} & \stackrel{\text{def}}{=} & d & \text{sinon.} \end{cases}$$

Nous montrons à nouveau que si L est un littéral positif de $\mathcal{H}_P$, alors $\overline{L}$ est aussi un littéral de $\mathcal{H}_P$. Pour cela, nous prouvons d'abord le lemme suivant.

Lemme 6.1 *Si* $\mathsf{Distinct}(u_1, u_2) \in F_P^n(\emptyset)$*, alors* $\overline{\mathsf{Distinct}(u_1, u_2)} \in F_P^n(\emptyset)$.

preuve du lemme 6.1

Comme Distinct n'apparaît positivement que dans les clauses de $\mathcal{C}_{\neq}$, l'interprétation de Distinct dans $\mathcal{H}_P$ reste l'ensemble des paires :

$$(s_h^k(h), s_d^m(d)), (s_d^m(d), s_h^k(h)), (s_d^m(d), s_d^k(d)) \text{ et } (s_h^i(h), s_h^j(h)) \text{ avec } i \neq \text{ et } i, j, k, m \in \mathbb{N}.$$

On pose $t_i = x_i\theta$ et on considère des termes u_1, u_2 tels que $\mathsf{Distinct}(u_1, u_2) \in F_P^n(\emptyset)$. Trois cas sont alors possibles :

1. soit $u_1, u_2 \notin \{t_1, \ldots, t_k\}$, alors en examinant les quatre cas possibles pour u_1 et u_2, on conclut que $\overline{\mathsf{Distinct}(u_1, u_2)} = \mathsf{Distinct}(d, d) \in F_P(\emptyset)$;
2. soit $u_1 \in \{t_1, \ldots, t_k\}$ et $u_2 \notin \{t_1, \ldots, t_k\}$ (ou le contraire), alors $\overline{\mathsf{Distinct}(u_1, u_2)} = \mathsf{Distinct}(s_h^j(h), d)$ (ou $\mathsf{Distinct}(d, s_h^j(h))$), qui appartient aussi à $F_P(\emptyset)$;
3. soit $u_1, u_2 \in \{t_1, \ldots, t_k\}$: $u_1 = s_h^{m_i}(h), u_2 = s_h^{m_j}(h)$ avec $i \neq j$, donc $\overline{\mathsf{Distinct}(u_1, u_2)} = \mathsf{Distinct}(s_h^i(h), s_h^j(h)) \in F_P^{|j-i|}(\emptyset)$. Dans ce cas, $|j - i| \leqslant |m_j - m_i|$ par construction, d'où le résultat.

fin de la preuve du lemme 6.1

Montrons alors, par récurrence sur n, que si L est un littéral positif de $\mathcal{H}_P$ tel que $n_L = n$, alors $\overline{L}$ est un littéral de $\mathcal{H}_P$.

Si $n_L = 0$, il n'y a pas de littéral L tel que $n_L = n$ donc il n'y a rien à démontrer.

Supposons la propriété vraie pour $n_l \leqslant n$. Soit L littéral positif de $\mathcal{H}_P$ tel que $n_L = n + 1$. Alors il existe une clause contrainte c_L et des littéraux positifs $L_1, \ldots, L_k \in \mathcal{H}_P^+$, antécédents de L par c_L. Deux cas sont alors possibles.

- Si $c_L \in \mathcal{C}_P$, alors

$$c_L = B_1(\vec{x}), \ldots, B_k(\vec{x}) \Rightarrow A(\vec{x}) \mid K$$

  et $L = A(\vec{x})\theta\sigma$, $L_i = B_1(\vec{x})\theta\sigma$ avec $\mathcal{I}_\mathcal{S}, \theta \models K$. De plus, $n_{L_i} \leqslant n$ pour tout i tel que $1 \leqslant i \leqslant k$ donc, par hypothèse de récurrence, $\overline{L_1}, \ldots, \overline{L_k} \in \mathcal{H}_P^+$.

  Comme c_L est une clause contrainte admissible, elle ne contient pas les symboles s_h et s_d et donc $\overline{L} = A(\vec{x})\overline{\theta}\overline{\sigma}$, $\overline{L_i} = B_1(\vec{x})\overline{\theta}\overline{\sigma}$. De plus, comme $\mathcal{I}_\mathcal{S}, \theta \models K$ et que K *ne contient pas le prédicat* Ha, on déduit $\mathcal{I}_\mathcal{S}, \overline{\theta} \models K$. D'où $\overline{L} \in c_L(\{\overline{L_1}, \ldots \overline{L_k})$, et donc $\overline{L} \in \mathcal{H}_P^+$.
- Si $c_L \in \mathcal{C}_{\neq}$, alors $L = \mathsf{Distinct}(u_1, u_2)$ et le lemme 6.1 permet de conclure également que $\overline{L} \in \mathcal{H}_P$.

On considère maintenant $\mathcal{H}_1 = \overline{\mathcal{H}_0}$. D'après ce qui précède, $\mathcal{H}_1$ est un sous-ensemble (fini) de $\mathcal{H}_P$ et $\mathcal{H}_1$ falsifie $\overline{\phi\theta\sigma}$. Comme $\phi \mid K$ est une clause contrainte admissible, ϕ ne contient pas les symboles s_h et s_d donc $\overline{\phi\theta\sigma} = \phi\overline{\theta}\overline{\sigma}$. De plus, la fonction de projection a été choisie de telle manière que $\mathcal{I}_\mathcal{S}, \overline{\theta} \models K$, donc $\mathcal{H}_1 \not\models [\![\phi \mid K]\!]$: $\mathcal{H}_1$ est une attaque sur $\mathcal{C}_P$ pour la propriété $\phi \mid K$. De plus l'ensemble $\mathcal{H}_1$ contient exactement $p \leqslant k$ termes de sorte Ha et un terme de sorte Da, ce qui termine la démonstration. □

6.1.2.3 $k + 1$ agents peuvent être nécessaires

La borne $k + 1$ est minimale : pour tout entier k, il existe un ensemble de clauses $\mathcal{C}_P$ et une propriété ϕ avec k variables de sorte Ha tels que toute attaque sur $\mathcal{C}_P$ pour ϕ utilise au moins $k + 1$ agents.

Toutes les clauses qui suivent sont contraintes par la clause :

$K_0 \stackrel{\text{def}}{=} \mathsf{Message}(x) \wedge \mathsf{Message}(y) \wedge \mathsf{Message}(z) \wedge \mathsf{Num}(s) \wedge \mathsf{Trace}(t) \wedge \mathsf{Agent}(a_1) \wedge \cdots \wedge \mathsf{Agent}(a_k)$

On pose $u =< a_1, \ldots, a_k >$.

Initialisation

$$\mathsf{Fresh}(t, s), T(t) \Rightarrow T([st(a_1, 1, 1, u), s] \cdot [st(a_2, 2, 1, a_2), s] \cdot \cdots \cdot [st(a_k, k, 1, a_k), s] \cdot t)$$

Premier message : $A_1 \rightarrow A_2 : \{A_1, A_2, \ldots, A_k, N_{A_1}\}_{\mathsf{pub}(A_2)}$, $A_i \neq A_j$, pour $i \neq j$.

$$\left.\begin{array}{r} T(t), \mathsf{Distinct}(a_i, a_j) \quad i \neq j \\ \mathsf{In}([st(a_1, 1, 1, u), s], t), \\ \mathsf{NotPlayed}(a_1, 1, 2, s, t) \end{array}\right\} \Rightarrow \begin{array}{ll} T(& [\{u, n_1(a_1, \ldots, a_k, s)\}_{\mathsf{pub}(a_2)}, s] \\ & \cdot[st(a_1, 1, 2, < u, n_1(a_1, \ldots, a_k, s) >), s] \\ & \cdot t) \end{array}$$

Deuxième message : $A_2 \rightarrow A_1 : \{N_{A_1}, N_{A_2}\}_{\mathsf{pub}(A_1)}$

$$\left.\begin{array}{r} T(t), \mathsf{Distinct}(a_i, a_j) \quad i \neq j \\ I(\{u, x\}_{\mathsf{pub}(a_2)}, t) \\ \mathsf{In}([st(a_2, 2, 1, a_2), s], t), \\ \mathsf{NotPlayed}(a_2, 2, 2, s, t) \end{array}\right\} \Rightarrow \begin{array}{ll} T(& [\{x, n_2(a_1, \ldots, a_k, s)\}_{\mathsf{pub}(a_1)}, s] \\ & \cdot[st(a_2, 2, 2, < u, n_2(a_1, \ldots, a_k, s) >), s] \\ & \cdot t) \end{array}$$

Troisième message : $A_1 \rightarrow A_2 : \{N_{A_2}\}_{\mathsf{pub}(A_2)}$

$$\left.\begin{array}{r} T(t), I(\{n_1(a_1, \ldots, a_k, s), y\}_{\mathsf{pub}(a_1)}, t) \\ \mathsf{In}([st(a_1, 1, 2, < u, n >), s], t), \\ \mathsf{NotPlayed}(a_2, 1, 3, s, t) \end{array}\right\} \Rightarrow \begin{array}{ll} T(& [\{y\}_{\mathsf{pub}(a_1)}, s] \\ & \cdot[st(a_1, 1, 3, < u, n_1(a_1, \ldots, a_k, s), y >), s] \\ & \cdot t) \end{array}$$

FIGURE 6.2 - *Protocole de Needham-Schroeder étendu à k agents.*

En effet, soit $k \geqslant 2$. On considère le protocole décrit à la figure 6.2, inspiré du protocole de Needham-Schroeder et on pose $\mathcal{C}_P$ l'ensemble des clauses « indépendantes du protocole » décrites dans la partie 3.4.1 du chapitre 3, auxquelles on ajoute les clauses de la figure 6.2. Le protocole informel correspondant est le suivant :

$$\begin{array}{lll} A_1 & \to A_2 & : \{A_1, A_2, \ldots, A_k, N_{A_1}\}_{\mathsf{pub}(A_2)}, \qquad\qquad A_i \neq A_j \text{ pour } i \neq j \\ A_2 & \to A_1 & : \{N_{A_1}, N_{A_2}\}_{\mathsf{pub}(A_1)} \\ A_1 & \to A_2 & : \{N_{A_2}\}_{\mathsf{pub}(A_2)} \end{array}$$

On pourrait ajouter d'autres règles pour les participants $A_3, \ldots, A_k$ afin de rendre le protocole plus réaliste.

On considère alors la propriété :

$$\phi \mid K = \neg T(t) \vee \neg I(n_2(x_1, \ldots, x_k, s), t) \quad \mid \mathsf{Ha}(x_1) \wedge \cdots \wedge \mathsf{Ha}(x_k) \wedge \mathsf{Num}(s) \wedge \mathsf{Trace}(t).$$

En reprenant l'attaque trouvée par G. Lowe [Low96], il existe une attaque sur $\mathcal{C}_P$ pour ϕ qui utilise exactement $k+1$ agents. Montrons maintenant que toute attaque utilise au moins $k+1$ agents. Soit $\mathcal{H}_0$ une attaque : $\mathcal{H}_0 \subset \mathcal{H}_P$ où $\mathcal{H}_P$ est le plus petit modèle de Herbrand de $\mathcal{C}_P \cup \mathcal{C}_{\neq}$. Alors, il existe $t, s, a_1, \ldots, a_k$ tels que $I(n_2(a_1, \ldots, a_k, s), t) \in \mathcal{H}_0$ et les a_i sont dans l'interprétation de Ha pour $\mathcal{C}_P$. Comme $n_2(a_1, \ldots, a_k, s)$ n'est produit que si $\mathsf{Distinct}(a_i, a_j)$ est vérifié pour tout $i \neq j$ et comme $a_1, \ldots, a_k$ sont dans l'interprétation de Ha, les $a_1, \ldots, a_k$ sont distincts. De plus, si aucune identité malhonnête n'est utilisée alors l'intrus ne pourra décrypter aucun des messages et ne pourra donc pas obtenir $n_2(a_1, \ldots, a_k, s)$. Par conséquent, au moins une identité malhonnête a été utilisée et donc, au total, $\mathcal{H}_0$ contient au moins $k+1$ termes de sorte Agent.

6.1.3 Discussion des hypothèses

Les deux théorèmes de réduction concernent des ensembles très généraux de clauses $\mathcal{C}_P$. Par contre, la propriété de sécurité doit être une clause contrainte purement négative, ce qui n'est pas le cas des propriétés d'authentification énoncées au paragraphe 3.4.2.2 du chapitre 3. Nous allons donc voir comment transformer certaines propriétés de sécurité en clauses purement négatives.

Tout d'abord, il est à noter que les théorèmes 6.1 et 6.2 ne sont pas valides si on ne suppose pas que la propriété de sécurité est purement négative.

Exemple 6.4 *En effet, considérons l'ensemble de clauses admissibles :*

$$\mathcal{C}_P \stackrel{def}{=} \left\{ \begin{array}{lll} \Rightarrow & A(x,y) & \mid \mathsf{Da}(x) \wedge \mathsf{Agent}(y) \\ \Rightarrow & A(x,y) & \mid \mathsf{Agent}(x) \wedge \mathsf{Da}(y) \\ \Rightarrow & A(x,x) & \mid \mathsf{Agent}(x) \end{array} \right.$$

et ϕ la clause $A(x,y)$. Alors $\mathcal{H}_0 = \{\neg A(h, s_h(h))\}$ est une attaque sur $\mathcal{C}_P$ pour ϕ et pourtant, il n'existe pas d'attaque avec un seul agent honnête.

Examinons de plus près les propriétés d'authentification. Elles sont de la forme :

$$P_1 \stackrel{\text{def}}{=} \neg T(t) \vee \neg I(m(x,y,s), t) \vee \mathsf{In}([st(x,i,j,m(x,y,s)), s], t) \mid \mathsf{Ha}(x) \wedge \mathsf{Ha}(y) \wedge \mathsf{Num}(s) \wedge \mathsf{Trace}(t),$$

où $\mathsf{In}([st(x,i,j,m(x,y,s)),s],t)$ est un littéral positif. Cependant, ce littéral fait partie d'une classe de littéraux définis en clauses de Horn et tels que leurs négations s'expriment elles aussi en clauses de Horn. Nous présentons une telle classe dans le paragraphe suivant puis nous l'appliquons de manière à obtenir une classe plus générale de propriétés de sécurité admissibles.

6.1.3.1 Techniques de complémentations

Nous utilisons ici les techniques de complémentations développées dans [Com88, BMPT90, Com91]. Étant donné un prédicat P et des clauses de Horn le définissant, ces techniques visent à construire un ensemble de clauses définissant un prédicat $\tilde{P}$ tel que la formule $P \wedge \tilde{P}$ soit fausse *pour toute instance close* dans le plus petit modèle de Herbrand. Nous avons en fait besoin d'une propriété supplémentaire, à savoir que la propriété $P \vee \tilde{P}$ est vraie pour toute instance close dans le plus petit modèle de Herbrand. Nous introduisons la classe des prédicats *déterministes* qui vérifie cette propriété lors du passage à la négation.

Définition 6.4 (prédicat déterministe) *Soit $\mathcal{C}$ un ensemble de clauses de Horn. L'ensemble des symboles de* prédicats déterministes *de $\mathcal{C}$ est le plus petit ensemble de symboles de prédicats $\mathcal{D}$ tel que, pour toute clause $B \Rightarrow Q(\vec{u})$ de $\mathcal{C}$ avec $\vec{u} = (u_1, \ldots, u_k)$,*
- *si les variables libres de B sont contenues dans les variables libres de $\vec{u}$,*
- *si chaque littéral de B est de la forme $P(y_1, \ldots, y_n)$ avec les y_i tous distincts,*
- *si chaque prédicat apparaissant dans B est soit Q lui-même, soit un élément de $\mathcal{D}$,*
- *et si au moins l'un des u_i n'est pas une variable,*

alors $Q \in \mathcal{D}$.

Exemple 6.5 *De tels prédicats permettent en particulier de définir les ensembles reconnus par des automates d'arbres avec égalité.*

Ils permettent également de reconnaître tous les ensembles primitifs récursifs.

Un prédicat déterministe P est donc défini par une clause de la forme :

$$\bigvee_i \exists \vec{y_i} \left[\vec{x} = \vec{u_i}(\vec{y_i}) \wedge \bigwedge_j Q_{i,j}(\vec{y_i}) \right] \Rightarrow P(\vec{x}),$$

où les $Q_{i,j}$ sont également des prédicats déterministes.

Définition 6.5 (négation d'un prédicat déterministe) *On définit le prédicat $\tilde{P}$ associé au prédicat déterministe P par la formule ϕ_P définie ci-dessous :*

$$\bigwedge_i \forall \vec{y_i} \left[\vec{x} \neq \vec{u_i}(\vec{y_j}) \wedge \bigvee_j \tilde{Q}_{i,j}(\vec{y_j}) \right] \Rightarrow \tilde{P}(\vec{x}).$$

En utilisant les techniques d'élimination des quantificateurs [Com88, CL89], toute formule ϕ_P ainsi obtenue est logiquement équivalente à une conjonction de clauses de Horn. On note $\mathcal{C}_P$ l'ensemble de clauses de Horn ainsi obtenu. D'après les résultats de [Com88], la formule $P \wedge \tilde{P}$ est fausse pour toute instance close dans le plus petit modèle de Herbrand.

Pour les prédicats déterministes, la formule $P \vee \tilde{P}$ est vraie pour toute instance close dans le plus petit modèle de Herbrand.

Proposition 6.1 *Soit $\mathcal{C}$ un ensemble de clauses de Horn et $\mathcal{D}$ l'ensemble des prédicats déterministes associés à $\mathcal{C}$. On pose $\mathcal{C}' = \mathcal{C} \cup \bigcup_{P \in \mathcal{D}} \mathcal{C}_P$ et $\mathcal{H}$ le plus petit modèle de Herbrand de $\mathcal{C}'$. Alors pour tout prédicat P de $\mathcal{D}$, pour toute substitution close σ, la formule $P(\vec{x})\sigma \vee \tilde{P}(\vec{x})\sigma$ est vraie dans $\mathcal{H}$.*

Preuve. Montrons que pour tout vecteur $\vec{t}$ de termes clos, $\mathcal{H} \models P(\vec{t})$ ou $\mathcal{H} \models \tilde{P}(\vec{t})$, par récurrence sur la taille de $\vec{t}$.

Soit $\vec{t}$ un vecteur de termes clos.

1. Ou bien il existe une substitution σ et un entier i tels que $\vec{t} = u_i(\vec{y_i})\sigma$ et pour tout j $Q_{i,j}(\vec{y_i})\sigma \in \mathcal{H}$ et alors $P(\vec{x})\sigma \in \mathcal{H}$.
2. Ou bien pour toute substitution σ et pour tout entier i, a) $\vec{t} \neq u_i(\vec{y_i})\sigma$ ou b) $\exists j\ Q_{i,j}(\vec{y_i})\sigma \notin \mathcal{H}$. Fixons une substitution σ et un entier i tels que $\vec{t} = u_i(\vec{y_i})\sigma$. Alors, $\exists j\ Q_{i,j}(\vec{y_i})\sigma \notin \mathcal{H}$. Comme les y_i ne sont pas répétés et qu'au moins une des u_i n'est par une variable, le vecteur $\vec{y_i}\sigma$ est de taille strictement inférieure au vecteur $\vec{t}$. On peut donc appliquer l'hypothèse de récurrence et il vient que pour toute substitution σ et pour tout entier i, $\vec{t} \neq u_i(\vec{y_i})\sigma$ ou $\exists j\ \tilde{Q}_{i,j}(\vec{y_i})\sigma \in \mathcal{H}$ et donc $\tilde{P}(\vec{x})\sigma \in \mathcal{H}$.

Il reste à vérifier le cas où $\vec{t}$ est de taille 1 : $\vec{t} = (a)$ où a est une constante. Le cas 2b) n'est pas possible, on en déduit que : $\mathcal{H} \models P(\vec{t})$ ou $\mathcal{H} \models \tilde{P}(\vec{t})$. □

6.1.3.2 Extension des propriétés de sécurité admissibles

Nous pouvons maintenant étendre la définition des propriétés admissibles de manière à obtenir une classe de propriétés englobant le secret et l'authentification.

Définition 6.6 (propriété de sécurité admissible) *Une clause contrainte $\phi \mid K$ est une* propriété de sécurité admissible *si ϕ est de la forme :*

$$\phi = \phi_1 \vee \mathsf{In}(u_1, t_1) \vee \cdots \vee \mathsf{In}(u_n, t_n),$$

où ϕ_1 est une clause purement négative, les t_i sont des variables de sorte Trace*, les u_i sont des termes avec variables. De plus, ϕ doit vérifier les propriétés suivantes :*

1. *elle ne contient pas les symboles s_h et s_d,*
2. *les variable des termes u_i sont de sorte* Num *ou* Ha*,*
3. *tout sous terme clos de u_i de sorte* Agent *est de sorte* Ha*.*

Ainsi la propriété d'authentification P_1 définie à la page 125 est une propriété de sécurité admissible.

Avec cette nouvelle définition, la réduction à $k+1$ agents reste valide.

Théorème 6.3 *Soit $\mathcal{C}_P$ un ensemble de clauses admissibles, qui ne contient pas de variables de sorte* Ha*. Soit $\phi \mid K$ une propriété de sécurité admissible. S'il existe une attaque sur $\mathcal{C}_P$ pour $\phi \mid K$ qui utilise n agents, alors il existe une attaque sur $\mathcal{C}_P$ pour $\phi \mid K$ qui utilise (au plus) $k+1$ agents où k est le nombre de variables de sortes* Ha *dans ϕ.*

Ainsi, trois agents sont suffisants pour la propriété de sécurité P_1 décrite à la page 125.

Preuve. Pour chaque littéral positif $L_i = \mathsf{In}(u_i, t_i)$ de $\phi \mid K$, nous construisons un ensemble de clauses de Horn C_{L_i} définissant un prédicat $\widetilde{L_i}$ tel que :

1. le plus petit modèle de Herbrand $\mathcal{H}_{P,\phi}$ de $\mathcal{C}_P \cup \mathcal{C}_{\neq} \cup \bigcup_i C_{L_i}$ contienne $\mathcal{H}_P$;
2. pour toute substitution σ et toute substitution θ telle que $\mathcal{I}_S, \theta \models K$, on vérifie $\mathcal{H}_P \not\models L_i\theta\sigma$ si et seulement si $\mathcal{H}_{P,\phi} \models \widetilde{L_i}\theta\sigma$;
3. $\mathcal{C}_P \cup \bigcup_i C_{L_i}$ est admissible.

Soient $x_1^i, \ldots, x_n^i$ les variables de u_i. On rappelle que, par hypothèse, les x_j^i sont de sorte Num ou Ha. On commence par construire C_{L_i} en utilisant les techniques de complémentations décrites dans [Com88] :

$$\begin{array}{rcll} & \Rightarrow & \widetilde{L_i}(x_1^i, \ldots, x_n^i, \perp) & \mid K \\ \widetilde{L_i}(x_1^i, \ldots, x_n^i, t), \mathit{Diff}(u_i, y) & \Rightarrow & \widetilde{L_i}(x_1^i, \ldots, x_n^i, y \cdot t) & \mid K. \end{array}$$

Ces clauses utilisent un nouveau prédicat *Diff* dont l'interprétation dans le plus petit modèle de Herbrand est exactement l'ensemble des couples de termes distincts. Un tel prédicat peut être défini par un ensemble de clauses de Horn $\mathcal{C}_{\mathit{Diff}}$.

Construisons également l'ensemble de clauses $\mathcal{C}_{\mathsf{In}}$ définissant le prédicat $\widetilde{\mathsf{In}}$ correspondant à la négation de In.

$$\begin{array}{rcl} & \Rightarrow & \widetilde{\mathsf{In}}(x, \perp) \\ \widetilde{\mathsf{In}}(x, t), \mathit{Diff}(x, y) & \Rightarrow & \widetilde{\mathsf{In}}(x, y \cdot t). \end{array}$$

Soit $\mathcal{H}'$ le plus petit modèle de Herbrand de $\mathcal{C}_P \cup \mathcal{C}_{\neq} \cup \mathcal{C}_{\mathsf{In}} \cup \mathcal{C}_{\mathit{Diff}}$. Le prédicat In est déterministe, donc d'après la proposition 6.1 et les résultats de [Com88], pour toute substitution close σ, on obtient $\mathcal{H}' \not\models \mathsf{In}\sigma$ si et seulement si $\mathcal{H}' \models \widetilde{\mathsf{In}}\sigma$.

Soit maintenant $\mathcal{H}_{P,\phi}$ plus petit modèle de Herbrand de $\mathcal{C}_P \cup \mathcal{C}_{\neq} \cup \bigcup_i C_{L_i} \cup \mathcal{C}_{\mathit{Diff}}$. La condition 1 est vérifiée. De plus, pour toute substitution σ et toute substitution θ telle que $\mathcal{I}_S, \theta \models K$, le littéral $L_i\theta\sigma$ est égal à $\mathsf{In}\sigma'$ pour une certaine substitution σ' et pour cette même substitution : $\widetilde{L_i}\theta\sigma = \widetilde{\mathsf{In}}\sigma'$. On en déduit que la condition 2 est vérifiée.

Cependant, la définition en clauses de Horn du prédicat *Diff* ne satisfait pas la dernière condition. On remplace donc le prédicat *Diff* par le prédicat Diff' défini par l'ensemble de clauses $\mathcal{C}_{\mathit{Diff}'}$ suivant :

$$\begin{array}{rcll} & \Rightarrow & \mathit{Diff}'(f(x_1, \ldots, x_n), g(y_1, \ldots, y_k)) & \forall f \neq g, f, g \notin \{s_h, s_d\} \\ \mathit{Diff}'(x_i, y_i) & \Rightarrow & \mathit{Diff}'(f(x_1, \ldots, x_n), f(x_1, \ldots, x_n)) & 1 \leqslant i \leqslant n, f \notin \{s_h, s_d\} \\ \mathsf{Distinct}(x, y) & \Rightarrow & \mathit{Diff}(x, y)) & \mid \mathsf{Agent}(x) \wedge \mathsf{Agent}(y) \end{array}$$

On note C'_{L_i} l'ensemble des clauses obtenu à partir de C_{L_i} en remplaçant le prédicat *Diff* par Diff' et $\widetilde{L_i}$ par $\widetilde{L_i}'$. Les ensembles $\mathcal{C}_{\mathit{Diff}'}$ et C'_{L_i} forment cette fois des ensembles admissibles. On note $\mathcal{H}_{P,\phi}$ le plus petit modèle de Herbrand associé à $\mathcal{C}_P \cup \mathcal{C}_{\neq} \cup \bigcup_i C_{L_i} \cup \mathcal{C}_{\mathit{Diff}}$ et $\mathcal{H}'_{P,\phi}$ le plus petit modèle de Herbrand associé à $\mathcal{C}_P \cup \mathcal{C}_{\neq} \cup \bigcup_i C'_{L_i} \cup \mathcal{C}_{\mathit{Diff}'}$.

Comme les interprétations de *Diff* et de Diff' coïncident sur les termes dont les sous-termes maximaux de sorte Agent sont des termes de sorte Ha, les interprétations de $\widetilde{L_i}$ et de $\widetilde{L_i}'$ coïncident également pour des substitutions σ telles que les sous-termes de sorte Agent de $u_i\sigma$ sont des termes de sorte Ha. En particulier, d'après le choix des u_i, pour toute substitution σ et toute substitution θ telle que $\mathcal{I}_S, \theta \models K$, le littéral $\widetilde{L_i}\theta\sigma$ est dans $\mathcal{H}_{P,\phi}$ si et seulement si $\widetilde{L_i}'\theta\sigma$ est dans $\mathcal{H}'_{P,\phi}$, ce qui termine la démonstration. □

6.1.4 Applications

Un tel résultat de réduction a des applications pratiques. En effet, lorsqu'on cherche à montrer à l'aide d'un outil (comme ceux présentés au paragraphe 7.2.3), qu'un protocole est sûr pour un nombre fixé de sessions, il faut instancier les rôles pour chaque session. Notre résultat de réduction permet de limiter l'espace des recherches. En effet, considérons k sessions d'un protocole à r rôles (en général r est égal à 2 ou 3), il y a alors au plus rk participants au protocole. Au lieu de considérer les $(rk)^{rk}$ possibilités théoriques pour les différentes instanciations des sessions, il suffit de considérer :

- au plus 2^{rk} instanciations distinctes si un agent peut se parler à lui-même,
- au plus $(r+1)^{rk}$ instanciations distinctes si un agent ne peut pas se parler à lui-même.

Notons que la plupart des outils utilisaient cette simplification sans la justifier. De plus, on peut en fait se restreindre à un nombre plus faible d'instanciations en remarquant que le jeu d'une règle par un agent malhonnête n'apprend rien à l'intrus et en enlevant les instanciations identiques à permutation des sessions près.

Cette réduction justifie également *a posteriori* les abstractions de certains résultats théoriques. Nous allons commenter les trois exemples [Bla01, Pau97a, BLP03] mentionnés dans l'introduction de ce chapitre. Dans l'abstraction proposée par Y. Lakhnech dans [BLP03], une session est particularisée. Dans cette session, un même agent joue tous les rôles de la session et tous les autres participants des autres sessions sont identifiés à l'intrus. De cette façon, seules deux identités sont considérées, l'une honnête et l'autre malhonnête. Notre résultat permet ainsi de montrer que cette partie de l'abstraction est correcte, du moins si on suppose qu'un agent peut se parler à lui-même.

D'autre part, L. Paulson [Pau97a] prouve que le protocole (typé) d'Otway-Rees est secret pour deux participants honnêtes et un intrus et il déclare que la preuve pourrait être généralisée, par induction, à un nombre arbitraire d'agents mais qu'une telle preuve demanderait beaucoup d'effort. La preuve du secret du protocole d'Otway-Rees pour un nombre arbitraire d'agents est maintenant un simple corollaire du résultat de L. Paulson et de notre résultat de réduction.

Un dernier exemple concerne les travaux de B. Blanchet [Bla01] où seules deux identités honnêtes sont considérées, ce qui peut maintenant être justifié par notre résultat.

Intrus passif. Notre résultat de réduction permet également d'obtenir un résultat de décidabilité dans le cadre d'un intrus passif. Un tel intrus capte et analyse les messages circulant sur le réseau mais il n'en envoie jamais de nouveaux. Dans ce contexte, on peut supposer que les agents ne peuvent pas confondre des messages provenant de sessions différentes : il suffit d'ajouter à chaque message le numéro de la règle jouée et un nonce caractérisant la session. L'intrus ne peut alors pas modifier ces en-têtes. Il n'y a alors plus lieu de considérer des entrelacements de sessions, et notre résultat de réduction assure qu'il suffit de considérer un nombre fini de participants. De plus, étant donné un ensemble de messages S et un message m, on peut décider en temps polynomial si l'intrus est capable de déduire m de S. Ce résultat est une conséquence des travaux de D. Mac Allester [All93]. Ces différents arguments permettent de conclure que le secret est décidable en $\mathsf{EXP}(r) \times \mathsf{PTIME}$ où r est le nombre de rôles du protocole considéré.

6.2 De la difficulté à borner le nombre de sessions

Notre résultat de réduction sur le nombre de participants ne permet cependant pas de déduire un quelconque résultat de décidabilité dans le cadre d'un intrus actif et d'un nombre non borné de sessions.

Pendant un temps, nous avons cherché à obtenir un résultat similaire sur le nombre de sessions à considérer, dans un cadre restreint. En effet, la plupart des attaques connues sur les protocoles, comme celles présentées dans [CJ97], n'utilisent que très peu de sessions (souvent une ou deux).

Un résultat de réduction sur les sessions permettrait d'obtenir des résultats de décidabilité dans le cadre d'un nombre *a priori* non borné de sessions. D'autre part, ce résultat serait immédiatement exploitable par les outils qui vérifient les protocoles pour un nombre fixé de sessions, comme ceux décrits au paragraphe 7.2.3. Il est bien sûr impossible de borner le nombre des sessions *en général* puisque, par exemple, le secret est indécidable.

L'objet de cette partie est de présenter d'abord quelques résultats négatifs concernant la réduction du nombre de sessions afin de situer le cadre dans lequel il faudrait chercher une telle réduction. Nous résumons ensuite quelques travaux existants sur la réduction du nombre de sessions.

6.2.1 Nombre de sessions parallèles

Nous commençons par définir informellement les sessions parallèles. Une session est dite *ouverte* s'il existe un agent qui a joué au moins une règle de la session et qui n'a toujours pas terminé le protocole pour cette session. On dit que k *sessions parallèles* sont nécessaires à une attaque si pour toute attaque, il existe un moment de son déroulement tel qu'au moins k sessions soient ouvertes en même temps. Borner le nombre de sessions parallèles nécessaires est un problème qui a des implications sur la décidabilité du secret ou l'authentification. Ainsi, R. Amadio et W. Charatonik [AC02] ont montré que si l'on suppose que le nombre de sessions parallèles est borné, alors le problème de l'accessibilité est décidable sous réserve de fortes restrictions sur les variables copiées, restrictions que nous n'expliciterons pas ici.

Notons que borner le nombre de sessions parallèles ne suffit pas à borner le nombre total de sessions car on peut encore avoir besoin d'un nombre arbitraire de sessions en séquence.

Il est tout de même impossible de borner *a priori* le nombre de sessions parallèles nécessaire à une attaque de secret par exemple. Considérons ainsi une variante du protocole considéré au chapitre 3, paragraphe 3.5. Nous reprenons en particulier les notations introduites dans ce paragraphe. Soit $(u_i, v_i)_{1 \leqslant i \leqslant n}, u_i, v_i \in \Sigma^*$ une entrée du problème de correspondance de Post. On construit le protocole suivant :

$$\begin{array}{rcll}
A & \rightarrow & B : \{< 0, 0 >, 0\}_{K_{ab}} & \\
B & \rightarrow & A : \{< 0, 0 >, N_b\}_{K_{ab}} & \\
A : \{< 0, 0 >, z\}_{K_{ab}} & \rightarrow & B : \{< 0, 0 >, \{z\}_{N_a}\}_{K_{ab}} & \\
B & \rightarrow & A : \{< 0, 0 >, \{z\}_{N_a}\}_{K_{ab}} & \\
A : \{< x, y >, \{z\}_{N_a}\}_{K_{ab}} & \rightarrow & B : \{< xu_i, yv_i >, z'\}_{K_{ab}}, \{s\}_{\{< xu_i, xu_i >, 0\}_{K_{ab}}} & 1 \leqslant i \leqslant n
\end{array}$$

Ce protocole code également le problème de correspondance de Post mais un « compteur » est ajouté : tous les messages échangés entre deux agents honnêtes sont de la forme $\{< u, v >, t\}_{K_{ab}}$, un intrus ne peut ni les déchiffrer ni les modifier. De plus le terme t représente toujours un nonce chiffré successivement par plusieurs autres nonces. On note n_t le nombre de chiffrements utilisés pour construire t.

La deuxième règle de A consiste à incrémenter le nombre de chiffrements. Cette deuxième règle ne peut être jouée que si les termes u et v sont tous les deux égaux à zéro, c'est-à-dire si l'on n'a pas encore commencé à résoudre le problème de Post. Inversement, pour que l'une des dernières règles de A soit jouée, il faut pouvoir décrémenter d'un le nombre de chiffrements utilisés pour t. Pour simuler la résolution du problème de correspondance de Post, il faut ainsi :

- commencer par ouvrir exactement autant de sessions entre A et B que de mots nécessaires pour construire une solution,
- puis jouer successivement dans chaque session la deuxième règle de A en renvoyant dans la session $i+1$ le message obtenu de A dans la session i,
- et enfin jouer successivement, mais à rebours de la dernière à la première session ouverte, la règle de A correspondant au couple de mots à ajouter pour construire la solution.

On peut montrer par induction que, d'une part, le secret s est révélé si et seulement si $(u_i, v_i)_{1\leqslant i\leqslant n}$ est une solution du problème de correspondance de Post et que d'autre part, une attaque nécessite n sessions parallèles où n est le nombre d'éléments de $(u_i, v_i)_{1\leqslant i\leqslant n}$ utilisés pour construire le plus petit mot w solution :

$$w = u_{\phi(1)} \cdots u_{\phi(n)} = v_{\phi(1)} \cdots v_{\phi(n)}.$$

Nous déduisons qu'il n'est pas possible de borner le nombre de sessions parallèles nécessaires à une attaque pour des protocoles arbitraires. Notre raisonnement reste assez informel mais nous pensons qu'il peut se traduire formellement dans la plupart des modèles de protocoles cryptographiques. Notons qu'on pourrait également construire un protocole qui n'utilise pas de clefs composées en reprenant la transformation présentée au paragraphe 3.5.

6.2.2 Indécidabilité du secret pour une profondeur bornée des messages

Nous verrons au chapitre 8 qu'il est possible d'obtenir un résultat de décidabilité pour un nombre non borné de sessions en considérant des protocoles sans nonces et tels que chaque règle demande au plus une copie arbitraire de message. Cette dernière hypothèse sera explicitée au chapitre 8.

Nous avons cherché à étendre ce résultat aux protocoles avec nonces mais les résultats de [DLMS99] et [AC02] montrent qu'il faut encore restreindre le cadre des recherches. En effet, J. Mitchell *et al.* [DLMS99] montrent que le secret est indécidable pour un nombre non borné de sessions, même si l'on borne la profondeur des messages échangés. Ce résultat indique que même lorsque les copies ne concernent pas des messages arbitraires mais des messages de profondeur bornée, le nombre de sessions ne peut pas être borné. Cependant, le codage présenté dans cet article utilise plusieurs copies dans une même règle.

Les travaux de R. Amadio et W. Charatonik [AC02] raffinent ce résultat. Ils montrent en effet que le secret est indécidable en codant une machine à deux compteurs tout en bornant la profondeur des messages et en n'utilisant la paire qu'en tête des messages : les messages sont de la forme $< m_1, \ldots, m_k >$ où chaque m_i est de la forme $\{\ldots\{\{N\}_{K_1}\}_{\ldots}\}_{K_n}$. De plus, pour chaque règle, les seuls messages copiés sont des nonces : on peut se restreindre à des variables de type nonce.

On en déduit que dans le même cadre – messages de profondeur bornée, variables de type nonce – il est impossible de borner le nombre des sessions à considérer pour trouver une attaque de secret.

6.2.3 Résultats de réduction

Certains résultats de réduction existent cependant, mais toujours dans un cadre assez restreint.

Tout d'abord, G. Lowe [Low98] a montré que pour le secret, on peut se restreindre à l'étude des protocoles tels que les rôles sont joués chacun au plus un fois, par des agents distincts et honnêtes et tels que, de plus, aucun agent honnête ne parle à un agent malhonnête. Pour cela, les protocoles considérés doivent satisfaire un certain nombre d'hypothèses dont nous décrivons les principales.

- Les messages doivent être entièrement typés, tous les types sont atomiques (chaque type représente un ensemble de constantes) et disjoints. Par conséquent, les messages reçus (et envoyés) par les agents sont entièrement spécifiés, à renommage des noms près.
- Les clefs sont atomiques ou de la forme $\mathsf{shr}(a)$ ou $\mathsf{pub}(a)$: les clefs composées ne sont pas représentées.
- Les seuls symboles de fonctions applicables à des messages sont la paire et le chiffrement.
- Les secrets « à long terme » (comme les clefs privées) doivent être toujours en position de clefs.
- Il ne doit pas y avoir de confusion possible entre des messages provenant d'étapes différentes du protocole.
- Si une valeur n'est pas déclarée secrète, alors l'intrus la connaît dès qu'elle est créée : pas de secret temporaire.

A.W. Roscoe et P.J. Broadfoot [RB99] ont cherché à borner le nombre maximal de nonces distincts nécessaires pour trouver une attaque. En effet, si le nombre de nonces distincts est borné, on peut plus facilement vérifier le protocole à l'aide des models checkers comme FDR [Low96, LR97]. Leur approche est la suivante : à tout protocole, ils associent un protocole transformé où les nonces correspondants à des sessions terminées pour les participants (et donc « oubliés » par ces participants) sont projetés sur un unique nonce. Si on suppose de plus que chaque agent ne joue qu'un nombre borné de sessions parallèles, le protocole transformé n'utilise plus qu'un nombre fini de nonces. Ils montrent que leur transformation est correcte : si le protocole transformé vérifie la propriété de sécurité alors le protocole initial la vérifie également, mais elle n'est pas complète. De plus, pour pouvoir appliquer leur résultat, il faut commencer par borner *a priori* le nombre de sessions parallèle puis développer un procédé de vérification adapté au protocole transformé qui renomme les nonces au fur et à mesure. Il faut donc utiliser un outil « adapté » à leur transformation ce qui n'est pas le cas de tous les outils.

S.D. Stoller [Sto01, Sto02] a développé une condition suffisante qui permet de conclure qu'il suffit de considérer des protocoles où chaque rôle n'est joué qu'un nombre borné de fois en même temps. Il considère une fonction f de l'ensemble des rôles dans les entiers, qui à chaque rôle r associe le nombre maximal de jeux du rôle r dans des sessions simultanées. À toute fonction f et protocole Π, il associe une autre fonction $\beta(f,\Pi)$ calculée en fonction de f et de Π. Dans les grandes lignes, son résultat peut s'énoncer ainsi : si pour tout ensemble de messages H obtenu en ne jouant chaque rôle r qu'au plus $\beta(f,\Pi)(r)$ fois, H peut également être obtenu en ne jouant chaque rôle r qu'au plus $f(r)$ fois, alors le protocole Π vérifie la propriété ϕ (d'authentification ou de secret) si et seulement si le protocole Π tel que chaque rôle r n'est joué qu'au plus $f(r)$ fois vérifie la propriété ϕ. L'intérêt de ce résultat est que l'on peut vérifier la condition suffisante mécaniquement. Il reste alors à chercher récursivement une fonction f qui convienne. Cependant, certains protocoles peuvent ne jamais vérifier la condition suffisante quelle que soit la fonction f choisie.

J. Heather et S. Schneider [HS00] ont introduit une notion de *fonction de rang* qui représente un invariant pour la connaissance de l'intrus et permet de prouver qu'un protocole vérifie une propriété de secret ou d'authentification. Pour faciliter la construction d'une telle fonction, ils suggèrent de se limiter à quatre agents (trois honnêtes et un malhonnête) et deux nonces par agents. Ils montrent que cette restriction est correcte mais n'est bien sûr pas complète.

CHAPITRE 7

Securify : un outil de vérification du secret

En 1999, la plupart des logiciels permettant la vérification des protocoles consistaient en la recherche d'attaques [Low97a, MMS97, Son99, JRV00] et permettaient ainsi de détecter les failles des protocoles. Mais aucun de ces logiciels ne fournissait une preuve de la correction d'un protocole dans le cadre d'un nombre non borné de sessions. En effet, une preuve de sécurité nécessite de développer un cadre convaincant pour modéliser les règles des protocoles et, plus encore, pour exprimer les propriétés de sécurité recherchées. Encore actuellement, il n'existe pas de consensus sur les définitions possibles de l'authentification même si plusieurs solutions ont été proposées [Sch97, Low97b, Syv94, AFG00].

A l'heure actuelle, plusieurs outils prouvant le secret ont été développés. Nous présentons ici une procédure qui a été conçue après une collaboration au SRI et qui a fait l'objet de l'article [CMR01].

Le point de départ de cette procédure est la généralisation des preuves faites en PVS et tirées de [MR00] pour les protocoles d'Otway-Rees et Needham-Schroeder-Lowe. Concrètement, nous partons d'un protocole exprimé dans le modèle Millen-Rueß et nous cherchons à montrer inductivement que l'application d'une règle du protocole ne compromet aucun secret. Pour ce faire, nous avons mis en évidence trois conditions suffisantes et une procédure de recherche en arrière : si aucun de nos lemmes « de base » ne permet de conclure au secret, on recherche alors toutes les instances des règles qui ont pu engendrer une partie des prémisses de la règle déjà considérée, et ainsi de suite. Il est à noter que le secret est indécidable pour la classe de protocoles considérée. La méthode de preuve ainsi obtenue ne peut donc pas être complète.

Cette procédure a été implémentée en *Ocaml* puis l'outil ainsi obtenu a été intégré au projet RNTL EVA [Cor02a]. Les comportements de l'outil sont de trois types :

- réponse « oui » : le secret des données à protéger est garanti dans le modèle Millen-Rueß,
- réponse « échec » : la méthode de preuve échoue, on ne peut pas conclure si le protocole est sûr ou non. Cependant, l'arbre d'échec de la preuve peut assez souvent permettre de construire une attaque et de montrer ainsi que le protocole n'est pas correct.
- pas de réponse : l'outil ne termine pas. Ce cas est rare sur les protocoles fortement typés provenant de spécifications réelles comme ceux décrits par J. Clark et J. Jacob [CJ97] mais nous verrons au paragraphe 7.1.5 des exemples de protocoles sur lesquels la pro-

cédure ne termine pas.

Dans ce chapitre, nous commençons par décrire la procédure de preuve, nous prouvons sa correction puis discutons ses limites à l'aide de plusieurs exemples. Dans un deuxième temps, nous exposons les résultats obtenus après implémentation de la procédure et nous les comparons à ceux d'autres outils également dédiés à la preuve du secret des protocoles pour un nombre non borné de sessions.

7.1 L'algorithme

Comme annoncé dans l'introduction, les preuves de secret sont effectuées dans le cadre du modèle de Millen-Rueß. Ces preuves reposent sur le théorème de J. Millen et H. Rueß , déjà décrit au chapitre 4, qui permet de découper la preuve en deux parties, l'une indépendante du protocole et l'autre dépendante du protocole. Nous rappelons ici ce théorème :

Théorème (théorème 4.1)
Soit P un protocole. P est impénétrable si et seulement si P préserve la confidentialité.

Exemple 7.1 *Nous prenons comme premier exemple le protocole d'Otway-Rees, décrit figure 7.1. Le cas du protocole de Yahalom, utilisé aux chapitres 3 et 4 sera étudié plus tard dans ce chapitre.*

Pour montrer que le protocole d'Otway-Rees préserve la confidentialité, il suffit de vérifier que :

- *pour toute connaissance initiale I,*
- *pour tout historique accessible, et*
- *pour toute spécification $C \in H$, C compatible avec I,*

alors pour toute transition applicable :

$$\mathsf{Msg}(Post(t)) \subseteq Coideal(Sec(C)).$$

Pour les transitions qui sont des instances de la règle 7.5, la vérification est immédiate. Pour les instances $t = rl_2\theta$ de la règle 7.6, notée rl_2, il faut alors vérifier que si :

1. *$\{N_a\} \ddagger \{A, B\}\theta \;\in H$,*
2. *$\mathsf{A}_{1,1}(A, B, Srv)\theta \;\in H$, et*
3. *$N\theta \in Frais(H)$,*

alors $< N, A, \{N_a, N, B\}_{\mathsf{shr}(A)} > \theta \in Coideal(Sec(C))$.

En appliquant la définition d'un idéal, $< N, A, \{N_a, N, B\}_{\mathsf{shr}(A)} > \theta \in Coideal(Sec(C))$ si et seulement si : $N\theta \in Coideal(Sec(C))$, $A\theta \in Coideal(Sec(C))$ et $\{N_a, N, B\}_{\mathsf{shr}(A)}\theta \in Coideal(Sec(C))$. Le deuxième cas est immédiat car les agents ne sont pas éléments des idéaux. Le premier cas est vérifié car $N\theta$ est frais *dans H. Pour le troisième cas, on applique à nouveau la définition : $\{N_a, N, B\}_{\mathsf{shr}(A)}\theta \in Coideal(Sec(C))$ si et seulement si $\mathsf{shr}(A)\theta \in Sec(C)$ ou bien si $< N_a, N, B > \theta \in Coideal(Sec(C))$. Ce découpage en sous-cas est formalisé au paragraphe 7.1.1.*

En appliquant à nouveau les mêmes arguments, on obtient que

$$<N, A,\{N_a, N, B\}_{\mathsf{shr}(A)} > \theta \in Coideal(Sec(C)) \Leftrightarrow \mathsf{shr}(A)\theta \in Sec(C) \text{ ou } N_a\theta \in Coideal(Sec(C)).$$

Protocole informel

$$A \rightarrow B : \quad N, A, \{N_a, N, B\}_{\mathsf{shr}(A)} \tag{7.1}$$

$$B \rightarrow S : \quad N, B, A, \{N_a, N, B\}_{\mathsf{shr}(A)}, \{N_b, N, A\}_{\mathsf{shr}(B)} \tag{7.2}$$

$$A \rightarrow B : \quad N, \{N_a, K\}_{\mathsf{shr}(A)}, \{N_b, K\}_{\mathsf{shr}(B)} \tag{7.3}$$

$$B \rightarrow A : \quad N, \{N_a, K\}_{\mathsf{shr}(A)} \tag{7.4}$$

Protocole dans le modèle Millen-Rueß

$$\emptyset \xrightarrow{\{N_a, N_b, K\}} \left\{ \begin{array}{l} \{N_a\} \ddagger \{A, B\}, \{N_b\} \ddagger \{A, B\}, \\ \{K\} \ddagger \{A, B\}, \mathsf{A}_{1,1}(A, B, Srv), \\ \mathsf{B}_{2,1}(B, Srv), \mathsf{Srv}_{3,1}(Srv) \end{array} \right\} \tag{7.5}$$

$$\left\{ \ \{N_a\} \ddagger \{A, B\}, \mathsf{A}_{1,1}(A, B, Srv) \ \right\} \xrightarrow{\{N\}} \left\{ \begin{array}{l} \mathsf{A}_{1,2}(A, B, Srv, N_a), \\ < N, A, \{N_a, N, B\}_{\mathsf{shr}(A)} > \end{array} \right\} \tag{7.6}$$

$$\left\{ \begin{array}{l} < N, A, X >, \mathsf{B}_{2,1}(B, Srv), \\ \{N_b\} \ddagger \{A, B\} \end{array} \right\} \xrightarrow{\emptyset} \left\{ \begin{array}{l} \mathsf{B}_{2,2}(B, Srv, A, N_b), \\ < N, B, A, X, \{N_b, N, A\}_{\mathsf{shr}(B)} > \end{array} \right\} \tag{7.7}$$

$$\left\{ \begin{array}{l} M, \\ \mathsf{Srv}_{3,1}(Srv), \{K\} \ddagger \{A, B\} \end{array} \right\} \xrightarrow{\emptyset} \left\{ \begin{array}{l} \mathsf{Srv}_{3,2}(Srv), \\ < N, \{N_a, K\}_{\mathsf{shr}(A)}, \{N_b, K\}_{\mathsf{shr}(B)} > \end{array} \right\} \tag{7.8}$$

$$\{< N, X, \{N_b, K\}_{\mathsf{shr}(B)} >\} \xrightarrow{\emptyset} \{< N, X >\} \tag{7.9}$$

où $M \stackrel{\text{def}}{=} < N, B, A, \{N_a, N, B\}_{\mathsf{shr}(A)}, \{N_b, N, A\}_{\mathsf{shr}(B)} >$.

FIGURE 7.1 - *Protocole d'Otway-Rees.*

On remarque alors que deux cas sont possibles : soit la spécification C considérée est $\{N_a\} \ddagger \{A, B\}\theta$ soit c'est une spécification différente. Dans le premier cas,on vérifie bien $\mathsf{shr}(A)\theta \in Sec(\{N_a\} \ddagger \{A, B\})$*. Dans le deuxième cas $N_a\theta \notin Sec(C)$ car deux spécifications distinctes ne peuvent pas contenir deux nonces identiques, d'où $N_a\theta \in Coideal(Sec(C))$. La propriété est donc vérifiée pour les instances de la deuxième règle. On peut procéder de manière similaire pour les autres règles.*

La procédure que nous allons présenter consiste en une systématisation de ce type de raisonnement.

7.1.1 Décomposition en branches

D'une manière générale, on remarque qu'un message m est dans le coidéal engendré par un ensemble $Sec(C)$ si et seulement si pour chaque nonce ou clef z de $\mathsf{parts}(\{m\})$, ou bien z n'est pas dans l'ensemble $Sec(C)$, ou bien z est chiffré avec au moins une clef de $Sec(C)$.

C'est pourquoi, à chaque nonce (resp. clef) d'un message, on associe l'ensemble des clefs nécessaires au déchiffrement du nonce (resp. de la clef).

Définition 7.1 *Soient M, M_1, M_2 des messages avec variables. L'ensemble des branches de M, noté* $\mathsf{branch}(M)$ *est défini par* $\mathsf{branch}(M, \emptyset)$ *où* $\mathsf{branch}(M, K_s)$ *est lui-même défini* *inductivement de la manière suivante :*

$$
\begin{array}{rcll}
\mathsf{branch}(a, K_s) & = & \emptyset & \textit{si } a \in \textit{Agent} \\
\mathsf{branch}(z, K_s) & = & \{(z, K_s)\} & \textit{si } z \in \textit{Nonce} \cup \textit{Key} \cup \mathcal{K}_L \\
\mathsf{branch}(A, K_s) & = & \emptyset & \\
\mathsf{branch}(N, K_s) & = & \{(N, K_s)\} & \\
\mathsf{branch}(K, K_s) & = & \{(K, K_s)\} & \\
\mathsf{branch}(X, K_s) & = & \{(X, K_s)\} & \\
\mathsf{branch}(< M_1, M_2 >, K_s) & = & \mathsf{branch}(M_1, K_s) \cup \mathsf{branch}(M_2, K_s) & \\
\mathsf{branch}(\{M\}_k, K_s) & = & \mathsf{branch}(M, K_s \cup \{k^{-1}\}) & \textit{si } k \in \textit{Key} \cup \mathcal{K}_L \\
\mathsf{branch}(\{M\}_K, K_s) & = & \mathsf{branch}(M, K_s \cup \{K\}) & \\
\mathsf{branch}(\{M\}_{\mathsf{pub}(A)}, K_s) & = & \mathsf{branch}(M, K_s \cup \{\mathsf{prv}(A)\}) & \\
\mathsf{branch}(\{M\}_{\mathsf{prv}(A)}, K_s) & = & \mathsf{branch}(M, K_s \cup \{\mathsf{pub}(A)\}) & \\
\mathsf{branch}(\{M\}_{\mathsf{shr}(A)}, K_s) & = & \mathsf{branch}(M, K_s \cup \{\mathsf{shr}(A)\}) & \\
\mathsf{branch}(\mathsf{h}(M), K_s) & = & \emptyset &
\end{array}
$$

L'opérateur $\mathsf{branch}(\_)$ *est étendu aux ensembles de messages de manière canonique : soit* S *un ensemble de messages,* $\mathsf{branch}(S) = \bigcup_{m \in S} \mathsf{branch}(m)$.

Intuitivement $\mathsf{branch}(M)$ désigne un ensemble de paires (m, K_s), où m est un sous-terme de M, telles que la connaissance de m est assujettie à celle de M et des éléments de K_s. Les messages de K_s « gardent » le message m dans M. Nous rappelons que l'ensemble $\mathcal{K}_L$ désigne l'ensemble des clefs publiques, symétriques ou partagées des agents, c'est-à-dire l'ensemble des clefs à long terme.

Exemple 7.2 *On reprend le message* $< N, A, \{N_a, N, B\}_{\mathsf{shr}(A)} >$ *considéré au paragraphe précédent :*

$$\mathsf{branch}(< N, A, \{N_a, N, B\}_{\mathsf{shr}(A)} >) = \{(N, \emptyset), (N_a, \{\mathsf{shr}(A)\}), (N, \{\mathsf{shr}(A)\})\}.$$

On montre alors qu'un message m est dans le coidéal d'un ensemble de secrets basiques S si et seulement si pour chacune de ses branches, ou bien le nonce ou la clef de la branche n'est pas l'un des secrets de S, ou bien il est chiffré par au moins une clef dont l'inverse est dans S, c'est-à-dire s'il est gardé par au moins une clef de S.

Proposition 7.1 *Soient* S *un ensemble de nonces et de clefs :* $S \subset \textit{Nonce} \cup \textit{Key} \cup \mathcal{K}_L$ *et* m *un message (clos) :*
$m \in \textit{Coideal}(S)$ *si et seulement si, pour toute branche* $(z, K_s) \in \mathsf{branch}(m)$ *:*

$$z \notin S \textit{ ou } K_s \cap S \neq \emptyset.$$

Preuve. La propriété se démontre par induction sur la structure des messages.

1. Si $m \in \textit{Agent}$, alors $\mathsf{branch}(m) = \emptyset$ et $m \in \textit{Coideal}(S)$ donc la propriété est vraie.
2. Si $m \in \textit{Nonce} \cup \textit{Key} \cup \mathcal{K}_L$, alors $\mathsf{branch}(m) = \{(m, \emptyset)\}$ et $m \in \textit{Coideal}(S)$ si et seulement si $m \notin S$, la propriété est vérifiée.
3. Si $m =< m_1, m_2 >$, alors $\mathsf{branch}(< m_1, m_2 >) = \mathsf{branch}(m_1) \cup \mathsf{branch}(m_2)$. En appliquant la définition du coidéal, $m \in \textit{Coideal}(S)$ si et seulement si $m_1 \in \textit{Coideal}(S)$ et $m_2 \in \textit{Coideal}(S)$. Par induction, on obtient que $m \in \textit{Coideal}(S)$ si et seulement si pour toute branche $(z, K_s) \in \mathsf{branch}(m_1)$ et pour toute branche $(z, K_s) \in \mathsf{branch}(m_2)$, $z \notin S$ ou $K_s \cap S \neq \emptyset$, ce qui démontre la propriété.

4. Si $m = h(m')$, alors $\mathsf{branch}(m) = \emptyset$ et $m \in Coideal(S)$; la propriété est vraie.
5. Enfin, si $m = \{m'\}_k$, alors

$$\mathsf{branch}(m) = \bigcup_{(z,K_s)\in \mathsf{branch}(m')} (z, K_s \cup \{k^{-1}\}).$$

De plus, $m \in Coideal(S)$ si et seulement si $k^{-1} \in \mathcal{I}(S)$ ou $m' \in Coideal(S)$, c'est-à-dire $k^{-1} \in S$ ou $m' \in Coideal(S)$. Par équivalences successives :

$$\begin{array}{rcl} m \in Coideal(S) & \Leftrightarrow & k^{-1} \in S \vee m' \in Coideal(S) \\ & \Leftrightarrow & k^{-1} \in S \vee \forall (z, K'_s) \in \mathsf{branch}(m')[z \notin S \vee K'_s \cap S \neq \emptyset] \\ & \Leftrightarrow & \forall (z, K'_s) \in \mathsf{branch}(m')[z \notin S \vee K_s \cap S \neq \emptyset \vee k^{-1} \in S] \\ & \Leftrightarrow & \forall (z, K'_s) \in \mathsf{branch}(m')[z \notin S \vee (K'_s \cup \{k^{-1}\}) \cap S \neq \emptyset] \\ & \Leftrightarrow & \forall (z, K_s) \in \mathsf{branch}(m)[z \notin S \vee K_s \cap S \neq \emptyset], \end{array}$$

ce qui démontre la propriété. □

La propriété énoncée ici est valable pour les messages clos. Cependant si M est un message avec variables et θ une substitution bien sortée alors

$$\mathsf{branch}(m\theta) = \mathsf{branch}(m)\theta.$$

On en déduit la proposition suivante.

Corollaire 7.1 *Soit S un ensemble de nonces, de clefs ou de variables à valeurs dans $Nonce$ ou Key. Soit M un message avec variables.*

$M\theta \in Coideal(S\theta)$ pour toute substitution bien sortée θ si et seulement si, pour toute branche $(Z, K_s) \in \mathsf{branch}(M)$: $Z \notin S$ ou $K_s \cap S \neq \emptyset$.

On définit maintenant une propriété sur les branches de manière à ce qu'un protocole préserve la confidentialité si et seulement si cette propriété est vérifiée pour l'ensemble des branches associées aux règles de ce protocole. Il est tout d'abord utile d'introduire la notion de P-configuration.

Définition 7.2 *Soit P un protocole. Soient I un ensemble de messages (représentant la connaissance initiale de l'intrus), H un historique et C une spécification. On dit que (I, H, C) est une P-*configuration *si H est un historique accessible : $H \in reachable(P, I)$ et I-impénétrable, et si C est une spécification de H, compatible avec I.*

Définition 7.3 *Soit P un protocole, $b = (z, K_s)$ une branche telle que z est un message basique et K_s un ensemble de clefs. Soient E_s un ensemble d'événements et N_s un ensemble de messages basiques. La propriété $conf(P, b)(E_s, N_s)$ est vraie si et seulement si pour toute P-configuration (I, H, C) telle que*

1. *$E_s \subseteq H$,*
2. *$N_s \subseteq Frais(H)$, et*
3. *$z \in Sec(C)$,*

on vérifie $K_s \cap Sec(C) \neq \emptyset$*.*

Pour toute transition t*, on écrit* $conf(P,b)(t)$ *au lieu de* $conf(P,b)(Pre(t), New(t))$*.*

Soient b *une branche,* E_s *un ensemble d'événements avec variables,* N_s *un ensemble de messages basiques ou de variables à valeurs dans* $Nonce$ *ou* Key*. La propriété* $conf(P,b)(E_s, N_s)$ *est vraie si pour toute substitution bien sortée* θ*,* $conf(P, b\theta)(E_s\theta, N_s\theta)$ *est vraie.*

Si rl *est une règle de* P*, on écrit* $conf(P,b)(rl)$ *au lieu de* $conf(P,b)(Pre(rl), New(rl))$*.*

La proposition 7.1 permet alors de donner une nouvelle caractérisation pour le caractère confidentiel d'un protocole.

Proposition 7.2 *Un protocole* P *préserve la confidentialité si et seulement si*

1. *pour toute règle* rl *de* P*,*
2. *pour tout message* M *de* $Post(rl)$*,*
3. *pour toute branche* b *de* $\mathsf{branch}(M)$*,*

la propriété $conf(P,b)(rl)$ *est vraie.*

Preuve. Supposons tout d'abord que le protocole P préserve la confidentialité. Soit rl une règle de P, M un message de $Post(rl)$, et $b = (Z, K_s)$ une branche de $\mathsf{branch}(M)$. Vérifions la propriété $conf(P,b)(rl)$: soient θ une substitution bien sortée, et (I,H,C) une P-configuration.

Supposons que $Pre(rl)\theta \subseteq H$, $New(rl)\theta \subseteq Frais(H)$ et $z\theta \in Sec(C)$. Alors $t = rl\theta$ est une transition applicable. Comme P préserve la confidentialité, on déduit $Msg(Post(t)) \subseteq Coideal(Sec(C))$. En particulier, $M\theta \in Coideal(Sec(C))$ et donc, d'après la propriété 7.1 et comme $b\theta \in \mathsf{branch}(M\theta)$,

$$Z\theta \notin Sec(C) \text{ ou } K_s\theta \cap Sec(C) \neq \emptyset.$$

On déduit $K_s\theta \cap Sec(C) \neq \emptyset$ et donc $conf(P, b\theta)(t)$.

Ainsi, la propriété $conf(P, b\theta)(rl\theta)$ est vérifiée pour toute substitution bien sortée θ et donc $conf(P,b)(rl)$ est vérifiée.

Réciproquement, supposons que pour toute règle rl de P, pour tout message M de $Post(rl)$, et pour toute branche b de $\mathsf{branch}(M)$, la propriété $conf(P,b)(rl)$ est vraie. Montrons que P préserve la confidentialité.

Soit (I,H,C) une P-configuration. Soit t une transition applicable. Montrons que

$$Msg(Post(t)) \subseteq Coideal(Sec(C)).$$

Soit $m \in Post(t)$. D'après la propriété 7.1, $m \in Coideal(Sec(C))$ si et seulement si pour toute branche $(z, K_s) \in \mathsf{branch}(m)$,

$$z \notin Sec(C) \text{ ou } K_s \cap Sec(C) \neq \emptyset.$$

Or $t = rl\theta$ pour une certaine règle rl de P et pour une substitution bien sortée θ. En particulier $m = M\theta$ pour un certain message avec variables $M \in Post(rl)$. Soit $(z, K_s) \in \mathsf{branch}(m)$, $(z, K_s) = (Z, K'_s)\theta$ pour une certaine branche $(Z, K'_s) \in \mathsf{branch}(M)$. Par hypothèse, la propriété $conf(P,(z,K_s))(rl)$ est vraie. On déduit $z \notin Sec(C)$ ou $K_s \cap Sec(C) \neq \emptyset$, et donc $m \in Coideal(Sec(C))$.

On a vérifié que P préserve la confidentialité. □

7.1.2 Tests élémentaires

L'objet de ce paragraphe est de définir des conditions suffisantes sur les branches pour que la propriété $conf(P,b)(rl)$ soit vérifiée.

Proposition 7.3 (tests élémentaires) *Soit $b = (Z, K_s)$ une branche, E_s un ensemble d'événements avec variables et N_s un ensemble de messages basiques ou de variables à valeurs dans Nonce ou Key. La propriété $conf(P,b)(E_s, N_s)$ est vraie dès que l'une des propositions suivantes est vérifiée.*

1. *$Z \in N_s$.*
2. *Il existe un ensemble de clefs ou de variables de clefs K'_s tel que $K'_s \subseteq K_s$ et $(Z, K'_s) \in \mathsf{branch}(Msg(E_s))$; dans ce cas, on écrit $(Z, K_s) \tilde{\in} \mathsf{branch}(Msg(E_s))$.*
3. *Il existe une spécification $C \in E_s$ telle que $Z \in SpecSec(C)$ et $K_s \cap Sec(C) \neq \emptyset$; dans ce cas, on écrit $ds(E_s, Z, K_s)$ (pour « secrets disjoints »).*

L'opérateur $\tilde{\in}$ indique qu'une branche (Z, K_s) est « presque » dans l'ensemble S au sens ou la branche correspondante à Z dans S contient éventuellement moins de clefs que celles dans K_s : un secret est encore mieux protégé s'il est chiffré à l'aide d'un plus grand nombre de clefs. Les trois tests définis dans la proposition sont appelés *tests élémentaires*.

Preuve. Montrons que si l'une des trois propriétés est vraie, alors $conf(P,b)(E_s, N_s)$ est vérifié. Soit (I, H, C) une P-configuration. Soit θ une substitution bien sortée telle que
- $E_s\theta \subseteq H$;
- $N_s\theta \subseteq Frais(H)$;
- et $Z\theta \in Sec(C)$.

1. Supposons $Z \in N_s$. Alors $Z\theta \in Frais(H)$ et donc $Z\theta \notin Sec(C)$, contradiction. On en déduit que $conf(P,b)(E_s, N_s)$ est vérifié.
2. Supposons $(Z, K_s) \tilde{\in} \mathsf{branch}(Msg(E_s))$. Soit $K'_s \subseteq K_s$ tel que $(Z, K'_s) \in \mathsf{branch}(Msg(E_s))$. Comme $E_s \subseteq H$ et H est I-impénétrable, il s'ensuit que $Msg(E_s) \subseteq Coideal(Sec(C))$. On déduit en utilisant la proposition 7.1 que $K'_s \cap Sec(C) \neq \emptyset$, et donc $K_s \cap Sec(C) \neq \emptyset$.
3. Supposons $ds(E_s, Z, K_s)$. Soit C' tel que $C' \in E_s$, $Z \in SpecSec(C')$ et $K_s \cap Sec(C') \neq \emptyset$. On raisonne alors comme dans l'exemple 7.1 : ou bien $C = C'$ et $conf(P, b\theta)(E_s\theta, N_s\theta)$ est vérifié, ou bien $C \neq C'$. Par construction des règles d'un protocole du modèle Millen-Rueß, $SpecSec(C'\theta)$ et $SpecSec(C)$ sont nécessairement disjoints. De plus, $Z\theta$ est un message basique, donc $Z\theta \notin Sec(C)$, contradiction. On en déduit que la propriété $conf(P,b)(E_s, N_s)$ est vérifiée. □

Exemple 7.3 (suite de l'exemple 7.1) *Considérons la troisième règle d'Otway-Rees :*

$$or_3 = \left\{ \begin{array}{l} < N, A, X >, \mathsf{B}_{2,1}(B, Srv), \\ \{N_b\} \ddagger \{A, B\} \end{array} \right\} \xrightarrow{\emptyset} \left\{ \begin{array}{l} \mathsf{B}_{2,2}(B, Srv, A, N_b), \\ < N, B, A, X, \{N_b, N, A\}_{\mathsf{shr}(B)} > \end{array} \right\}$$

Calculons les branches du message introduit :

$$\mathsf{branch}(< N, B, A, X, \{N_b, N, A\}_{\mathsf{shr}(B)} >) = \{(N, \emptyset), (N, \{\mathsf{shr}(B)\}), (X, \emptyset), (N_b, \{\mathsf{shr}(B)\})\}.$$

Protocole informel

$$\begin{array}{lll} A & \rightarrow B : & \{A, N_a\}_{\mathsf{pub}(B)} \\ B & \rightarrow A : & \{N_a, N_b, B\}_{\mathsf{pub}(A)} \\ A & \rightarrow B : & \{N_b\}_{\mathsf{pub}(B)} \end{array}$$

Protocole dans le modèle Millen-Rueß.

$$\emptyset \xrightarrow{\{N_a, N_b\}} \left\{ \begin{array}{l} \{N_a\} \ddagger \{A, B\}, \{N_b\} \ddagger \{A, B\}, \\ \mathsf{A}_{1,1}(A, B), \mathsf{B}_{2,1}(B) \end{array} \right\} \quad (7.10)$$

$$\{\{N_a\} \ddagger \{A, B\}, \mathsf{A}_{1,1}(A, B)\} \xrightarrow{\emptyset} \{\{A, N_a\}_{\mathsf{pub}(B)}, \mathsf{A}_{1,2}(A, B, N_a)\} \quad (7.11)$$

$$\left\{ \begin{array}{l} \{N_b\} \ddagger \{A, B\}, \\ \mathsf{B}_{2,1}(B), \{A, N_a\}_{\mathsf{pub}(B)} \end{array} \right\} \xrightarrow{\emptyset} \{\{N_a, N_b, B\}_{\mathsf{pub}(A)}, \mathsf{B}_{2,2}(B, A, N_b, N_a)\} \quad (7.12)$$

$$\{\{N_a, N_b, B\}_{\mathsf{pub}(A)}, \mathsf{A}_{1,2}(A, B, N_a)\} \xrightarrow{\emptyset} \{\{N_b\}_{\mathsf{pub}(B)}, \mathsf{A}_{1,3}(A, B, N_a, N_b)\} \quad (7.13)$$

FIGURE 7.2 - *Protocole de Needham-Schroeder-Lowe .*

La propriété 2 permet de conclure pour les trois premières branches. En effet,

$$(N, \emptyset), (N, \{\mathsf{shr}(B)\}), (X, \emptyset) \mathrel{\tilde{\in}} \mathsf{branch}(< N, A, X >).$$

Pour la branche $(N_b, \{\mathsf{shr}(B)\})$*, c'est la propriété 3 qui permet de conclure. En effet,* $\{N_b\} \ddagger \{A, B\} \in Pre(rl)$*,* $N_b \in SpecSec(\{N_b\} \ddagger \{A, B\})$ *et* $\mathsf{shr}(B) \in Sec(\{N_b\} \ddagger \{A, B\})$*.*

On a donc vérifié $conf(OR, b)(or_3)$ *pour toutes les branches* b *de* $Post(or_3)$*. De la même façon, les trois tests élémentaires suffisent à montrer* $conf(OR, b)(or)$ *pour toutes les règles* or *du protocole d'Otway-Rees, noté OR, et toutes les branches* b *associées. Cela permet de conclure que le protocole d'Otway-Rees préserve la confidentialité.*

Cependant, ces tests élémentaires ne suffisent pas toujours à prouver le caractère confidentiel d'un protocole. Ainsi, considérons le protocole de Needham-Schroeder-Lowe, noté NSL, présenté à la figure 7.2. On note nsl_3 la troisième règle du protocole (règle 7.12). Alors $(N_a, \{\mathsf{prv}(A)\}) \in \mathsf{branch}(\{N_a, N_b, B\}_{\mathsf{pub}(A)})$ mais aucun des trois tests ne permet de conclure que la propriété $conf(NSL, (N_a, \{\mathsf{prv}(A)\}))(nsl2)$ est vérifiée. On introduit alors une procédure de recherche en arrière pour apporter de nouvelles informations et appliquer à nouveau les tests élémentaires.

7.1.3 Procédure de recherche

La procédure de recherche en arrière repose sur le fait que tout message présent dans un historique a été produit par le jeu d'une étape honnête ou d'une étape malhonnête. Dans les deux cas, on obtient des informations supplémentaires sur le contenu de l'historique considéré et on peut appliquer à nouveau les tests élémentaires aux événements obtenus.

La procédure est décrite à la figure 7.3. Il est à noter que les propositions de la forme $P_1 \vee P_2$ sont calculées de manière non déterministe : dès que le calcul de P_1 (ou P_2) s'arrête avec la valeur $true$, alors le calcul de $P_1 \vee P_2$ s'arrête avec la valeur $true$.

Soit b une branche, $b = (Z, K_s)$.

$$
\begin{array}{rcl}
main(P,b)(rl) & \stackrel{\text{def}}{=} & Z \in New(rl) \vee search(P,b)(Pre(rl)) \\
search(P,b)(E_s) & \stackrel{\text{def}}{=} & b \mathrel{\widetilde{\in}} \mathsf{branch}(Msg(E_s)) \ \vee ds(E_s, Z, K_s) \ \vee \ back(P,b)(E_s) \\
back(P,b)(E_s) & \stackrel{\text{def}}{=} & (\exists M \in E_s : \ Z \in \mathsf{parts}(M))[honest(P,b)(M) \wedge fake(P,b)(M)] \\
honest(P,b)(M) & \stackrel{\text{def}}{=} & (\forall rl' \in P, \exists M' \in \mathsf{parts}(Msg(Post(rl')), \exists \theta = mgu(M, M'))) \\
 & & \qquad [M\theta \in \mathsf{parts}(Msg(Pre(rl'\theta))) \vee search(P, b\theta)(Pre(rl'\theta))] \\
fake(P,b)(M) & \stackrel{\text{def}}{=} & [(\forall M_1, M_2) \\
 & & \qquad < M_1, M_2 >= M \Rightarrow \vee_{M_i}[Z \in \mathsf{parts}(M_i) \wedge search(P,b)(\{M_i\})]] \\
 & & \vee \left[(\forall M', K) \ M = \{M'\}_K) \Rightarrow search(P,b)(\{M'\})\right] \\
 & & \vee \left[(\forall M') \ M = h(M')) \Rightarrow search(P,b)(\{M'\})\right]
\end{array}
$$

FIGURE 7.3 - *Procédure de preuve du secret.*

Soit P un protocole, $b = (Z, K_s)$ une branche et rl une règle de P. Soit E_s un ensemble d'événements avec variables.

$main(P,b)(rl)$ teste si Z fait partie des données engendrées par la règle et lance la procédure principale : $search(P,b)(Pre(rl))$.

$search(P,b)(E_s)$ effectue les tests élémentaires 2 et 3 puis lance un pas de la procédure de recherche en arrière.

$back(P,b)(E_s)$ recherche toutes les manières (honnêtes ou malhonnêtes) dont les messages de E_s ont pu être engendrés. Dans chaque cas, on doit vérifier le secret : la procédure doit terminer à vrai.

$honest(P,b)(M)$ recherche toutes les règles honnêtes qui ont pu conduire à la création de M, c'est-à-dire toutes les instances de règles de P telles que M soit dans le $post$ de la règle instanciée mais non dans le pre. Encore une fois, la procédure doit terminer à vrai dans chaque cas.

$fake(P,b)(M)$ décompose M : si l'intrus a construit M, il doit posséder les sous-termes de M qui permettent de le construire.

Notation : Si $b = (Z, K_s)$ est une branche, la procédure $fake(P, Z, K_s)(M)$ représentera la procédure $fake(P,(Z,K_s))(M)$ et ainsi de suite pour toutes les fonctions utilisées par l'algorithme.

Une telle procédure est correcte.

Théorème 7.1 (Correction) *Soit P un protocole. Si $main(P,b)(t)$ est évalué à vrai*
- *pour toutes les règles rl de P,*
- *pour tous les messages avec variables M de $Post(t)$, et*
- *pour toutes les branches b de* branch(M),

alors P est impénétrable.

Ce résultat de correction est démontré à la section suivante. Par contre, la procédure n'est pas complète. Nous verrons au paragraphe 7.1.5 un exemple de protocole impénétrable pour lequel notre procédure échoue à prouver le secret. Elle permet cependant de prouver le secret dans de nombreux cas, récapitulés au paragraphe 7.2.

Exemple 7.4 (Preuve du secret du protocole de Needham-Schroeder-Lowe)
Revenons à la troisième règle du protocole de Needham-Schroeder-Lowe pour la branche $(N_a, \{\mathsf{prv}(A)\})$. *Nous appliquons la procédure. Par équivalences successives :*

$$\begin{array}{rl} & main(NSL, b)(nsl2) \\ \Longleftrightarrow & search(NSL, b)(\{\{N_b\} \ddagger \{A, B\}, \{A, N_a\}_{\mathsf{pub}(B)}, \mathsf{B}_{2,1}(B)\}) \\ \Longleftrightarrow & back(NSL, b)(\{\{N_b\} \ddagger \{A, B\}, \{A, N_a\}_{\mathsf{pub}(B)}, \mathsf{B}_{2,1}(B)\}) \\ \Longleftrightarrow & honest(NSL, b)(\{A, N_a\}_{\mathsf{pub}(B)}) \ \wedge fake(NSL, b)(\{A, N_a\}_{\mathsf{pub}(B)}) \end{array}$$

Seule la première règle du protocole contient un message de la forme $\{A, N_a\}_{\mathsf{pub}(B)}$ *dans son post, d'où :*

$$\begin{array}{rl} & honest(NSL, b)(\{A, N_a\}_{\mathsf{pub}(B)}) \\ \Longleftrightarrow & search(NSL, b)(\{\{N_a\} \ddagger \{A, B\}, \mathsf{A}_{1,1}(A, B)\}) \\ \Longleftrightarrow & true \end{array}$$

La proposition est évaluée à vrai grâce au deuxième test élémentaire.

$$\begin{array}{rl} & fake(NSL, b)(\{A, N_a\}_{\mathsf{pub}(B)}) \\ \Longleftrightarrow & search(NSL, b)(< A, N_a >) \\ \Longleftrightarrow & true \end{array}$$

car $(N_a, \{\mathsf{prv}(A)\}) \mathrel{\widetilde{\in}} \mathsf{branch}(< A, N_a >)$.

Cette dérivation peut être visualisée sous forme d'un arbre de preuve dont chaque nœud est étiqueté par la valeur courante de l'ensemble E_s *dans la procédure de preuve et chaque arête est étiquetée par la fonction intermédiaire appelée :* $honest$ *ou* $fake$ *entre deux procédures de recherche,* ds *ou* $\widetilde{\in}$ *lorsqu'un test élémentaire permet de conclure.*

L'ensemble des arbres de preuve pour chacune des branches de chaque règle du protocole est représenté à la figure 7.4. On en déduit que le protocole de Needham-Schroeder-Lowe préserve la confidentialité.

7.1.4 Correction de la procédure

Nous prouvons ici le théorème 7.1 de correction de la procédure. Pour cela, nous prouvons le lemme suivant :

Lemme 7.1 *Soit* P *un protocole,* b *une branche et* E_s *un ensemble d'événements avec variables.*

Si la proposition $search(P, b)(E_s)$ *est évaluée à vrai alors pour tout ensemble* N_s *de messages basiques ou de variables à valeurs dans* $Nonce$ *ou* Key*, alors la propriété* $conf(P, b)(E_s, N_s)$ *est vérifiée elle aussi.*

Première règle (7.10) : pas de branche, il n'y a rien à vérifier.

Deuxième règle (7.11) : $\mathsf{branch}(\{A, N_a\}_{\mathsf{pub}(B)}) = \{(N_a, \{\mathsf{prv}(B)\})\}$.

$$\begin{tikzcd}
\left\{\begin{array}{l}\{N_a\} \ddagger \{A,B\},\\ \mathsf{A}_{1,1}(A,B)\end{array}\right\} \arrow[d, dash, "ds"] \\
true
\end{tikzcd}$$

branche $(N_a, \{\mathsf{prv}(B)\})$

Troisième règle (7.12) : $\mathsf{branch}(\{N_a, N_b, B\}_{\mathsf{pub}(A)}) = \{(N_a, \{\mathsf{prv}(A)\}), (N_b, \{\mathsf{prv}(A)\})\}$.

$$\begin{tikzcd}
& \left\{\begin{array}{l}\{A, N_a\}_{\mathsf{pub}(B)},\\ \{N_b\} \ddagger \{A,B\}, \mathsf{B}_{2,1}(B)\end{array}\right\} \arrow[dl, dash, "honest"'] \arrow[dr, dash, "fake"] & \\
\left\{\begin{array}{l}\{N_a\} \ddagger \{A,B\},\\ \mathsf{A}_{1,1}(A,B)\end{array}\right\} \arrow[d, dash, "ds"] & & \{< A, N_a >\} \arrow[d, dash, "\tilde{\in}"] \\
true & & true
\end{tikzcd}$$

branche $(N_a, \{\mathsf{prv}(A)\})$

$$\begin{tikzcd}
\left\{\begin{array}{l}\{A, N_a\}_{\mathsf{pub}(B)},\\ \{N_b\} \ddagger \{A,B\}, \mathsf{B}_{2,1}(B)\end{array}\right\} \arrow[d, dash, "ds"] \\
true
\end{tikzcd}$$

branche $(N_b, \{\mathsf{prv}(A)\})$

Quatrième règle (7.13) : $\mathsf{branch}(\{N_b\}_{\mathsf{pub}(B)}) = \{(N_b, \{\mathsf{prv}(B)\})\}$.

$$\begin{tikzcd}
& \left\{\begin{array}{l}\{N_a, N_b, B\}_{\mathsf{pub}(A)}\\ \mathsf{A}_{1,2}(A,B,N_a)\end{array}\right\} \arrow[dl, dash, "honest"'] \arrow[dr, dash, "fake"] & \\
\left\{\begin{array}{l}\{N_b\} \ddagger \{A,B\},\\ \{A, N_a\}_{\mathsf{pub}(B)}\end{array}\right\} \arrow[d, dash, "ds"] & & \{N_a, N_b\} \arrow[d, dash, "\tilde{\in}"] \\
true & & true
\end{tikzcd}$$

branche $(N_b, \{\mathsf{prv}(B)\})$

FIGURE 7.4 - *Preuve du secret du protocole de Needham-Schroeder-Lowe.*

Le théorème 7.1 est alors une conséquence directe de ce lemme, du théorème 4.1 et des propositions 7.2 et 7.3.

Preuve. Pour prouver le lemme 7.1, nous allons en fait montrer une propriété plus forte.

Définition 7.4 *Soit P un protocole, $b = (z, K_s)$ une branche telle que z est un message basique et K_s un ensemble de clefs. Soit E_s un ensemble d'événements. La propriété $confstrong(P, b)(E_s)$ est vraie si et seulement si pour toute P-configuration (I, H, C) telle que :*

1. *$Nonstates(E_s) \subseteq H$ et*
2. *$z \in Sec(C)$,*

on vérifie $K_s \cap Sec(C) \neq \emptyset$.

La notation $Nonstates(E_s)$ désigne l'ensemble des événements de E_s qui ne sont pas des états.

Cette définition est étendue aux termes avec variables de la même manière que pour $conf(P, b)(E_s, N_s)$: soient b une branche et E_s un ensemble d'événements avec variables, la propriété $confstrong(P, b)(E_s)$ est vraie si pour toute substitution bien sortée θ, $confstrong(P, b\theta)(E_s\theta)$ est vraie.

Pour montrer que pour tout ensemble N_s de messages basiques ou de variables à valeurs dans $Nonce$ ou Key, la propriété $conf(P, b)(E_s, N_s)$ est vérifiée, il suffit de montrer que la propriété $confstrong(P, b)(E_s)$ est vérifiée.

Nous allons donc montrer que $search(P, b)(E_s) \Rightarrow confstrong(P, b)(E_s)$ par récurrence sur le nombre de d'appels à la fonction $back$ utilisés pour évaluer $search(P, b)(E_s)$ à vrai.

Initialisation. Si le calcul de $search(P, b)(E_s)$ termine à vrai sans qu'aucun appel à $back$ n'ait été utilisé, c'est que $(Z, K_s) \mathrel{\widetilde{\in}} \mathsf{branch}(Msg(E_s))$ ou que $ds(E_s, Z, K_s)$ est vérifié. La proposition 7.3 permet de conclure.

Récurrence. Soit $n \geqslant 0$. Supposons que $search(P, b)(E_s) \Rightarrow confstrong(P, b)(E_s)$ si le calcul de $search(P, b)(E_s)$ a utilisé au plus n appels à $back$.

Soient P, Z, K_s et E_s tels que $search(P, Z, K_s)(E_s)$ soit évalué à vrai avec $n + 1$ appels à $back$. Soit θ une substitution bien sortée. Montrons que $confstrong(P, b\theta)(E_s\theta)$ est vérifié. On considère (I, H, C) une P-configuration telle que $Nonstates(E_s)\theta \subseteq H$ et $Z\theta \in Sec(C)$.

Comme $search(P, Z, K_s)(E_s)$ est évalué à vrai, il existe un message $M \in E_s$ qui vérifie $Z \in \mathsf{parts}(M)$ et $honest(P, Z, K_s)(M) \wedge fake(P, Z, K_s)(M) = true$ et tel que le calcul de $honest(P, Z, K_s)(M)$ et $fake(P, Z, K_s)(M)$ utilise au plus n appels à $back$.

D'autre part, comme M est dans E_s, $M\theta$ est dans $E_s\theta$ et donc dans H. Une petite récurrence sur le nombre de transitions nécessaires pour atteindre H permet de montrer qu'il existe deux historiques accessibles H_1 et H_2 tels que :

$$global(P, I)(H_1, H_2),\ Nonstates(H_2) \subseteq H,\ M\theta \in \mathsf{parts}(Msg(H_2)),\ M\theta \notin \mathsf{parts}(Msg(H_1)).$$

Deux cas sont à étudier suivant que la transition de H_1 à H_2 soit honnête ou non.

Cas $honest(P, I)(H_1, H_2)$: il existe une transition t applicable telle que $H_2 = Post(t) \cup (H_1 \backslash (Pre(t) \cap States))$ et $t = rl\sigma$ pour une certaine substitution σ et $rl \in P$. Comme $M\theta$ est dans $\mathsf{parts}(Msg(H_2))$ mais non dans $\mathsf{parts}(Msg(H_1))$, on déduit que $M\theta \in$

parts$(Msg(Post(t)))$, c'est-à-dire que $M\theta$ est un sous-terme d'un certain message $M'\sigma$ avec $M' \in Post(rl)$. Supposons que $M\theta$ est sous-terme de $X\sigma$ pour X variable de M', auquel cas $M\theta$ est également sous-terme d'un message de $Pre(t)$ car il n'y a pas de nouvelles variables introduites dans le *post* d'un message. Or on a choisi H_1 tel que $M\theta \notin$ parts$(Msg(H_1))$, contradiction. On déduit que $M\theta = M''\sigma$ pour M'' sous-terme de M'. Soit $\theta' = mgu(M, M'')$. En utilisant une nouvelle fois que $M\theta \notin$ parts$(Msg(H_1))$, on déduit que $M\theta' \notin$ parts$(Pre(rl\theta'))$ et donc, comme $honest(P, Z, K_s)(M)$ est évalué à vrai, il vient que $search(P, Z, K_s\theta')(Pre(rl\theta'))$ est évalué à vrai et son calcul utilise au plus n appels à *back*.

En appliquant l'hypothèse de récurrence, on obtient que la propriété

$$confstrong(P, (Z\theta', K_s\theta'))(Pre(rl\theta'))$$

est vraie. Comme $Nonstates(Pre(rl)\theta) \in H$ et que $Z\theta \in Sec(C)$, on déduit $K_s\theta \cap Sec(C) \neq \emptyset$ et donc $confstrong(P, b\theta)(E_s\theta)$ est vérifié.

Cas $fake(P, I)(H_1, H_2)$ **:** il existe un message $m' \in$ fake$(Msg(H_1) \cup I)$ tel que $H_2 = H_1 \cup \{m'\}$. Comme $M\theta \in$ parts$(Msg(H_2))$, on obtient $M\theta \in$ parts(fake$(Msg(H_1) \cup I)$). On peut facilement vérifier que :

$$\mathsf{parts}(\mathsf{fake}(Msg(H_1) \cup I)) = \mathsf{fake}(Msg(H_1) \cup I) \cup \mathsf{parts}(Msg(H_1) \cup I) \quad .$$

Or $M\theta \notin$ parts$(Msg(H_1))$ et $M\theta \notin$ parts(I) (sinon $Z\theta \in$ parts(I) et $Z\theta \in SpecSec(C)$ ce qui contredit l'hypothèse que C est compatible avec I), donc $M\theta \notin$ parts$(Msg(H_1) \cup I)$: $M\theta$ a été synthétisé par l'intrus. M n'est pas une variable sinon $M = Z$ et alors $M\theta$ est un message basique et on peut déduire de $M\theta \in$ fake$(Msg(H_1) \cup I)$ que $M\theta \in$ parts$(Msg(H_1) \cup I)$, contradiction.

M est donc un message composé. Trois cas sont possibles :

1. $M = < M_1, M_2 >$ et $M_1\theta, M_2\theta \in$ fake$(Msg(H_1) \cup I)$,
2. ou $M = \{M'\}_K$ et $M'\theta \in$ fake$(Msg(H_1) \cup I)$,
3. ou $M =$ h(M') et $M'\theta \in$ fake$(Msg(H_1) \cup I)$.

Nous traitons le premier cas, les autres sont similaires. On pose $H' = H \cup \{M_1\theta, M_2\theta\}$. (I, H', C) est toujours une P-configuration car H' est accessible à partir de H en jouant deux transitions malhonnêtes. On peut supposer sans perte de généralité que $Z \in$ parts(M_1). Alors, comme $fake(P, Z, K_s)(M)$ est évalué à vrai, $search(P, Z, K_s)(\{M_1\})$ est évalué à vrai et son calcul utilise au plus n appels à *back*. En appliquant l'hypothèse de récurrence, on déduit que $confstrong(P, (Z, K_s))(\{M_1\})$ est vérifié. Comme $M_1\theta \in H'$ et $Z\theta \in Sec(C)$, on déduit $K_s\theta \cap Sec(C) \neq \emptyset$ et donc $confstrong(P, b\theta)(E_s\theta)$ est vérifié.

Dans tous les cas, on a montré que $confstrong(P, b\theta)(E_s\theta)$ était vérifié, ce qui permet de conclure $search(P, b)(E_s) \Rightarrow confstrong(P, b)(E_s)$ quel que soit le nombre d'appels à la fonction *back* utilisés pour évaluer $search(P, b)(E_s)$ à vrai. □

7.1.5 Exemples d'utilisation et limites de l'algorithme

Nous avons vu aux paragraphes précédents que notre procédure permet de prouver la correction des protocoles de Needham-Schroeder-Lowe et d'Otway-Rees. De plus, nous présentons au paragraphe 7.2 une liste de protocole vérifiables par notre procédure.

Protocole informel

$$\begin{aligned} A_1 &\to A_2 : & < A_1, < A_2, \cdots < A_n, \{A_1, N_a\}_{\mathsf{pub}(A_2)} > \cdots > \\ A_2 &\to A_1 : & \{A_1, A_2, N_a\}_{\mathsf{pub}(A_1)} \end{aligned}$$

Protocole dans le modèle Millen-Rueß (on omet la description des états).

$$\emptyset \xrightarrow{\{N_a\}} \left\{ \ \{N_a\} \ddagger \{A_1, A_2\} \ \right\} \tag{7.14}$$

$$\{\{N_a\} \ddagger \{A_1, A_2\}\} \xrightarrow{\emptyset} \left\{ \begin{array}{l} < A_1, < A_2, \cdots < A_n, \\ \qquad \{A_1, N_a\}_{\mathsf{pub}(A_2)} > \cdots > \end{array} \right\} \tag{7.15}$$

$$\left\{ \begin{array}{l} \{N_b\} \ddagger \{A_1, A_2\}, \\ < A_1, < A_2, \cdots < A_n, \\ \qquad \{A_1, N_a\}_{\mathsf{pub}(A_2)} > \cdots > \end{array} \right\} \xrightarrow{\emptyset} \{\{A_1, A_2, N_a\}_{\mathsf{pub}(A_1)}\} \tag{7.16}$$

Branche $(N_a, \{\mathsf{prv}(A)\})$ pour la deuxième règle (7.16) du protocole :

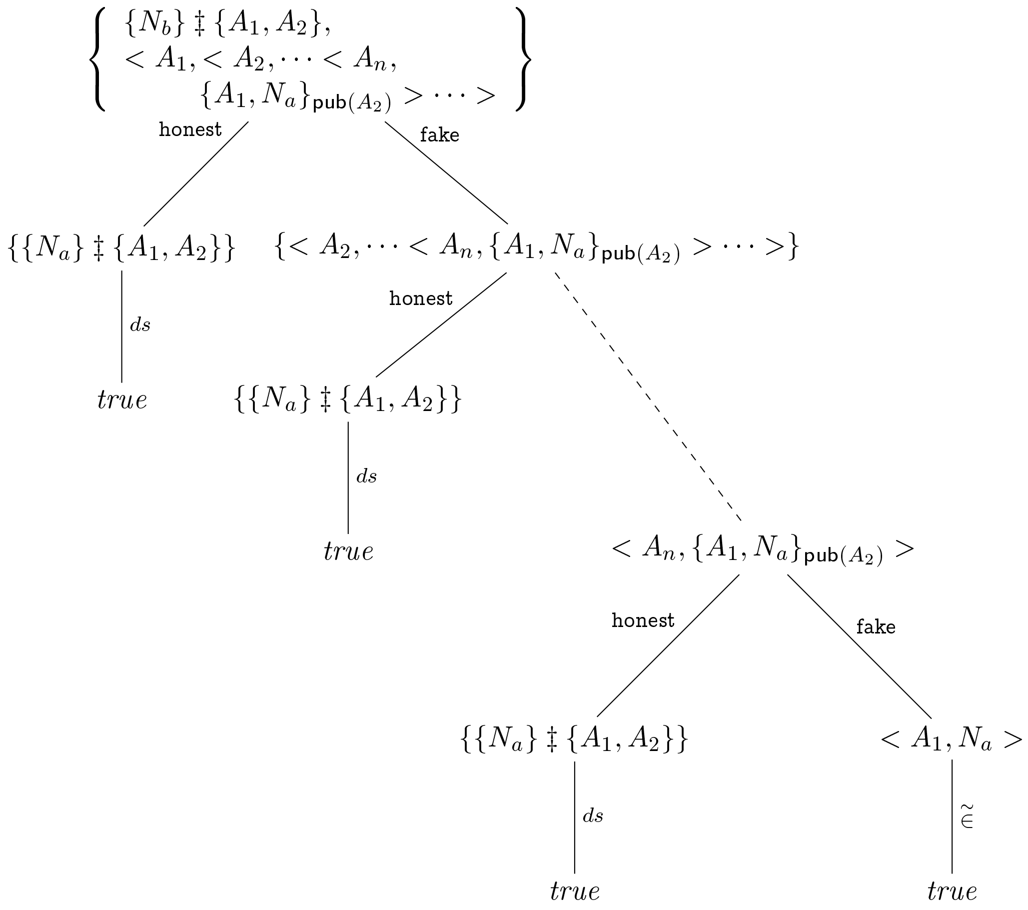


FIGURE 7.5 - *Protocole dont la preuve du secret demande n recherches en arrière.*

L'objet de cette section est de montrer quels sont les différents comportements possibles de notre algorithme et d'expliquer comment interpréter le résultat obtenu.

Une première limite de l'algorithme est la sur-approximation faite pour la notion de secret lorsque nous avons défini le caractère *impénétrable* d'un protocole. Ainsi, nous avons vu au paragraphe 4.1.3 du chapitre 4 que le protocole de Yahalom tel qu'il y est décrit n'est pas impénétrable et pourtant, le contre-exemple trouvé ne correspond pas à une véritable attaque du secret. Les autres limites sont plutôt dues au caractère incomplet de la procédure.

7.1.5.1 Nombre arbitraire de recherches en arrière

La preuve du secret du protocole d'Otway-Rees ne demande aucune recherche en arrière (*cf* paragraphe 7.1.2) alors que la preuve du secret du protocole de Needham-Schroeder-Lowe demande au maximum une recherche en arrière par branche de l'arbre de preuve (*cf* paragraphe 7.1.3). En fait, le nombre maximal de recherches en arrière peut être arbitrairement grand.

Ainsi, fixons un entier n supérieur ou égal à 2. On considère le protocole décrit à la figure 7.5, inspiré du protocole de Needham-Schroeder. La preuve du secret correspondant à la deuxième règle du protocole (règle 7.16) et à la branche $(N_a, \{\mathsf{prv}(A)\})$ est unique et demande exactement n recherches en arrière. Elle peut être visualisée, toujours à la figure 7.5.

7.1.5.2 Non terminaison

Comme annoncé dans l'introduction, la procédure peut ne pas terminer, que le protocole soit secret ou non. Cela peut en particulier survenir lorsque les règles du protocole contiennent des variables de type messages, c'est-à-dire des variables qui peuvent être instanciées par des messages arbitraires. Supposons par exemple qu'une règle du protocole contienne un message de la forme $\{X, A\}_K$ dans son *pre* et un message de la forme $< \{X\}_K, M_2 >$ dans son *post*. Supposons également que la procédure $back(P, b)(E_s)$ est appelée au moins une fois avec b et E_s tels qu'il existe $M \in E_s$ vérifiant $Z \in \mathsf{parts}(M)$ et $M = \{M'\}_K$. Alors la procédure *honest* appelle $back(P, b)(\{M', A\}_K)$, puis à nouveau $back(P, b)(\{M', A, A\}_K)$ et ainsi de suite sans jamais terminer.

Mais la procédure peut également ne pas terminer pour des protocoles dont aucune règle ne contient de variables de type message. Ainsi, on considère le protocole de Needham-Schroeder-Lowe modifié où l'agent A envoie à nouveau son premier message dans la dernière règle :

$$\begin{array}{lll} A & \to B : & \{A, N_a\}_{\mathsf{pub}(B)} \\ B & \to A : & \{N_a, N_b, B\}_{\mathsf{pub}(A)} \\ A & \to B : & \{N_b\}_{\mathsf{pub}(B)}, \{A, N_a\}_{\mathsf{pub}(B)}. \end{array}$$

Dans le modèle Millen-Rueß, la règle 7.13 est remplacée par :

$$\{\{N_a, N_b, B\}_{\mathsf{pub}(A)}, \mathsf{A}_{1,2}(A, B, N_a)\} \xrightarrow{\emptyset} \{< \{N_b\}_{\mathsf{pub}(B)}, \{A, N_a\}_{\mathsf{pub}(B)} >, \mathsf{A}_{1,3}(A, B, N_a, N_b)\}.$$

Le secret du protocole de Needham-Schroeder-Lowe assure le secret de ce protocole et pourtant la procédure ne termine pas pour la branche $(N_a, \{\mathsf{prv}(A)\})$ de la deuxième règle (voir figure 7.6).

On omet les états dans les ensembles d'événements pour une meilleure lisibilité

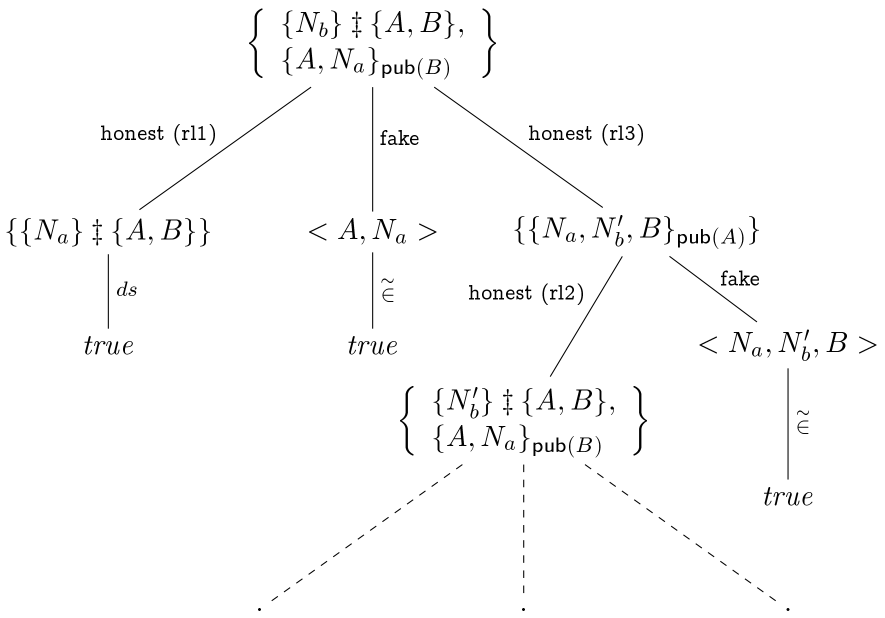


FIGURE 7.6 - *Non terminaison de la tentative de preuve.*

7.1.5.3 Échec de preuve

Lorsque la procédure termine avec la valeur $false$, cela ne signifie pas que le protocole n'est pas secret mais seulement que la procédure de preuve du secret a échoué. Cependant, l'arbre d'échec de la preuve fournit des informations qui peuvent permettre de trouver une véritable attaque sur le protocole.

Attaque. Ainsi, considérons le protocole de Needham-Schroeder, décrit figure 7.7. La procédure de preuve échoue, bien sûr, dans la preuve du protocole. Plus précisément, la fonction $main(NS, b)(rl)$ est évaluée à vrai pour toutes les branches des trois premières règles du protocole et $main(NS, b)(rl)$ est évalué à faux pour la branche $b = (N_b, \{\mathsf{prv}(B)\})$ de la dernière règle du protocole. L'arbre d'échec de la preuve est présenté à la figure 7.8.

On peut alors reconstruire l'attaque à partir de la feuille d'échec étiquetée par $false$. La partie la plus difficile de la reconstruction consiste à produire un historique accessible qui contienne les états et les spécifications mentionnés dans la branche conduisant à la feuille d'échec. Aussi, on joue les deux règles d'initialisation suivantes :

$$\emptyset \xrightarrow{\{N'_a, N_b\}} \left\{ \begin{array}{l} \{N'_a\} \ddagger \{A, B'\}, \{N_b\} \ddagger \{A, B'\}, \\ \mathsf{A}_{1,1}(A, B'), \mathsf{B'}_{2,1}(B') \end{array} \right\}$$
$$\emptyset \xrightarrow{\{N_a, N'_b\}} \left\{ \begin{array}{l} \{N_a\} \ddagger \{A, B\}, \{N'_b\} \ddagger \{A, B\}, \\ \mathsf{A}_{1,1}(A, B), \mathsf{B}_{2,1}(B) \end{array} \right\}.$$

Puis la deuxième règle du protocole correspondant à la session entre A et B :

$$\{\{N_a\} \ddagger \{A, B\}, \mathsf{A}_{1,1}(A, B)\} \xrightarrow{\emptyset} \{\{A, N_a\}_{\mathsf{pub}(B)}, \mathsf{A}_{1,2}(A, B, N_a)\}.$$

Protocole informel

$$\begin{array}{lll} A & \rightarrow B: & \{A, N_a\}_{\mathsf{pub}(B)} \\ B & \rightarrow A: & \{N_a, N_b\}_{\mathsf{pub}(A)} \\ A & \rightarrow B: & \{N_b\}_{\mathsf{pub}(B)} \end{array}$$

Protocole dans le modèle Millen-Rueß

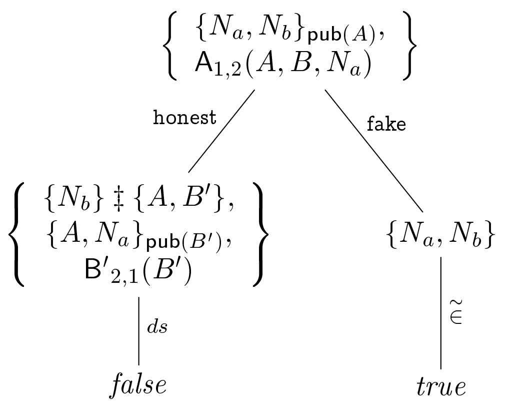
$$\emptyset \xrightarrow{\{N_a, N_b\}} \left\{ \begin{array}{l} \{N_a\} \ddagger \{A, B\}, \{N_b\} \ddagger \{A, B\}, \\ \mathsf{A}_{1,1}(A, B), \mathsf{B}_{2,1}(B) \end{array} \right\} \tag{7.17}$$

$$\{\{N_a\} \ddagger \{A, B\}, \mathsf{A}_{1,1}(A, B)\} \xrightarrow{\emptyset} \{\{A, N_a\}_{\mathsf{pub}(B)}, \mathsf{A}_{1,2}(A, B, N_a)\} \tag{7.18}$$

$$\left\{ \begin{array}{l} \{N_b\} \ddagger \{A, B\}, \\ \mathsf{B}_{2,1}(B), \{A, N_a\}_{\mathsf{pub}(B)} \end{array} \right\} \xrightarrow{\emptyset} \{\{N_a, N_b\}_{\mathsf{pub}(A)}, \mathsf{B}_{2,2}(B, A, N_b, N_a)\} \tag{7.19}$$

$$\{\{N_a, N_b\}_{\mathsf{pub}(A)}, \mathsf{A}_{1,2}(A, B, N_a)\} \xrightarrow{\emptyset} \{\{N_b\}_{\mathsf{pub}(B)}, \mathsf{A}_{1,3}(A, B, N_a, N_b)\} \tag{7.20}$$

FIGURE 7.7 - *Protocole de Needham-Schroeder .*



branche $(N_b, \{\mathsf{prv}(A)\})$

FIGURE 7.8 - *Échec de preuve du secret du protocole de Needham-Schroeder.*

On obtient alors un historique accessible :

$$H_1 = \{\{N'_a\} \ddagger \{A, B'\}, \{N_b\} \ddagger \{A, B'\}, \{N'_b\} \ddagger \{A, B\}, \{N_a\} \ddagger \{A, B\}, \\ \mathsf{A}_{1,1}(A, B'), \mathsf{B'}_{2,1}(B'), \mathsf{B}_{2,1}(B), \{A, N_a\}_{\mathsf{pub}(B)}, \mathsf{A}_{1,2}(A, B, N_a)\}.$$

Pour obtenir l'ensemble des événements qui étiquette le noeud prédécesseur de la feuille $false$, il suffit alors de jouer une règle malhonnête : si $\mathsf{prv}(B), \mathsf{pub}(B') \in I$ alors $\{A, N_a\}_{\mathsf{pub}(B')} \in \mathsf{fake}(Msg(H_1) \cup I)$ et donc l'historique $H_2 = H_1 \cup \{\{A, N_a\}_{\mathsf{pub}(B')}\}$ est accessible pour une connaissance initiale I telle que $\mathsf{prv}(B), \mathsf{pub}(B') \in I$.

Il suffit maintenant de remonter la branche de l'arbre : on atteint la racine en jouant la troisième règle du protocole :

$$\left\{ \begin{array}{l} \{N_b\} \ddagger \{A, B'\}, \\ \mathsf{B'}_{2,1}(B'), \{A, N_a\}_{\mathsf{pub}(B')} \end{array} \right\} \xrightarrow{\emptyset} \{\{N_a, N_b\}_{\mathsf{pub}(A)}, \mathsf{B'}_{2,2}(B', A, N_b, N_a)\}.$$

Puis on joue la règle sur laquelle a échoué la procédure :

$$\{\{N_a, N_b\}_{\mathsf{pub}(A)}, \mathsf{A}_{1,2}(A, B, N_a)\} \xrightarrow{\emptyset} \{\{N_b\}_{\mathsf{pub}(B)}, \mathsf{A}_{1,3}(A, B, N_a, N_b)\}.$$

Or $\{N_b\}_{\mathsf{pub}(B)} \notin \mathit{Coideal}(\{N_b\} \ddagger \{A, B'\})$ et pourtant $\{N_b\} \ddagger \{A, B'\}$ est une spécification compatible avec I si on choisit $I = \{\mathsf{prv}(B), \mathsf{pub}(B')\}$ par exemple. Cela correspond à une attaque sur le nonce engendré par B' pour A, où A et B' sont des agents honnêtes et B un agent malhonnête (l'intrus connaît sa clef privée). Cette attaque peut être résumée de la manière suivante :

$$\begin{array}{l} A \xrightarrow{\{A, N_a\}_{\mathsf{pub}(B')}} B' \\ B'(A) \xrightarrow{\{A, N_a\}_{\mathsf{pub}(B)}} B \\ A \xleftarrow{\{N_a, N_b\}_{\mathsf{pub}(A)}} B \\ A \xrightarrow{\{N_b\}_{\mathsf{pub}(B')}} B' \end{array}$$

Fausses attaques. La procédure peut également échouer sans qu'on puisse reconstruire une attaque. Une première raison est qu'un protocole peut être préserver le secret sans être « impénétrable ». Ainsi, notre procédure échoue à montrer le secret du protocole de Yahalom (décrit aux pages 58 et 59) car celui-ci n'est pas impénétrable : le dernier message envoyé $\{N_b\}_{K_{ab}}$ n'est pas dans le coidéal de $\{N_b\} \ddagger \{A, B\}$.

Une deuxième raison est le manque de tests pertinents pour les données dont on ne demande pas le secret. Ainsi considérons le protocole suivant où l'on demande le secret du nonce N_2 :

$$\begin{array}{lll} A & \to B : & \{N_1, N_2\}_{\mathsf{pub}(B)} \\ B & \to A : & \{N_1\}_{N_2}. \end{array}$$

Le secret du nonce est clairement préservé et pourtant notre procédure échoue (voir figure 7.9). Une perspective pour l'amélioration de notre outil serait de permettre de conclure à vrai lorsque le nonce de la branche a été engendré sans spécification de secret.

7.2 Implémentation

L'algorithme ainsi obtenu a été implémenté en *Ocaml* puis adapté au langage *eva* dans le cadre du projet RNTL EVA.

Protocole dans le modèle Millen-Rueß (nous ne précisons pas les changements d'états pour alléger les notations).

$$\begin{aligned}
\emptyset &\xrightarrow{\{N_2\}} \{\{N_2\} \ddagger \{A,B\}\} \\
\{\{N_2\} \ddagger \{A,B\}\} &\xrightarrow{\{N_1\}} \{\{N_1,N_2\}_{\mathsf{pub}(B)}\} \\
\{\{N_1,N_2\}_{\mathsf{pub}(B)}\} &\xrightarrow{\emptyset} \{\{N_1\}_{N_2}\}
\end{aligned}$$

On considère l'unique branche $(N_1, \{N_2\})$ du message $\{N_1\}_{N_2}$ émis à la dernière règle.

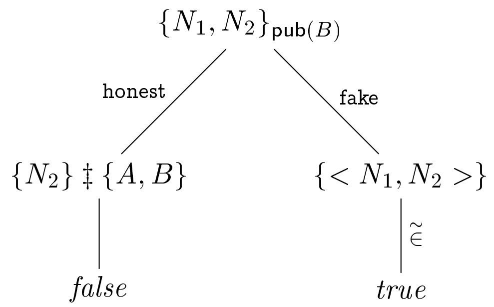


FIGURE 7.9 - *Exemple de fausse attaque.*

7.2.1 Structure de l'outil

La structure de l'outil est résumée à la figure 7.10.

L'outil admet deux types d'entrées :

- un fichier `.ml` contenant un protocole décrit dans le modèle Millen-Rueß,
- ou un fichier `.eva` contenant la spécification d'un protocole dans le langage *eva*.

Le langage *eva* est l'une des contributions du projet RNTL EVA. Ce dernier a consisté, dans un premier temps, à élaborer un langage commun de spécification des protocoles cryptographiques avec deux versions du langage :

- le langage concret destiné aux utilisateurs, où la description des protocoles est proche de celle utilisée dans [CJ97] ;
- le langage abstrait, destiné aux outils, où les transitions sont explicitées pour chaque participant du protocole.

De plus, il existe un traducteur automatique *evatrans* du langage concret (fichiers `.eva`) vers le langage abstrait (fichiers `.cpl`). Dans un deuxième temps, les participants au projet EVA ont développé trois logiciels de preuve automatique pour les protocoles, l'outil présenté ici est l'un d'entre eux.

Pour traiter un protocole décrit en *eva*, notre outil commence par utiliser le traducteur *evatrans* pour obtenir la spécification du protocole en *cpl*. Puis cette spécification est traduite en un protocole du modèle Millen-Rueß.

À partir d'un protocole du modèle Millen-Rueß, on applique l'algorithme décrit à la section précédente et on obtient deux types de sortie :

- une réponse oui ou non suivant que la procédure de preuve a réussi ou échoué,
- un affichage de l'ensemble des arbres de preuve.

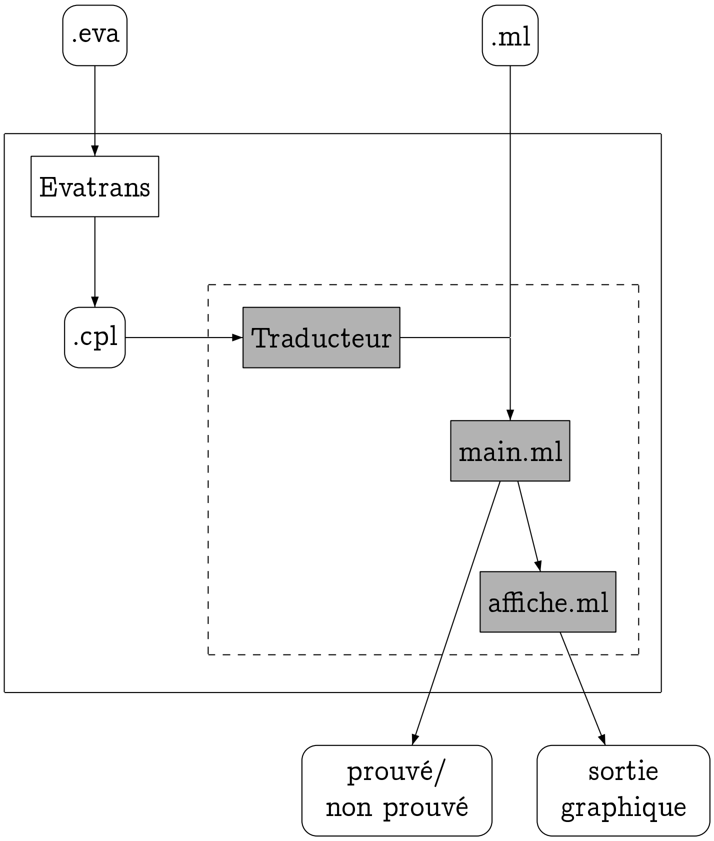


Figure 7.10 - *Structure de l'outil.*

Comme expliqué au paragraphe 7.1.5.3, ce dernier point est en particulier utile en cas d'échec de la preuve pour tenter de reconstituer une véritable attaque sur le protocole considéré.

Exemple 7.5 *Nous reprenons l'exemple du protocole de Needham-Schroeder-Lowe dont la preuve a été entièrement développée au paragraphe 7.1.3. La spécification de ce protocole en* eva *ainsi que sa spécification en* Ocaml *telles qu'elles peuvent être données à l'outil, sont présentées à la figure 7.11. La traduction est* cpl *peut être trouvée dans l'annexe B.*

7.2.2 Résultats

Notre outil peut être expérimenté en ligne à l'adresse suivante :
`http ://www-eva.imag.fr/fournitures1.html`

```
Needham_Schroeder-Lowe-_cles_publiques

alg : asym_algo
everybody knows alg

A, B: principal

basetype key
Na, Nb : key

keypair^alg PK, SK (principal)
everybody knows PK

A knows A, B, SK(A)
B knows B, SK(B)

{
  1. A -> B : {Na, A}_(PK(B))^alg
  2. B -> A : {Na, Nb, B}_(PK(A))^alg
  3. A -> B : {Nb}_(PK(B))^alg
}

s1. session *{A,B} A=A, B=B

assume secret (SK(A)@s1.A), secret (SK(B)@s1.B),
       secret (SK(B@s1.A)), secret (SK(A@s1.B))

claim *A*G secret (SK(A)@s1.A),
      *A*G secret (SK(B)@s1.B),
      *A*G secret (Na@s1.A),
      *A*G secret (Nb@s1.B)
```

Fichier `nsl.eva`

```
(* Needham_Schroeder_cles_publiques *)

let rl0 = ( [],
    [N 1; N 2],
    [Cast ([Bn (N 1)],[A 1; A 2]);
         Cast ([Bn (N 2)],[A 1; A 2])]
  ) ;;

let rl1 = ( [Cast ([Bn (N 1)],[A 1; A 2])],
    [],
    [M (Encra (Key (Pub (A 2)),
       Conc [Nonce (N 1); Agent (A 1)]))]
  ) ;;

let rl2 = ( [Cast ([Bn (N 2)],[A 1; A 2]);
       M (Encra (Key (Pub (A 2)),
 Conc [Nonce (N 1); Agent (A 1)]))],
    [],
    [M (Encra (Key (Pub (A 1)),
       Conc [Nonce (N 1); Nonce (N 2); Agent (A 2)]))]
  ) ;;

let rl3 = ( [M (Encra (Key (Pub (A 1)),
       Conc [Nonce (N 1); Nonce (N 2); Agent (A 2)]))],
    [],
    [M (Encra (Key (Pub (A 2)), Nonce (N 2)))]
  ) ;;

let nsl = [rl0; rl1; rl2; rl3];;
```

Fichier `nsl.ml`

FIGURE 7.11 - *Fichiers .eva et .ml correspondants au protocole de Needham-Schroeder-Lowe.*

7.2.2.1 Protocoles testés

Les résultats sont récapitulés dans le tableau ci-dessous. Ils ont été obtenus avec un Pentium III, 933 Mhz, 256 Mo de RAM.

Protocole	Preuve	# maximal de "back"	Temps ms
Otway-Rees	Oui	0	0,35
Woo and Lam	Oui	0	0,11
Denning-Sacco	Non	0	0,07
ISO Symmetric Key	Oui	0	0,15
Needham-Schroeder-Lowe	Oui	1	1,08
Needham-Schroeder	Non	1	1,23
Wide-Mouthed-Frog (modifié)	Oui	2	4,76
Kao-Chow	Oui	3	8,94

Les spécifications en *eva* des protocoles cités dans ce tableau sont données dans l'annexe B. Comme on le voit, tous les calculs sont très rapides puisque les temps de réponse sont donnés en millisecondes.

Il est à noter que les échecs de preuve mentionnés dans ce tableau permettent, dans les deux cas, de retrouver une attaque réelle sur le protocole. Il existe cependant des cas où la procédure échoue sans donner d'indications sur une réelle faille du protocole testé.

7.2.2.2 Sortie graphique

La sortie graphique permet de visualiser les arbres de preuve (ou d'échec de preuve) de chacune des branches des règles du protocole étudié. Ces arbres de preuves se présentent de la même manière que ceux dessinés aux paragraphes 7.1.3 et 7.1.5. Ainsi, les quatre arbres de preuve obtenus pour le protocole de Needham-Schroeder-Lowe se trouvent dans l'annexe B. Ils sont similaires à ceux dessinés à la figure 7.4.

7.2.3 Comparaisons

L'objet de ce paragraphe est de comparer notre outil avec les autres logiciels existants. Comme rappelé dans l'introduction de ce chapitre, certains de ces logiciels sont uniquement consacrés à la recherche d'attaques sur les protocoles. Ils ne sont donc pas directement comparables à notre outil mais nous allons tout de même en profiter pour les présenter en quelques mots. Toutefois l'ensemble des outils présentés ici est loin d'être une liste exhaustive des logiciels existants.

7.2.3.1 Outils consacrés à la recherche d'attaques

Casper/FDR Cet outil a été développé par G. Lowe [Low97a, RSG$^+$00] pour vérifier les protocoles décrits en CSP [Sch96a]. Le nombre de sessions et de participants est borné, la taille des messages est également bornée. De plus, les messages sont typés et il n'y a pas de clefs composées. La vérification du protocole correspond alors au model-checking d'un modèle fini. La principale difficulté est le traitement de l'explosion du nombre d'états lorsqu'on considère plusieurs sessions par exemple. C'est en particulier le model-checking avec FDR du protocole de Needham-Schroeder qui a permis à G. Lowe de trouver son attaque sur ce protocole [Low96] puis d'en proposer une nouvelle version.

MURϕ L'outil MURϕ [MMS97] est assez proche de Casper. Il traite lui aussi des protocoles avec un nombre de sessions et une taille des messages bornée. De même, les messages sont typés. Le modèle utilisé suppose de plus que l'intrus a une mémoire finie. Les résultats obtenus permettent de retrouver quelques unes des attaques classiques mais l'outil ne parvient pas à traiter plus de deux ou trois sessions en parallèle. Cependant, cet outil n'est pas uniquement dédié à la vérification des protocoles cryptographique. Il peut également vérifier la « cohérence de cache » de protocoles (pas nécessairement cryptographiques) [SD95].

Athena Comme les deux outils précédents, Athena [Son99] se place dans un cadre où le nombre de sessions et de participants est borné. Par contre, il ne suppose pas que la taille des messages est bornée *a priori*. Les protocoles sont exprimés dans le modèle des *strand spaces* [THG99] et l'outil suppose à nouveau que les messages sont typés et qu'il n'y a pas de clefs composées. Le tableau comparatif des résultats entre Athena et MURϕ fait apparaître Athena comme plus rapide mais peu d'exemples d'attaques sont donnés.

Casrul L'outil Casrul [JRV00] permet également de chercher des attaques pour un nombre de sessions et de participants borné. C'est l'un des seuls outils qui permet de traiter les messages dans toute leur généralité : les clefs composées sont supportées et les messages n'ont pas besoin d'être typés. Les protocoles sont décrits sous forme de règles de réécriture. CASRUL retrouve ainsi de nombreuses attaques mentionnées dans le [CJ97].

Aviss Le logiciel Aviss [ABB+02] est un logiciel assez récent qui traduit les protocoles en un langage commun à trois logiciels de vérification. L'utilisation commune de ces trois logiciels permet de retrouver de très nombreuses attaques déjà connues (la majorité de celles mentionnées dans [CJ97]) et a également permis de découvrir une attaque nouvelle sur le protocole de Denning-Sacco.

7.2.3.2 Outils consacrés à la preuve

Les deux outils les plus proches de Securify sont l'outil développé par B. Blanchet et l'outil Hermes, développé par Verimag à Grenoble.

Blanchet B. Blanchet [Bla01] exprime les protocoles sous forme de règles Prolog. Il vérifie le secret pour un nombre non borné de sessions. La taille des messages n'est pas bornée non plus. Les nonces sont abstraits par une fonction des paramètres reçus dans les messages précédents ce qui peut conduire à de fausses attaques. Il se peut, en effet, qu'un protocole ne soit pas prouvé correct car sujet à des attaques qui utilisent l'abstraction faites sur les nonces alors qu'une telle attaque n'est pas reproductible dans la réalité. Comme notre outil, celui-ci type au moins en partie les messages et ne permet pas d'exprimer les clefs composées. Les résultats obtenus sont assez similaires aux nôtres, à ceci près que l'outil de B. Blanchet demande un peu plus de temps pour conclure au secret.

Hermes Comme l'outil précédent, Hermes [BLP03] permet de prouver le secret de manière complètement automatique, pour un nombre non borné de sessions et de participants. Les clefs doivent également être atomiques et toutes les variables utilisées pour chiffrer doivent être de type clef. Pour vérifier le secret, cet outil commence par abstraire les nonces par un nombre fini de constantes en distinguant une session particulière de toutes les autres, les nonces des autres sessions étant confondus. De plus, il se ramène par abstraction à un nombre fini de participants. Un calcul symbolique des messages accessibles est ensuite effectué, en utilisant éventuellement du « widening ».

Les deux outils suivants sont également dédiés à la vérification des protocoles pour un nombre non borné de sessions et de participants mais ils ne sont pas complètement automatiques. En effet, ils nécessitent l'intervention de l'utilisateur à certaines étapes.

TAPS E. Cohen a développé un outil, TAPS [Coh00], qui vérifie des propriétés de sécurité en réalisant des preuves d'invariants. Les protocoles et les propriétés sont décrits en logique du premier ordre. Cet outil peut donc vérifier d'autres propriétés que le secret. L'invariant est souvent généré et prouvé automatiquement, mais c'est parfois à l'utilisateur de renforcer la propriété à démontrer pour aider l'outil à générer le bon invariant. Sous ces conditions, l'outil TAPS a permis de vérifier la plupart des protocoles de [CJ97], en modifiant éventuellement la propriété à vérifier pour les protocoles qui ont une faille.

Paulson L. Paulson [Pau97a] propose un modèle de trace pour les protocoles, modèle qui a en partie inspiré le modèle Millen-Rueß. La taille des messages n'est pas bornée, mais comme dans le modèle Millen-Rueß, les messages peuvent être typés, ce qui facilite la preuve des protocoles. Plus qu'une procédure automatique de preuve, il s'agit ici d'une méthode de preuve, partiellement automatisée en Isabelle/HOL. L'utilisateur doit lui-même faire la preuve du protocole, mais de nombreuses tactiques ont été programmées pour faciliter cette tâche.

Troisième partie

Classes décidables

CHAPITRE 8

Classes décidables de protocoles cryptographiques

Dans le cadre d'un nombre borné de sessions, M. Rusinowitch et M. Turuani ont montré que le secret était NP-complet pour une classe de protocoles très générale comprenant les clefs composées et le chiffrement asymétrique. Cependant, la vérification pratique des protocoles cryptographiques par des outils comme Casper [Low97a] ou Casrul [JRV00] se limite à trois ou quatre sessions, la complexité des algorithmes étant trop grande pour permettre la vérification d'un nombre plus important de sessions.

Il serait donc intéressant d'obtenir des algorithmes de décision dans le cadre d'un nombre non borné de sessions. Malheureusement, de nombreuses classes de protocoles sont indécidables : rappelons quelques-uns des résultats d'indécidabilité.

1. Nous avons montré au chapitre 3, que le secret des protocoles sans nonces permettant au moins deux copies arbitraires par règles est indécidables.
2. J. Mitchell *et al.* [DLMS99] ont montré que le secret des protocoles avec nonces, mais tels que la profondeur des messages est bornée, est indécidable.
3. R. Amadio et W. Charatonik [AC02] ont montré que le secret des protocoles avec nonces mais n'utilisant que la primitive de chiffrement (et non la paire) et tels que la profondeur des messages est bornée, est toujours indécidable.

Le résultat 1 motive notre première restriction : nous considérons des protocoles qui ne permettent qu'au plus une copie d'un message arbitraire par transition. Les résultats 2 et 3 motivent notre seconde restriction : nous ne considérons que des protocoles sans nonce (ou avec un nombre fini de nonces). Nous montrons alors que le secret et l'authentification sont décidables pour de tels protocoles, puis nous étendons ce résultat aux protocoles avec « ou » exclusif. À notre connaissance, il s'agit du premier résultat de décidabilité pour un nombre non borné de sessions et en présence de propriétés algébriques.

Pour cela, nous introduisons un nouveau fragment décidable de la logique du premier ordre (partie 8.2) puis nous généralisons ce résultat à un fragment plus large permettant la modélisation du « ou » exclusif (partie 8.3). Enfin, nous appliquons ces résultats aux protocoles cryptographiques (partie 8.4) et nous comparons les résultats obtenus aux autres travaux (partie 8.5).

8.1 Cadre

Le but de cette partie est d'une part de justifier les hypothèses introduites et d'autre part, de rappeler la terminologie et quelques propriétés classiques de la résolution en logique du premier ordre sous forme clausale.

8.1.1 Discussion des hypothèses

Nous allons tout d'abord justifier un peu plus la classe de protocoles considérée.

8.1.1.1 Une seule copie

Comme annoncé dans l'introduction, nous considérons des protocoles cryptographiques ne permettant qu'au plus une copie de message arbitraire par transition. Cette hypothèse nous semble pertinente, d'une part parce qu'elle est nécessaire au regard du résultat 1 énoncé dans l'introduction de ce chapitre et d'autre part parce qu'elle est satisfaite par un grand nombre de protocoles de [CJ97].

Exemple 8.1 *Considérons une nouvelle fois le protocole de Needham-Schroeder-Lowe à clefs publiques :*

$$\begin{array}{lll} A & \rightarrow B: & \{A, N_a\}_{\mathsf{pub}(B)} \\ B & \rightarrow A: & \{B, N_a, N_b\}_{\mathsf{pub}(A)} \\ A & \rightarrow B: & \{N_b\}_{\mathsf{pub}(B)} \end{array}$$

La première règle du protocole n'induit aucun test ni copie : l'agent A crée et envoie un nouveau nonce N_a. Dans la deuxième règle, B copie le message inconnu correspondant à N_a et toutes les autres composantes du message reçu et du message envoyé sont connues de B. Enfin, dans la troisième règle, l'agent A reconnaît son propre nonce N_a (ce n'est donc pas un test arbitraire) et copie le message correspondant au nonce N_b. On constate ainsi pour ce protocole qu'au plus un message arbitraire est copié et/ou testé.

8.1.1.2 Nombre fini de nonces

Notre deuxième hypothèse est nettement plus restrictive : ne considérer qu'un nombre fini de nonces peut amener des « fausses » attaques.

Exemple 8.2 *Considérons le protocole suivant :*

$$\begin{array}{lll} A & \rightarrow B: & \{N, N'\}_{\mathsf{pub}(B)} \\ B & \rightarrow A: & \{N_1, N\}_{\mathsf{pub}(B)}, \{\mathsf{Secret}\}_{\{N_1, N_1\}_{\mathsf{pub}(B)}} \end{array}$$

L'agent A engendre deux nonces N et N' et les envoie chiffrés avec la clef publique de B. L'agent B recopie le nonce N et engendre un nouveau nonce N_1. Ce protocole est sûr si on suppose que B engendre bien des nonces différents à chaque fois mais ne l'est plus sinon.

Cependant, l'abstraction qui consiste à remplacer les nonces par un nombre fini de constantes est une abstraction correcte : si un protocole est secret dans le modèle abstrait, alors il est également secret dans le modèle concret.

8.1.2 Résolution

Nous introduisons ici les règles de résolution et les stratégies ordonnées. Toutes les définitions, notations et propriétés sont tirées de [FLHT01] et [GLM97].

Définition 8.1 (résolution) *Soient C et C' deux clauses, L et L' deux atomes. Une* règle de résolution binaire *est une règle de la forme :*

$$\frac{C \vee L \quad C' \vee \neg L'}{C\sigma \vee C'\sigma} \; \sigma = mgu(L, L').$$

La clause $C\sigma \vee C'\sigma$ est appelée résolvant *et l'atome $L\sigma$ est appelé* atome résolu.

La règle de factorisation *est la règle :*

$$\frac{C \vee L \vee L'}{C\sigma \vee L\sigma} \; \sigma = mgu(L, L').$$

La clause $C\sigma \vee L\sigma$ est également appelée résolvant *et l'atome $L\sigma$ est appelé* atome résolu.

L'ensemble des résolvants d'un ensemble de clause S est noté $Res(S)$ et on définit $R(S) = S \cup Res(S)$.

La résolution est correcte et complète pour la réfutation.

Une autre règle d'inférence est la règle de *séparation* [Wei01] qui permet de « séparer » deux parties d'une même clause contenant des ensembles de variables disjoints.

Définition 8.2 *Soient L et L' deux littéraux d'une clause C. On définit $L \sim_v L'$ si $V(L) \cap V(L') \neq \emptyset$ et $\sim_{v^*}$ la clôture transitive de $\sim_v$. Une classe d'équivalence pour $\sim_{v^*}$ est appelée* composante *de C. Une clause est dite* décomposée *si elle n'a qu'une composante.*

Définition 8.3 (séparation) *Soit S un ensemble de clauses. L'ensemble $SPLIT(S)$ est l'ensemble d'ensembles de clauses, obtenu en* séparant *au maximum les clauses de S. On définit récursivement :*

1. *$\Sigma_0 = \{S\}$;*
2. *Si pour tout ensemble S' de Σ_i et pour toute clause C de S', la clause C est décomposée, alors : $\Sigma_{i+1} = \Sigma_i$.*
3. *S'il existe un ensemble S' de Σ_i et une clause C de S' telle que C se décompose en C' et C'', alors :*

$$\Sigma_{i+1} = (\Sigma_i - \{S\}) \cup \{(S' - \{C\}) \cup \{C'\}\} \cup \{(S' - \{C\}) \cup \{C''\}\}.$$

On définit alors : $SPLIT(S) = \Sigma_k$ où k est l'entier minimal tel que $\Sigma_k = \Sigma_{k+1}$.

Proposition 8.1 *Soit S un ensemble de clauses. S est insatisfaisable si et seulement si tous les ensembles de clauses de $SPLIT(S)$ sont insatisfaisables.*

La résolution peut-être raffinée en ne résolvant que sur des littéraux maximaux pour un ordre fixé : il s'agit de la résolution ordonnée.

Définition 8.4 (résolution ordonnée) *Soit S un ensemble de clauses et $\leqslant$ une relation d'ordre. On note $<$ la relation d'ordre strict associée : $L_1 < L_2$ si $L_1 \leqslant L_2$ et $L_1 \neq L_2$. La clause E est obtenue par* résolution ordonnée *sur $\leqslant$ à partir de S, noté $E \in R_{\leqslant}(S)$, si E est un résolvant de S tel que l'atome résolu L est maximal dans E pour l'ordre $\leqslant$: il n'existe pas d'atome L' dans E tel que $L < L'$.*

Remarque : Il s'agit ici d'un critère *a posteriori* car on teste la maximalité de l'atome résolu par rapport au résolvant. On peut également définir une stratégie de résolution ordonnée basée sur un critère *a priori* : on effectue la résolution entre $C \vee L$ et $C' \vee L'$ seulement si C est maximal dans C et si L' est maximal dans C'.

Une condition suffisante pour la complétude de la résolution ordonnée est d'utiliser un ordre *relevable* [FLHT01, KH69], aussi appelé *stable* dans [GLM97].

Définition 8.5 (ordre relevable) *Un ordre $\leqslant_{\mathcal{R}}$ est* relevable *si pour tous littéraux A, B et pour toute substitution θ, la relation $A \leqslant_{\mathcal{R}} B$ implique $A\theta \leqslant_{\mathcal{R}} B\theta$.*

On obtient alors les propriétés suivantes.

Proposition 8.2 *Soit $\leqslant$ un ordre relevable.*

La stratégie ordonnée par $\leqslant$ est correcte et complète pour la réfutation. Autrement dit, soit S un ensemble de clauses et S^* défini par $S^* = \cup_i S_i$ avec $S_0 = S$ et $S_{i+1} = R(S_i)$. Alors S est insatisfaisable si et seulement si la clause vide $\perp$ appartient à S^*.

La stratégie ordonnée par $\leqslant$, combinée à la règle de séparation est correcte et complète pour la réfutation. Autrement dit, soit S un ensemble de clauses, on définit la suite $\mathcal{S}_i$ par $\mathcal{S}_0 = \{S\}$, $\mathcal{S}_{2i+1} = \{S' \in SPLIT(S) \mid S \in \mathcal{S}_{2i}\}$ et $\mathcal{S}_{2i+2} = \{R_{\leqslant}(S) \mid S \in \mathcal{S}_{2i}\}$. Alors S est insatisfaisable si et seulement si il existe un entier i tel que pour tout ensemble de clauses S de $\mathcal{S}_i$, la clause vide $\perp$ appartient à S.

Nous utiliserons également dans ce chapitre quelques notions relatives à la réécriture. Nous nous reportons ainsi à [DJ90, Com88] pour les définitions de *règle de réécriture*, *système convergent*, *forme normale* et *redex*.

8.2 Un premier fragment décidable de la logique du premier ordre

Nous proposons dans cette partie une classe décidable de clauses qui est une variante de la classe $\mathcal{S}^+$ de [FLHT01] étendue aux littéraux contenant des termes fonctionnels clos.

8.2.1 Définitions

Nous fixons $\mathcal{F}$ un ensemble fini de symboles fonctionnels, $\mathcal{V}$ un ensemble de variables, et $\mathcal{P}$ un ensemble fini de symboles de prédicats. Nous reprenons les définitions et notations introduites au chapitre 3.

Notation : Pour toute clause C, l'ensemble des variables de C est noté $V(C)$ et le cardinal de l'ensemble $V(C)$ est noté $Card(V(C))$. Les notations sont identiques pour les littéraux et les termes. Soit P un littéral positif, on note $L = \pm P$ pour $L \in \{P, \neg P\}$. Si u et t sont des termes de $\mathcal{T}(\mathcal{F} \cup \mathcal{V})$ et x une variable de u, on note $u[t/x]$ le terme u dans lequel chaque occurrence de x a été remplacée par t.

Nous considérons une classe $\mathcal{C}$ de clauses qui étend à la fois la classe des clauses à une variable et la classe des clauses représentant les transitions des automates d'arbres.

Définition 8.6 *Un ensemble de clauses S appartient à la classe $\mathcal{C}$ si pour toute clause C de S, soit C ne contient qu'au plus une variable, soit pour chaque littéral L de C :*

1. *ou bien $L = \pm P(x_i)$ pour un certain prédicat P ;*
2. *ou bien $L = \pm P(u[f(x_1, \ldots, x_n)/y])$ pour un certain prédicat P et un certain symbole de fonction f, tels que $\{x_1, \ldots, x_n\} = V(C)$ et u est un terme de $\mathcal{T}(\mathcal{F} \cup \{y\})$.*

Nous définissons également un sous-ensemble strict $\mathcal{C}'$ de $\mathcal{C}$ tel que pour chaque clause C d'un ensemble S de $\mathcal{C}'$, si C contient deux littéraux de la forme $\pm P(x)$ avec des variables distinctes, alors C contient également un littéral de la forme $\pm Q(u[f(x_1, \ldots, x_n)])$ contenant toutes les variables de la clause C.

Exemple 8.3 *Soient a, s, f, g, h des symboles de fonction et x une variable. L'ensemble S formé des trois clauses ci-dessous appartient à la classe $\mathcal{C}$.*

$$\neg P_1(x_1) \vee \neg P_1(x_2) \vee P_2(s(f(x_1, x_2))) \vee P_1(f(g(x_2, x_1), g(x_2, x_1))))$$
$$\neg P(f(s(x), f(x, a))) \vee P(f(x, s(x)))$$
$$P(h(x_1, x_2, x_3)) \vee P(x_2)$$

Par contre, tout ensemble contenant la clause $P(f(x_1, f(x_1, x_2)))$ n'appartient pas à la classe $\mathcal{C}$.

Par abus de langage, on pourra écrire qu'une clause C est dans $\mathcal{C}$ au lieu d'écrire que l'ensemble $\{C\}$ est dans $\mathcal{C}$, pour exprimer que C est une clause de la forme définie ci-dessus. On pourra également écrire $P(u[f(x_1, \ldots, x_n)])$ au lieu de $P(u[f(x_1, \ldots, x_n)/y])$ lorsqu'il n'y a pas d'ambiguïté.

Nous montrons au paragraphe 8.2.2 que la satisfaisabilité d'un ensemble de clauses de $\mathcal{C}$ est décidable. Nous verrons dans la partie 8.4 comment ces clauses permettent de décrire les protocoles cryptographiques.

8.2.1.1 Comparaison avec la classe de Skolem étendue

Rappelons la classe $\mathcal{S}^+$ définie par C.G. Fermüller, A. Leitsch, U. Hustadt et T. Tammet dans [FLHT01].

Définition 8.7 ($\mathcal{S}^+$ [FLHT01]) *Un ensemble de clauses S appartient à la classe $\mathcal{S}^+$ si pour toute clause C de S et pour tout littéral L de C :*

1. *si t est un terme fonctionnel apparaissant dans C alors $V(t) = V(C)$;*
2. *$Card(V(L)) \leqslant 1$ ou $V(L) = V(C)$.*

Les classes $\mathcal{S}^+$ et $\mathcal{C}$ sont assez proches mais incomparables. En effet, la classe $\mathcal{C}$ étend $\mathcal{S}^+$ de deux façons.

1. Elle autorise des littéraux atomiques de la forme $\pm P(x)$ pour des clauses contenant plusieurs variables. Ainsi la clause : $P(x), P(y), P(f(x, y))$ est dans $\mathcal{C}$ mais non dans $\mathcal{S}^+$.

2. Elle autorise des termes fonctionnels clos. Ainsi la clause $P(f(f(x,y),f(a,b))$, où a et b sont des constantes, appartient à $\mathcal{C}$ mais non à $\mathcal{S}^+$.

Inversement, la classe $\mathcal{S}^+$ permet des constantes au même niveau que les variables : $P(f(a,x,y))$ est une clause de $\mathcal{S}^+$ mais non de $\mathcal{C}$. D'autre part, pour les clauses C de $\mathcal{S}^+$, les termes de la forme $f(x_1,\dots,x_n)$ contenant toutes les variables de C peuvent être distincts dans un même littéral. Ainsi, la clause $P(f(f(x,y),g(x,y,x)))$ est une clause de $\mathcal{S}^+$ mais non de $\mathcal{C}$.

Nous pensons qu'il est possible d'étendre notre classe $\mathcal{C}$ de manière à inclure $\mathcal{S}^+$, tout en préservant la décidabilité mais une telle extension n'a pas d'applications pour les protocoles cryptographiques et nous ne l'avons donc pas considérée ici pour ne pas alourdir les preuves.

8.2.1.2 Définitions de l'ordre

Pour montrer la décidabilité de satisfaisabilité des ensembles de clauses de $\mathcal{C}$, nous allons utiliser des techniques de résolution ordonnée comme celles développées dans [FLHT01, GLM97]. Pour cela, nous introduisons un ordre sur les littéraux, basé sur la profondeur des constantes et des variables.

Définition 8.8 (profondeur) *Soit t un terme de $\mathcal{T}(\mathcal{F}\cup\mathcal{V})$. La* profondeur *de t, notée $|t|$, est définie inductivement de la manière suivante :*

$$\begin{array}{rcll} |a| & = & 1 & \textit{si } a \textit{ est une constante ou une variable,} \\ |f(t_1,\dots,t_n)| & = & 1+\max_{1\leqslant i\leqslant n}|t_i| & \textit{sinon.} \end{array}$$

Soit x une variable, la profondeur *de x dans t, notée $|t|_x$, est définie inductivement de la manière suivante :*

$$|y|_x = \begin{cases} 1 & \textit{si } y = x, \\ 0 & \textit{si } y \textit{ est une constante ou si } y \neq x, \end{cases}$$

$$|f(t_1,\dots,t_n)|_x = \begin{cases} 1+\max_{1\leqslant i\leqslant n}|t_i|_x & \textit{si } x \textit{ apparaît dans } f(t_1,\dots,t_n), \\ 0 & \textit{sinon.} \end{cases}$$

Nous définissons également la taille d'un terme.

Définition 8.9 (taille) *Soit t un terme de $\mathcal{T}(\mathcal{F}\cup\mathcal{V})$. La* taille *de t, notée $s(t)$, est définie inductivement de la manière suivante :*

$$\begin{array}{rcll} s(a) & = & 1 & \textit{si } a \textit{ est une constante ou une variable,} \\ s(f(t_1,\dots,t_n)) & = & 1+\sum_{1\leqslant i\leqslant n} s(t_i). & \end{array}$$

Définition 8.10 (ordre sur les littéraux) *Soient A et B deux littéraux.*

$$A \leqslant B \quad \textit{si} \quad |A| \leqslant |B| \quad \textit{et si} \quad \forall x \in V(A)\cup V(B) \quad |A|_x \leqslant |B|_x.$$

En particulier, si $A \leqslant B$, les variables de A sont contenues dans les variables de B : $V(A) \subseteq V(B)$. L'ordre strict $<$ associé à $\leqslant$ est défini par $A < B$ si $A \leqslant B$ et $A \neq B$.

Proposition 8.3 *La relation $\leqslant$ est un ordre partiel et relevable sur les littéraux.*

Preuve. Le fait que la relation $\leqslant$ est un ordre partiel est immédiat. Soient deux littéraux A et B tels que $A \leqslant B$ et soit θ une substitution quelconque. On vérifie :

$$|A\theta| = \max(|A|, \max_{x \in V(A)} (|A|_x + |x\theta|)).$$

Comme $|A| \leqslant |B| \leqslant |B\theta|$ et $|A|_x \leqslant |B|_x$, on obtient $|A\theta| \leqslant |B\theta|$. Soit maintenant x une variable de $A\theta$. Alors :

$$|A\theta|_x = \max(|A|_x, \max_{\substack{y \in V(A) \\ x \in V(y\theta)}} |A|_y + |y\theta|_x),$$

donc : $|A\theta|_x \leqslant |B\theta|_x$. On déduit $A\theta \leqslant B\theta$. $\square$

8.2.1.3 Quelques résultats sur l'unification

L'unification de littéraux de clauses de $\mathcal{C}$ conserve la profondeur maximale des termes.

Proposition 8.4 *Soient t_1 et t_2 deux termes tels que $t_1 = u[f(x_1, \ldots, x_k)/y]$ et $t_2 = v[g(y_1, \ldots, y_l)/y]$ avec u et v termes de $\mathcal{T}(\mathcal{F} \cup \{y\})$. Soit $\sigma = mgu(t_1, t_2)$ le plus général unificateur de t_1 et t_2. Quatre cas sont alors possibles (au renommage des variables près) :*

1. *$k = l$ et pour tout entier i tel que $1 \leqslant i \leqslant k$, la substitution vérifie : $x_i\sigma = y_i\sigma = z_i$;*
2. *ou pour tout entier i tel que $1 \leqslant i \leqslant k$, la substitution vérifie : $x_i\sigma = v_i[g(y_1, \ldots, y_l)]$, avec v_i sous-terme de v ;*
3. *ou pour tout entier i tel que $1 \leqslant i \leqslant l$, la substitution vérifie : $y_i\sigma = u_i[f(x_1, \ldots, x_k)]$, avec u_i sous-terme de u ;*
4. *ou la substitution σ est close et soit les $x_i\sigma$ sont des sous-termes clos de u ou v, soit les $y_i\sigma$ sont des sous-termes clos de u ou v.*

De plus, dans les trois premiers cas, on vérifie $|t_1\sigma| = \max(|t_1|, |t_2|)$ et dans le dernier cas : $|t_1\sigma| \leqslant 2\max(|t_1|, |t_2|)$.

La profondeur de $|t_1\sigma|$ peut effectivement augmenter.

Exemple 8.4 *Soit n un entier. Considérons les termes $t_1 = f(f(x_1, x_2), h^n(f(x_1, x_2)))$ et $t_2 = f(f(h^n(a), h^n(a)), h^n(f(y_1, y_2)))$. La substitution $\sigma = mgu(t_1, t_2)$ vérifie $x_1\sigma = y_1\sigma = h^n(a)$ et $x_2\sigma = y_2\sigma = h^n(a)$. On obtient alors $|t_1| = |t_2| = n + 3$ et $|t_1\sigma| = 2n + 3$.*

Preuve. Nous prouvons la propriété 8.4 par induction sur la somme des profondeurs de u et v.

Cas de base : si y n'apparaît pas dans u ou v alors on obtient le cas 4. Si $u = v = y$, alors on obtient le cas 1. Si $u = y$ et $v = f(v_1, \cdots, v_k)$, alors on obtient le cas 2 et symétriquement si $v = y$.

Supposons maintenant que $u = f(u_1, \ldots, u_n)$ et $v = f(v_1, \ldots, v_n)$. Alors pour tout entier i tel que $1 \leqslant i \leqslant n$, on vérifie $u_i[f(x_1, \ldots, x_k)/y]\sigma = v_i[g(y_1, \ldots, y_l)/y]\sigma$. Posons $\sigma_i = mgu(u_i[g(y_1, \ldots, y_l)/y], v_i[g(y_1, \ldots, y_l)/y])$. Pour tout entier i tel que $1 \leqslant i \leqslant n$, on vérifie $\sigma = \sigma_i\sigma_i'$ pour une certaine substitution σ_i'. Par hypothèse de récurrence, chaque σ_i vérifie l'un des quatre cas de la propriété. On vérifie qu'ils satisfont en fait tous le même cas sinon u et v ne seraient pas unifiables. S'ils satisfont tous les cas 1 ou 4, on conclut que σ satisfait le cas 1 ou 4. Supposons qu'ils satisfont tous le cas 2 (le cas 3 est similaire) : pour tous entiers i, j tels que $1 \leqslant i \leqslant n$ et $1 \leqslant j \leqslant k$, $\sigma_i(x_j) = u_j^i[g(y_1, \ldots, y_l)]$ donc $x_j\sigma = u_j^i[g(y_1, \ldots, y_l)]\sigma'_i$. Cela signifie que pour tous entiers i, i' tels que $1 \leqslant i, i' \leqslant n$, on obtient l'égalité : $u_j^i[g(y_1, \ldots, y_l)]\sigma'_i = u_j^{i'}[g(y_1, \ldots, y_l)]\sigma'_{i'}$ On utilise alors le lemme suivant.

Lemme 8.1 *Soient u et v deux termes ayant pour unique variable y et $\theta = mgu(u, v)$. Alors $y\theta = y$ ou $y\theta$ est un sous-terme de u ou v.*

Ce lemme est prouvé par induction sur la profondeur de u et v.

On en déduit que $y_j\sigma = y_j$ pour tout entier j tel que $1 \leqslant j \leqslant l$ (cas 2) ou $y_j\sigma$ est un sous-terme clos de de v (cas 4). □

Avec exactement la même technique de preuve, on montre des résultats similaires à la proposition 8.4 mais dans le cas où l'un des deux termes à unifier ne comporte qu'une seule variable.

Proposition 8.5 *Soient t_1 un terme de $\mathcal{T}(\mathcal{F} \cup \{x\})$ et t_2 un terme de $\mathcal{T}(\mathcal{F} \cup \{y\})$. Soit $\sigma = mgu(t_1, t_2)$ le plus général unificateur de t_1 et t_2. Quatre cas sont alors possibles :*

1. *$x\sigma = y\sigma = z$;*
2. *ou $x\sigma = v'$, avec v' sous-terme de t_2 ;*
3. *$y\sigma = u'$, avec u' sous-terme de t_1 ;*
4. *ou la substitution σ est close et $x\sigma$ ou $y\sigma$ est un sous-terme clos de t_1 ou t_2.*

De plus, dans les trois premiers cas, on vérifie $|t_1\sigma| = \max(|t_1|, |t_2|)$ et dans le dernier cas : $|t_1\sigma| \leqslant 2\max(|t_1|, |t_2|)$.

De la même manière, soient t_1 un terme de $\mathcal{T}(\mathcal{F} \cup \{x\})$ et $t_2 = v[g(y_1, \ldots, y_l)/y]$ avec $vn\mathcal{T}(\mathcal{F} \cup \{y\})$. Soit $\sigma = mgu(t_1, t_2)$ le plus général unificateur de t_1 et t_2. Trois cas sont alors possibles :

1. *$x\sigma = v'[g(y_1, \ldots, y_l)]$, avec v' sous-terme de v ;*
2. *ou pour tout entier i tel que $1 \leqslant i \leqslant l$, la substitution vérifie : $y_i\sigma = u_i$, avec u_i sous-terme de t_1 ;*
3. *ou la substitution σ est close et soit $x\sigma$ ou $y\sigma$ est un sous-terme clos de t_1 ou v.*

De plus, dans les deux premiers cas, on vérifie $|t_1\sigma| = \max(|t_1|, |t_2|)$ et dans le dernier cas : $|t_1\sigma| \leqslant 2\max(|t_1|, |t_2|)$.

8.2.2 Résultat de décidabilité

Nous pouvons maintenant énoncer notre résultat de décision.

Théorème 8.1 (décidabilité de $\mathcal{C}$) *Soit S un ensemble fini de clauses tel que S appartient à $\mathcal{C}$. La satisfaisabilité de S est décidable.*

Preuve. On applique à S la stratégie de résolution ordonnée par $\leqslant$, combinée à la règle de séparation : on construit la suite d'ensembles de clauses $(\mathcal{S}_i)_i$ définie par $\mathcal{S}_0 = \{S\}$, $\mathcal{S}_{2i+1} = \{S' \in SPLIT(S) \mid S \in \mathcal{S}_{2i}\}$ et $\mathcal{S}_{2i+2} = \{R_{\leqslant}(S) \mid S \in \mathcal{S}_{2i}\}$. De plus, on suppose que toute clause de la forme $L \vee L \vee C$ est immédiatement remplacée par la clause $L \vee C$. Une telle règle préserve la complétude.

D'après la proposition 8.2, cette stratégie est complète. Il nous reste à montrer qu'elle termine. Pour cela, nous définissons la profondeur d'une clause $\|C\|$ comme la profondeur maximale de ces littéraux :

$$\|C\| = \max_{L \in C} |L|.$$

Posons $N \stackrel{\text{def}}{=} \max_{C \in S} \|C\|$. Nous allons montrer que notre stratégie de résolution produit :

– soit des clauses closes de profondeur inférieure à $2N$;

– soit des clauses non closes de profondeur inférieure à N.

Supposons démontré le lemme suivant.

Lemme 8.2 *Soient C_1 et C_2 deux clauses non closes de $\mathcal{C}'$ et C le résolvant de C_1 et C_2 obtenu par résolution ordonnée sur $\leqslant$. La clause C est dans $\mathcal{C}$ et vérifie la propriété (P) suivante :*

– *soit C n'est pas close et $\|C\| \leqslant \max(\|C_1\|, \|C_2\|)$;*

– *soit C est close et $\|C\| \leqslant 2\max(\|C_1\|, \|C_2\|)$.*

De plus, notons que la résolution binaire de deux clauses dont l'une est close produit une clause close de profondeur plus petite que la clause close considérée.

Montrons par récurrence que pour tout entier i, les clauses des ensembles de $\mathcal{S}_i$ vérifient la propriété (P) et :

– si i est pair alors $\mathcal{S}_i \subset \mathcal{C}$;

– si i est impair alors $\mathcal{S}_i \subset \mathcal{C}'$.

La propriété de récurrence est vraie pour $i = 0$. Considérons maintenant $i \geqslant 1$. Si i est pair, alors $\mathcal{S}_i = \{R_{\leqslant}(S) \mid S \in \mathcal{S}_{i-1}\}$. Comme $\mathcal{S}_{i-1}$ vérifie l'hypothèse de récurrence, $\mathcal{S}_{i-1} \subset \mathcal{C}'$ donc, en appliquant le lemme 8.2, on déduit d'une part que $\mathcal{S}_i \subset \mathcal{C}$ et d'autre part que les clauses ajoutées par résolution vérifient la propriété (P). Si i est impair, $\mathcal{S}_i = \{S' \in \mathit{SPLIT}(S) \mid S \in \mathcal{S}_{i-1}\}$. Comme $\mathcal{S}_{i-1}$ vérifie l'hypothèse de récurrence et que la profondeur des clauses n'augmente par la règle de séparation, on déduit que $\mathcal{S}_i \subset \mathcal{C}'$ et que les clauses ajoutées vérifient la propriété (P).

Considérons maintenant le nombre maximal de variables utilisées par clauses des ensembles de $\mathcal{S}_i$. Soit k l'arité maximale des symboles de fonctions de l'alphabet $\mathcal{F}$ considéré. Les clauses de l'ensemble $\mathcal{C}'$ utilisent au plus k variables. En utilisant à nouveau la proposition 8.4, on vérifie que les clauses produites par résolution ordonnée de clauses de $\mathcal{C}'$ utilisent au plus k variables. On en déduit que pour $i \geqslant 1$ et $S \in \mathcal{S}_i$, les clauses de S utilisent au plus k variables.

Il nous suffit maintenant de montrer que les ensembles d'ensembles de clauses de profondeur inférieure à $2N$ et n'utilisant qu'au plus k variables sont en nombre fini. Pour cela, comme les clauses identiques au renommage près sont confondues, il suffit de montrer que les clauses de profondeur inférieure à $2N$ et n'utilisant qu'au plus k variables sont en nombre fini. On peut supposer que tous les littéraux d'une clause sont distincts grâce à la règle qui remplace toute clause $L \vee L \vee C$ par la clause $L \vee C$ donc cela revient à montrer que l'ensemble des termes de profondeur inférieure à $2N$ construits sur l'alphabet $\mathcal{F} \cup \{x_1, \ldots, x_k\}$ est fini, ce qui est le cas. □

Prouvons maintenant le lemme 8.2 introduit dans la preuve du théorème.

Preuve. Soient C_1 et C_2 deux clauses de $\mathcal{C}'$ et C le résolvant de C_1 et C_2 obtenu par résolution ordonnée sur $\leqslant$. Posons : $C_1 = C_1' \vee L_1$, $C_2 = C_2' \vee L_2$, et $\sigma = mgu(L_1, L_2)$. Le résolvant C vérifie : $C = C_1'\sigma \vee C_2'\sigma$. Nous distinguons les différents cas possibles pour L_1 et L_2.

Cas 1 : $L_1 = \pm P(x_1)$ et $L_2 = \mp P(x_2)$. Alors $x_1\sigma = x_2\sigma = x$ et par maximalité de $L_1\sigma$, les littéraux de $C_1'\sigma$ et de $C_2'\sigma$ sont de la forme $\pm P_j(x)$. On en déduit : $\|C\| = \|C_1\| = \|C_2\| = 1$.

Cas 2 : $L_1 = \pm P(x_1)$ et $L_2 = \mp P(u[f(y_1, \ldots, y_k)])$ (resp. $L_2 = \mp P(u(y))$). Alors $x_1\sigma = u[f(y_1, \ldots, y_k)]$ (resp. $x_1\sigma = u(y)$) et $y_i\sigma = y_i$ (resp. $y\sigma = y$) et, par maximalité de $L_1\sigma$, les littéraux de C_1' sont de la forme $\pm P_j(x_1)$. On en déduit d'une part que $C_2'\sigma = C_2'$ et d'autre part que, pour tous les littéraux L de C_1', le littéral $L\sigma$ est de la forme $\pm P_i(u[f(y_1, \ldots, y_k)])$ (resp. $\pm P_i(u(y))$). On conclut que la clause C est dans $\mathcal{C}'$ et vérifie : $\|C\| \leqslant \max(\|C_1\|, \|C_2\|)$.

Cas 3 : $L_1 = \pm P(u[f(x_1, \ldots, x_k)/z])$ et $L_2 = \mp P(v[g(y_1, \ldots, y_l)/z])$. D'après la proposition 8.4, quatre cas sont possibles :

1. $k = l$ et $x_i\sigma = y_i\sigma = z_i$ pour tout entier i tel que $1 \leqslant i \leqslant k$;
2. ou pour tout entier i tel que $1 \leqslant i \leqslant k$, la substitution vérifie : $x_i\sigma = v_i[g(y_1, \ldots, y_l)]$, avec v_i sous-terme de v ;
3. ou pour tout entier i tel que $1 \leqslant i \leqslant l$, la substitution vérifie : $y_i\sigma = u_i[f(x_1, \ldots, x_k)]$, avec u_i sous-terme de u ;
4. ou la substitution σ est close et soit les $x_i\sigma$ sont des sous-termes clos de u, soit les $y_i\sigma$ sont des sous-termes clos de v.

Dans le premier cas, on obtient que la clause C est dans $\mathcal{C}'$ et vérifie l'égalité : $\|C\| = \max(\|C_1\|, \|C_2\|)$. Les cas 2 et 3 sont symétriques, nous ne considérons donc que le cas 2.

Tout d'abord, $C_2'\sigma = C_2'$ et chaque littéral de $C_1'\sigma$ est de la forme $\pm P(w[g(y_1, \ldots, y_l)])$ donc la clause C est dans $\mathcal{C}'$. De plus, chaque littéral L de C_2' vérifies $|L\sigma| = |L| \leqslant \|C_2\|$. D'après la proposition 8.4, $|L_1\sigma| \leqslant \max(|L_1|, |L_2|) \leqslant \max(\|C_1\|, \|C_2\|)$. Considérons maintenant un littéral L de C_1'. Deux cas sont possibles pour le littéral L.

– Soit $|L\sigma| \leqslant |L_1\sigma|$. On vient de voir que $|L_1\sigma| \leqslant \max(\|C_1\|, \|C_2\|)$ donc : $|L\sigma| \leqslant \max(\|C_1\|, \|C_2\|)$.
– Soit $|L\sigma| > |L_1\sigma|$. Alors, par maximalité de $L_1\sigma$ dans $C_1'\sigma$, on déduit qu'il existe une variable y de $L_1\sigma$ telle que : $|L\sigma|_y < |L_1\sigma|_y$. Comme les seules variables de $L_1\sigma$ sont les y_i, on déduit que pour tout entier j tel que $1 \leqslant j \leqslant l$, l'inégalité $|L\sigma|_{y_j} \leqslant |L_1\sigma|_{y_j}$ est vérifiée, ce qui induit : $L = \pm P_i(u'[f(x_1, \ldots, x_k)/z])$ pour u' terme tel que $|u'|_z \leqslant |u|_z$. Maintenant : $|L\sigma| = |u'[f(x_1, \ldots, x_k)/z]\sigma| = \max(|u'|, |u'|_z + 1 + \max_{1 \leqslant i \leqslant k} |x_i\sigma|)$. De plus, $|u'| \leqslant |L| \leqslant \|C_1\|$ et $|u'|_z + 1 + \max_{1 \leqslant i \leqslant k} |x_i\sigma| \leqslant |L_1\sigma|$, donc $|L\sigma| \leqslant \max(\|C_1\|, \|C_2\|)$.

Dans tous les cas, on déduit l'inégalité attendue : $\|C\| \leqslant \max(\|C_1\|, \|C_2\|)$.

Il nous reste à traiter le quatrième cas de la proposition 8.4. La clause C est alors une clause close donc C est dans $\mathcal{C}'$. D'après la deuxième partie de la proposition : $|L_1\sigma| \leqslant 2\max(|L_1|, |L_2|) \leqslant 2\max(\|C_1\|, \|C_2\|)$. De plus, par maximalité de $L_1\sigma$ dans $C_1'\sigma$ et $C_2'\sigma$ et comme les clauses $C_1'\sigma$ et $C_2'\sigma$ sont closes, les littéraux L de $C_1'\sigma$ et $C_2'\sigma$ vérifient : $|L| \leqslant |L_1\sigma|$. On en déduit l'inégalité attendue : $\|C\| \leqslant 2\max(\|C_1\|, \|C_2\|)$.

Les cas $L_1 = \pm P(u[f(x_1, \ldots, x_k)/z])$ et $L_2 = \mp P(v(y))$ ou $L_1 = \pm P(u(x))$ et $L_2 = \mp P(v(y))$ sont similaires au cas 3 en utilisant cette fois la proposition 8.5. $\square$

Remarque : La résolution binaire ordonnée de deux clauses de $\mathcal{C}'$ peut effectivement produire une clause de $\mathcal{C}$. Considérons ainsi les clauses $C_1 = P(x_1), P(x_2), P(f(x_1, x_2))$ et $C_2 = P(f(y_1, y_2))$. Alors, par résolution ordonnée, on obtient la clause $P(x_1), P(x_2)$ qui n'est pas dans $\mathcal{C}'$.

8.2.3 Complexité

Calculons une borne supérieure de la complexité de la décision de la satisfaisabilité d'un ensemble S de $\mathcal{C}$.

En reprenant les notations introduites dans la preuve du théorème 8.1, on pose à nouveau $N = \max_{C \in S} \|C\|$. On note k l'arité maximal de l'alphabet $\mathcal{F}$ considéré et c le cardinal de $\mathcal{F}$. Posons u_n le cardinal de l'ensemble des termes de $T(\mathcal{F} \cup \{x_1, \ldots, x_k\})$ de profondeur maximale k. La suite u_n vérifie la relation de récurrence :

$$u_{n+1} \leqslant c \times u_n^k.$$

Comme u_1 est égal au nombre de constantes de $\mathcal{F}$ plus k, on déduit l'inégalité suivante :

$$u_n \leqslant (k+c)c^{(\frac{k^n - 1}{k-1})}.$$

On en déduit que le nombre d'ensembles de clauses distinctes de profondeur inférieure à $2N$ n'utilisant qu'au plus k variables est borné par $d \stackrel{\text{def}}{=} 2^{2 Card(\mathcal{P})(k+c)c^{(\frac{k^{2N}-1}{k-1})}}$, où $\mathcal{P}$ est l'ensemble de prédicats. De plus, lorsque l'on considère la suite des $\mathcal{S}_{2i}$, le cardinal maximal des ensembles de $\mathcal{S}_{2i}$ croît strictement avec i ou devient définitivement constant.

On en déduit que pour $i \geqslant 2d$, la suite $(\mathcal{S}_i)_i$ est constante. On vient donc de démontrer la propriété qui suit.

Proposition 8.6 (complexité de la saturation) *Soit S un ensemble de clauses de $\mathcal{C}$. La satisfaisabilité de C est décidable en $3 - EXPTIME$.*

Notre étude de complexité est assez grossière et pour que la borne supérieure donnée ici soit atteinte, il faudrait trouver un ensemble de clauses S telle que sa saturation demande d'engendrer tous les termes de profondeur bornée par la profondeur maximale des clauses de S et tous les ensembles possibles de clauses construites à partir de ces termes. Il est donc vraisemblable que la complexité théorique du problème soit plus faible.

D'autre part, il serait intéressant d'expérimenter cet algorithme sur des ensembles de clauses correspondant à la modélisation d'un protocole. Il est tout à fait possible que l'ensemble de clauses soit saturé en un temps (et un espace) raisonnable.

8.3 Extension au « ou » exclusif

8.3.1 Cadre

Nous introduisons ici les propriétés du « ou » exclusif ainsi que l'usage qui en est fait dans les protocoles cryptographiques, avant d'introduire notre nouvelle classe de clauses.

$$\begin{aligned} x \oplus (y \oplus z) &= (x \oplus y) \oplus z \\ x \oplus y &= y \oplus x \\ x \oplus 0 &= x \\ x \oplus x &= 0 \end{aligned}$$

FIGURE 8.1 - *Théorie équationnelle associée au symbole de fonction $\oplus$.*

8.3.1.1 Le « ou » exclusif

Le « ou » exclusif, aussi appelé xor et souvent noté $\oplus$ est une opération sur les bits :

$$\begin{array}{ll} 0 \oplus 0 = 0 & 0 \oplus 1 = 1 \\ 1 \oplus 0 = 1 & 1 \oplus 1 = 1. \end{array}$$

Ainsi $101 \oplus 110 = 011$. Dans la suite, on considérera le symbole de fonction $\oplus$, appliqué à des termes est dont la théorie équationnelle est représentée à la figure 8.1. En orientant les deux dernières règles de gauche à droite et en ajoutant la règle $((y \oplus x) \oplus x) \to y$, on obtient un système de réécriture convergent modulo l'associativité et la commutativité. Pour tout terme t de $T(\mathcal{F} \cup \{\oplus\} \cup \mathcal{V})$, on note $t\downarrow$ sa forme normale pour ce système de réécriture.

L'opération xor est très utilisée comme auxiliaire dans le chiffrement. En effet, pour chiffrer un message avec une clef k, on commence par découper le message en blocs de taille égale : $P_1, \ldots, P_n$. Une première méthode de chiffrement est le « Electronic Code Book » [CJ97] : elle consiste à chiffrer chacun des blocs séparément :

$$\{P_1 \cdots P_n\}_k = \{P_1\}_k \cdots \{P_n\}_k.$$

L'inconvénient de cette méthode est que le message chiffré peut être très facilement modifié en remplaçant un bloc chiffré par un autre. Une méthode plus souvent employée est le « Cipher Block Chaining » [CJ97] qui consiste à faire prendre le xor de chaque bloc avec le bloc chiffré précédent avant de le chiffrer avec la clef k :

$$\{P_1 \cdots P_n\}_k = C_0 C_1 \cdots C_n, \text{ avec } \left\{ \begin{array}{l} C_0 = I, \text{ bloc d'initialisation quelconque,} \\ C_i = \{C_{i-1} \oplus P_i\}_k \text{ pour tout entier } i \text{ tel que } 1 \leqslant i \leqslant n. \end{array} \right.$$

D'autres protocoles utilisent le xor comme algorithme de chiffrement. Ainsi, dans le protocole présenté dans [Pau97a], les clefs de sessions entre participants sont chiffrés par xor avec un autre message : la clef de session K_{ab} est envoyée sous la forme : $K_{ab} \oplus h(K_k, N_b)$. Mais P. Ryan et S. Schneider [RS98] ont trouvé une attaque sur ce protocole en utilisant les propriétés algébriques du xor alors que L. Paulson [Pau97a] a montré que si on suppose au contraire le chiffrement parfait, alors le protocole est correct. Ceci montre l'intérêt de tenir compte des propriétés du xor.

Notation : Dans toute la suite du chapitre, la relation $=$ représente l'égalité entre termes (ou littéraux) modulo les propriétés associatives et commutatives du xor, tandis que $=_\oplus$ représente l'égalité modulo la théorie complète du xor.

Comme le symbole $\oplus$ est associatif et commutatif, on peut considérer le symbole $\oplus$ comme un symbole variadique et se restreindre à des termes *aplatis* pour $\oplus$. Les sous-termes sont définis en conséquence.

Définition 8.11 (sous-terme) *Soit $\mathcal{F}$ un alphabet fini ne contenant par le symbole $\oplus$. L'ensemble des* sous-termes *d'un terme t de $T(\mathcal{F} \cup \{\oplus\})$, noté $ST(t)$ est défini inductivement de la manière suivante :*

$$\begin{array}{rcll} ST(a) & = & \{a\} & \textit{si } a \textit{ est une constante,} \\ ST(f(t_1,\ldots,t_n)) & = & \{f(t_1,\ldots,t_n)\} \bigcup_i ST(t_i) & \textit{si } f \in \mathcal{F}, \\ ST(t_1 \oplus \cdots \oplus t_n) & = & \{t_1 \oplus \cdots \oplus t_n\} \bigcup_i ST(t_i) & \textit{si les symboles de têtes des } t_i \textit{ sont dans } \mathcal{F}. \end{array}$$

On appelle symbole de tête *d'un terme t le symbole de fonction $f \in \mathcal{F} \cup \{\oplus\}$ tel que $t = f(t_1,\ldots,t_n)$.*

Exemple 8.5 *Les sous-termes de $f(a \oplus b \oplus g(x))$ sont : $f(a \oplus b \oplus g(x)), a \oplus b \oplus g(x), a, b, g(x), x$. En particulier, les termes $a \oplus b$ et $a \oplus g(x)$ ne sont pas sous-termes de $f(a \oplus b \oplus g(x))$.*

8.3.1.2 La classe $\mathcal{C}^{\oplus}$

Fixons un ensemble fini $\mathcal{F}$ de symboles de fonctions contenant le symbole constant 0, un ensemble $\mathcal{V}$ de variables et un ensemble fini $\mathcal{P}$ de symboles de prédicats. Nous définissons une nouvelle classe de clauses $\mathcal{C}^{\oplus}$ étendant la classe $\mathcal{C}$ définie précédemment.

Définition 8.12 *Un ensemble de clauses S appartient à $\mathcal{C}^{\oplus}$ si pour toute clause C de S, soit C ne contient qu'au plus une variable, soit pour tout littéral L de C :*

1. *ou bien $L = \pm P(x_i)$ pour un certain prédicat P ;*
2. *ou bien $L = \pm P(u[f(x_1,\ldots,x_n)/y])$ pour un certain prédicat P et un certain symbole de fonction $f \in \mathcal{F}$ tels que $\{x_1,\ldots,x_n\} = V(C)$ et u est un terme de $\mathcal{T}(\mathcal{F} \cup \{\oplus\} \cup \{y\})$;*
3. *ou $C = \neg P(x_1) \vee \neg P(x_2) \vee P(x_1 \oplus x_2)$, pour un prédicat P de $\mathcal{P}$.*

Remarque : Pour le second type de clause $\pm P(u[f(x_1,\ldots,x_n)/y])$, le symbole f ne doit pas être le symbole $\oplus$. Cependant, le symbole $\oplus$ peut apparaître dans u.

Nous verrons dans la partie 8.4 que la clause $C_0 \stackrel{\text{def}}{=} \neg I(x) \vee \neg I(y) \vee I(x \oplus y)$ permet de représenter le pouvoir de l'intrus à former le xor de deux messages. La clause C_0 est du troisième type de clauses définies ci-dessus. Nous verrons de plus que la nouvelle classe de clauses $\mathcal{C}^{\oplus}$ permet de modéliser des protocoles avec xor.

Dans la suite, l'ensemble S_0 représente l'ensemble des clauses de S du troisième type de clauses de la définition précédente.

Nous étendons nos définitions des profondeurs $|.|$ et $|.|_x$ sur les termes de $\mathcal{T}(\mathcal{F} \cup \{\oplus\} \cup \mathcal{V})$ en « oubliant » le symbole $\oplus$.

Définition 8.13 *La* profondeur $\|.\|$ *d'un terme de $\mathcal{T}(\mathcal{F} \cup \{\oplus\} \cup \mathcal{V})$ est définie inductivement :*

1. $\|a\| = 1$ *si* $a \in \mathcal{V}$ *ou si* a *est un symbole constant de* $\mathcal{F}$ *;*
2. $\|f(t_1, \ldots, t_k)\| = 1 + \max_{1 \leqslant i \leqslant k} \|t_i\|$ *si* $f \in \mathcal{F}$ *;*
3. $\|t_1 \oplus \cdots \oplus t_n\| = \max_{1 \leqslant i \leqslant n} \|t_i\|$ *si aucun des symboles de têtes des* t_i *n'est le symbole* $\oplus$.

La profondeur *d'un terme* t*, notée* $|t|$ *est définie par :* $|t| = \|t\downarrow\|$. *La profondeur d'une variable* x *d'un terme* t*, notée* $|.|_x$ *est définie de la même manière à ceci près que* $\|a\|_x = 1$ *si et seulement si* $a = x$. *On étend également la définition de la* profondeur *d'un littéral par* $|\pm P(t)| = |t|$ *et celle d'une clause par :*

$$|C| = \max_{L \in C} |L|.$$

8.3.2 Propriétés de l'ordre

La définition de l'ordre $\leqslant$ sur les littéraux est inchangée. Cependant, de nombreuses propriétés ne sont plus valables. L'objet de ce paragraphe est de rétablir des propriétés utiles à la preuve de la décidabilité de la satisfaisabilité de $\mathcal{C}^{\oplus}$.

8.3.2.1 Définitions

L'ordre $\leqslant$ n'est plus relevable comme on le voit dans l'exemple suivant.

Exemple 8.6 *Considérons les littéraux* $L_1 = P(a \oplus b)$ *et* $L_2 = P(f(x \oplus a) \oplus f(b \oplus a))$ *et la substitution* $x\theta = b$. *Alors* $L_1 \leqslant L_2$ *mais* $L_1\theta = P(a \oplus b) \not\leqslant L_2\theta = P(0)$.

C'est pourquoi nous introduisons une notion de substitution *sans réduction* et une notion d'ordre *étroitement relevable*. Une substitution est sans réduction si les termes obtenus par substitution sont en forme normale.

Définition 8.14 *Une substitution* σ *est dite* normalisée *si pour toute variable* x*, le terme* $x\sigma$ *est en forme normale.*

Une substitution σ *est dite* sans réduction *pour un ensemble de termes* S *si pour tout terme* t *de* S *:* $t\sigma\downarrow = t\downarrow\sigma$.

Notation : Soient $C_1, \ldots, C_n$ des clauses en forme normale. On note NS l'ensemble des substitutions normalisées et $CF(C_1, \ldots, C_n)$ l'ensemble des substitutions sans réduction pour les termes apparaissant dans les clauses $C_1, \ldots, C_n$.

Un ordre est étroitement relevable s'il est relevable pour les substitutions sans réduction.

Définition 8.15 (ordre étroitement relevable) *Un ordre* $\leqslant_{\mathcal{R}}$ *est étroitement relevable si pour tous atomes* A, B *et pour toute substitution* θ *sans réduction pour* B*, la relation* $A \leqslant_{\mathcal{R}} B$ *implique* $A\theta \leqslant_{\mathcal{R}} B\theta$.

En particulier, l'ordre $\leqslant$ défini au paragraphe 8.1.2 est étroitement relevable. L'intérêt d'un ordre étroitement relevable est que la complétude de la résolution ordonnée peut être rétablie à l'aide d'une règle de « réduction » permettant de traiter à part les cas où les substitutions induisent des réductions.

Proposition 8.7 *L'ordre* $\leqslant$ *est étroitement relevable sur les littéraux des clauses de* $\mathcal{C}^{\oplus}$.

Preuve. Comme $|B\theta| = \|B\theta\downarrow\| = \|B\downarrow\theta\|$ et $\|A\theta\downarrow\| \leqslant \|A\downarrow\theta\|$, on peut supposer que A et B sont en forme normale.

Montrons par induction sur la profondeur de B que si A et B sont des termes, si θ est une substitution sans réduction pour B telle que $A \leqslant B$, alors $|A\theta| \leqslant |B\theta|$ et pour toute variable x de $Var(B)$ et toute variable z de $Var(x\theta)$, les littéraux A et B vérifient : $|A\theta|_z \leqslant |B\theta|_z$.

Dans le cas de base, si B est une constante et $A \leqslant B$, alors A est un terme constant de profondeur 1 et $A\theta \leqslant B\theta$. Si B est une variable et $A \leqslant B$ alors $A = x \oplus t$ ou $A = t$ où t est un terme constant de profondeur 1. On obtient toujours $A\theta \leqslant B\theta$.

Supposons maintenant $|A| \leqslant |B|$ et pour toute variable x de $V(A) \cup V(B)$, $|A|_x \leqslant |B|_x$. Si A est une constante ou une variable, on déduit facilement le résultat. Sinon, $A = f(t_1, \ldots, t_k)$ et $B = g(u_1, \ldots, u_m)$ avec $f, g \in \mathcal{F} \cup \{\oplus\}$. Si f et g sont tous les deux dans $\mathcal{F}$ ou si $f = g = \oplus$, alors par définition de la profondeur, $\max_{1\leqslant i\leqslant k} |t_i| \leqslant \max_{1\leqslant i\leqslant m} u_i$ et $\max_{1\leqslant i\leqslant k} |t_i|_x \leqslant \max_{1\leqslant i\leqslant m} |u_i|_x$. Il existe des indices j_0 et j_x tels que pour tout entier i, $|t_i| \leqslant |u_{j_0}|$ et pour toute variable x de $V(A) \cup V(B)$, $|t_i|_x \leqslant |u_{j_x}|_x$. En appliquant l'hypothèse de récurrence, on obtient que $|t_i\theta|_z \leqslant |u_{j_x}\theta|_z$ pour tout entier i et toute variable z de $Var(x\theta)$. Il vient que $|A\theta|_z = \|A\theta\downarrow\|_z \leqslant \|A\theta\|_z \leqslant \|B\theta\|_z = |B\theta|_z$ car $B\theta$ et en forme normale. De même, $|A\theta| \leqslant |B\theta|$.

Si maintenant, $f = \oplus$ et $g \in \mathcal{F}$ (resp. $f \in \mathcal{F}$ et $g = \oplus$), alors $\max_{1\leqslant i\leqslant k} |t_i| \leqslant 1 + \max_{1\leqslant i\leqslant m} u_i$ et $\max_{1\leqslant i\leqslant k} |t_i|_x \leqslant 1 + \max_{1\leqslant i\leqslant m} |u_i|_x$ (resp. $1 + \max_{1\leqslant i\leqslant k} |t_i| \leqslant \max_{1\leqslant i\leqslant m} u_i$ et $1 + \max_{1\leqslant i\leqslant k} |t_i|_x \leqslant \max_{1\leqslant i\leqslant m} |u_i|_x$). Avec le même raisonnement, on démontre $A \leqslant B$.

□

8.3.2.2 Résultats sur l'unification

Le problème de l'unification modulo la théorie équationnelle du xor est un problème NP-complet [NGW96] et l'unification est finitaire. Cependant, nous avons besoin de résultats plus fins pour contrôler la profondeur des termes unifiés. Aussi, cette partie consiste en la description et la preuve de cinq propriétés techniques similaires à la propriété 8.4 pour l'unification de termes modulo la théorie de xor. La première propriété concerne l'unification de deux termes comportant une seule variable, identique pour les deux termes.

Proposition 8.8 *Soient u et v des termes de $T(\mathcal{F} \cup \{\oplus\} \cup \{x\})$ tels que $u \neq_\oplus v$. Soit u et v ne sont pas unifiables (modulo la théorie équationnelle du xor décrite à la figure 8.1), soit leur ensemble d'unificateurs est fini et chaque unificateur est de la forme $\sigma = \{x \mapsto w\}$ où w est un terme clos et :*

- *ou bien w est un sous-terme de u ou de v ;*
- *ou bien $w = w_1 \oplus w_2$ (sans réduction) avec w_1 et $x \oplus w_2$ sous-termes de u ou v.*

De plus, tout unificateur σ de u et v vérifie : $|x\theta| \leqslant \max\{|u|, |v|\}$.

La profondeur du terme unifié $u\sigma$ peut croître strictement.

Exemple 8.7 *Considérons $u = h^2(x) \oplus h^2(a) \oplus x$ et $v = h^2(x)$. L'unificateur de u et v est $x\sigma = h^2(a)$ et $u\sigma = h^4(a)$.*

Remarque : Nous pensons que si les termes u et v n'ont qu'une variable alors ils ne possèdent en fait qu'au plus un unificateur.

Preuve. (proposition 8.8). Montrons par récurrence sur $s(u) + s(v)$ que si $u\theta = v\theta$ alors soit $x\theta$ est un sous-terme clos de u ou v soit $x\theta = w_1 \oplus w_2$ (sans réduction) avec w_1 et $x \oplus w_2$ sous-termes clos de u ou v.

Si u et v sont des variables ou des constantes, le résultat est vrai.

Sinon, $u = u_1 \oplus \ldots \oplus u_n$ et $v = v_1 \oplus \cdots \oplus v_k$, tels que $u_1, \ldots, u_n, v_1, \ldots, v_k$ n'ont pas $\oplus$ pour symbole de tête. Trois cas sont possibles. Soit il existe deux entiers i, j tels que $u_i = v_j = x$. Dans ce cas, les solutions de l'équation $u = v$ sont les solutions de l'équation : $u_1 \oplus \ldots u_{i-1} \oplus u_{i+1} \oplus \ldots \oplus u_n = v_1 \oplus \ldots v_{j-1} \oplus v_{j+1} \oplus \ldots \oplus v_k$, ce qui nous permet de conclure en appliquant l'hypothèse de récurrence.

Soit aucun des termes u_i, v_j n'est une variable. Alors existe deux entiers i, j tels que $u_i\theta = u_j\theta$ ou $u_i\theta = v_j\theta$ ou $v_i\theta = v_j\theta$, ce qui nous permet de conclure en appliquant l'hypothèse de récurrence sauf si $u = u_1 = f(u'_1, \ldots, u'_m)$, $v = v_1 = f(v'_1, \ldots, v'_m)$ et $u'_i\theta = v'_i\theta$ pour tout entier i tel que $1 \leqslant i \leqslant m$. Dans ce cas, il existe au moins un entier i tel que $u_i \neq_{\oplus} v_i$, ce que nous permet de conclure en appliquant l'hypothèse de récurrence.

Soit (dernier cas) il existe un entier i tel que $u_i = x$ et aucun des termes $v)j$ n'est une variable. On obtient l'égalité : $x\theta = u'_1\theta \oplus \cdots u'_m\theta$ et chaque u'_i vérifie que $u'_i = u_j$ ou v_j pour un certain entier j. S'il existe deux entiers i, j tels que $u'_i\theta = u'_j\theta$ alors nous nous ramenons au cas précédent. Si tous les termes u'_i sont clos, alors il existe une unique solution $x = w_1 \oplus \ldots \oplus w_n$ qui satisfait la propriété demandée. Supposons maintenant que u'_1 n'est pas clos et considérons l'occurrence la plus *extérieure* de x dans u'_1, c'est-à-dire un terme $t = x \oplus t_1 \ldots \oplus t_k$ qui satisfait l'égalité $u'_1|_p = t$ pour un chemin p minimal. Posons $x\theta = w_1 \oplus \ldots \oplus w_m$ où les termes w_i sont distincts et n'ont pas $\oplus$ pour symbole de tête. De l'égalité : $w_1 \oplus \ldots \oplus w_m = u'_1\theta \downarrow \oplus u'_2\theta \downarrow \ldots \oplus u'_n\theta \downarrow$, on déduit qu'il existe un indice j tel que $w_j = u'_1\theta \downarrow$. En particulier, le terme w_j n'est pas un sous-terme strict de $u'_1\theta \downarrow$. Donc $w_j = t_i\theta \downarrow$ pour un certain entier i. Cela signifie que la substitution θ est solution de l'équation $t_i = u'_1$ avec t_1 sous-terme de u'_1. En appliquant l'hypothèse de récurrence, on déduit que la substitution θ vérifie la propriété demandée.

La dernière partie du lemme est une conséquence directe de la première. $\square$

Exemple 8.8 *Nous donnons ici quelques exemples d'unification :*

1. *L'équation* $x = f(x \oplus f(0))$ *a une unique solution* $x = f(0)$.
2. *L'équation* $x = f(x \oplus a \oplus f(x \oplus a \oplus f(a \oplus b))) \oplus b$ *a une unique solution* $\{x \mapsto b \oplus f(a \oplus b)\}$.
3. *L'équation* $x = f(x \oplus a) \oplus f(b) \oplus a \oplus b$ *a une unique solution* $\{x \mapsto a \oplus b\}$.

Nous obtenons un résultat similaire pour des termes de la forme $u = u'[f(x_1, \ldots, x_n)]$.

Proposition 8.9 *Soient* u *et* v *des termes tels que* $u \neq_{\oplus} v$, $u = u'[x' \mapsto f(x_1, \ldots, x_n)]$ *et* $v = v'[x' \mapsto f(x_1, \ldots, x_n)]$ *où* $u', v' \in T(\mathcal{F} \cup \{\oplus\} \cup \{x'\})$. *Ou bien* u *et* v *ne sont pas unifiables (modulo la théorie équationnelle du xor décrite à la figure 8.1), ou bien leur ensemble d'unificateurs est fini et chaque unificateur est de la forme* $\sigma = \{x \mapsto w\}$ *où* w *est un sous-terme clos de* u *ou* v. *De plus, pour tout unificateur* σ *de* u *et* v *vérifie :* $|x_i\theta) \leqslant \max\{|u|, |v|\}$.

Preuve. Nous reprenons la preuve de la proposition 8.8. Montrons par récurrence sur $s(u) + s(v)$ que si $u\theta = v\theta$ alors $x\theta$ est un sous-terme clos de u ou v.

Si u et v sont des constantes, le résultat est vrai.

Sinon, $u = u_1 \oplus \ldots \oplus u_n$ et $v = v_1 \oplus \cdots \oplus v_k$, tels que $u_1, \ldots, u_n, v_1, \ldots, v_k$ n'ont pas $\oplus$ pour symbole de tête. Il n'y a plus qu'un seul cas possibles : aucun des termes u_i, v_j n'est une variable. Il existe donc deux entiers i, j tels que $u_i\theta = u_j\theta$ ou $u_i\theta = v_j\theta$ ou $v_i\theta = v_j\theta$, ce qui nous permet de conclure en appliquant l'hypothèse de récurrence sauf si $u = u_1 = f(u'_1, \ldots, u'_m)$, $v = v_1 = f(v'_1, \ldots, v'_m)$ et $u'_i\theta = v'_i\theta$ pour tout entier i tel que $1 \leqslant i \leqslant m$. Dans ce cas, il existe au moins un entier i tel que $u_i \neq_\oplus v_i$, ce que nous permet de conclure en appliquant l'hypothèse de récurrence.

La dernière partie de la proposition est à nouveau une conséquence directe de la première. □

La troisième propriété concerne l'unification de deux termes comportant une seule variable, distincte pour les deux termes.

Proposition 8.10 *Soient u et v deux termes tels que $u \neq_\oplus v$, $Var(u) \cap Var(v) = \emptyset$ et $Var(u) \subseteq \{x\}, Var(v) \subseteq \{y\}$. Ou bien u et v ne sont pas unifiables (modulo la théorie équationnelle du xor décrite à la figure 8.1), ou bien tout plus général unificateur θ de u et v vérifie (au renommage des variables près) :*

- *soit il existe des sous-termes clos $w_1, \ldots, w_k$ de u ou v tels que $x\theta = w_1 \oplus \ldots \oplus w_k$ (resp. $y\theta = w_1 \oplus \ldots \oplus w_k$) et $y\theta$ est clos (resp. $x\theta$ est clos),*
- *soit $x\theta = z \oplus t_1 \oplus \cdots \oplus t_k$ et $y\theta = t'_1 \oplus \cdots \oplus t'_k \oplus u_1 \oplus \cdots u_n\theta$ (sans réduction), où les t_i et t'_i sont des sous-termes clos de u ou v, $n \geqslant 1$ et les u_i sont des sous-termes non clos de u,*
- *soit symétriquement $y\theta = z \oplus t_1 \oplus \cdots \oplus t_k$ et $x\theta = t'_1 \oplus \cdots \oplus t'_k \oplus v_1 \oplus \cdots v_n\theta$, où les t_i et t'_i sont des sous-termes clos de u ou v, $n \geqslant 1$ et les v_i sont des sous-termes clos de v.*

Preuve. Nous prouvons à nouveau la première partie du lemme par induction sur $s(u)+s(v)$.

Si u et v sont des variables, le résultat est vrai.

Sinon u et v vérifient les égalités $u = u_1 \oplus \ldots \oplus u_n$ et $v = v_1 \oplus \cdots \oplus v_k$, telles que le symbole de tête des termes $u_1, \ldots, u_n, v_1, \ldots, v_k$ n'est pas $\oplus$. À nouveau, quatre cas sont possibles.

Cas 1 : $u = u_1 = f(u'_1, \ldots, u'_m)$ et $v = v_1 = f(v'_1, \ldots, v'_m)$. Alors pour tout indice i, $u'_i\theta \downarrow = v'_i\theta \downarrow$. Il existe au moins un indice j tel que $u'_j \neq v'_j$ On applique alors l'hypothèse d'induction à $u'_j = v'_j$ et on considère un plus général unificateur σ de $u'_j = v'_j$ tel que $\theta = \sigma\sigma'$ pour un certain σ'. Si σ est clos alors $\sigma = \theta$ et on obtient alors le résultat attendu car les sous-termes clos de u'_i, v'_j sont également des sous-termes clos de u, v. Sinon, par symétrie, on peut supposer que $x\sigma = z \oplus t_1 \oplus \ldots \oplus t_q$ et $y\sigma = (s_1 \oplus \ldots \oplus s_p \oplus w_1 \oplus \ldots \oplus w_r)\sigma \downarrow$ tels que $p \geqslant 1$, les $s_1, \ldots, s_p$ sont des sous-termes non clos de u'_j et $t_1, \ldots, t_k, w_1, \ldots, w_r$ sont des sous-termes clos de u'_j, v'_j. Si $u\sigma \downarrow = v\sigma \downarrow$, alors le résultat est prouvé. Sinon, σ' est un plus général unificateur de $u\sigma, v\sigma$. En appliquant la proposition 8.8, il vient $\sigma' = \{z \mapsto w\}$ et ou bien w est un sous-terme clos de $u\sigma, v\sigma$, ou bien $w = w'_1 \oplus w'_2$ et w'_1 est un sous-terme clos de $u\sigma, v\sigma$, w'_2 est clos et le terme $z \oplus w'_2$ est un sous-terme de $u\sigma, v\sigma$. Remarquons que les sous-termes clos de $u\sigma, v\sigma$ sont nécessairement des sous-termes clos de u, v car $x\sigma$ et $y\sigma$ ne sont pas clos et leur sous-termes clos sont des sous-termes de u, v. Cela signifie que $x\theta = w \oplus t_1 \oplus \ldots \oplus t_q$ est une somme de sous-termes clos de u, v (et donc $y\theta$ est clos) ou bien $x\theta = w'_1 \oplus w'_2 \oplus t_1 \oplus \ldots \oplus t_q$ et w'_1 est un sous-terme clos de u, v, w'_2 est clos et $z \oplus w'_2$ est un sous-terme de $u\sigma, v\sigma$. La dernière partie implique que le terme w'_2 est lui-même une somme de sous-termes clos de u, v. On en déduit que $x\theta$ est également une somme de sous-termes clos de u, v.

Cas 2 : aucun des termes u_i, v_i n'est une variable et $n + k > 2$. Il existe alors des indices i et j tels que $u_i\theta = u_j\theta$, $u_i\theta = v_j\theta$ ou $v_i\theta = v_j\theta$. En retirant alors les termes u_i, u_j (resp. u_i, v_j, resp. v_i, v_j) des sommes $u_1 \oplus \ldots \oplus u_n$ et $v_1 \oplus \ldots \oplus v_k$, on obtient deux termes u', v' tels que $u'\theta \downarrow = v'\theta \downarrow$ et $s(u') + s(v') < s(u) + s(v)$. Ou bien $u' = v'$ et on applique l'hypothèse de récurrence à la paire u_i, u_j ou u_i, v_j ou v_i, v_j, ou bien nous pouvons appliquer l'hypothèse d'induction à u', v'. On considère alors un plus général unificateur σ de u', v' tel que $\theta = \sigma\sigma'$ et on procède exactement comme au cas 1.

Cas 3 : il existe un indice i tel que $u_i = x$ mais qu'aucun des termes v_i n'est une variable. S'il existe des indices j, k tels que $u_k\theta = u_j\theta$, $u_k\theta = v_j\theta$ ou $v_k\theta = v_j\theta$ alors on peut appliquer l'hypothèse de récurrence et procéder comme dans les cas précédents. Nous renommons les termes u_i de manière à obtenir l'égalité suivante : $x\theta \oplus u_1\theta \downarrow \oplus \cdots \oplus u_n\theta \downarrow = v_1\theta \downarrow \oplus \cdots \oplus v_k\theta \downarrow$. Si tous les termes u_i et v_j sont clos, alors il existe une unique solution $x = u_1 \oplus \ldots \oplus u_n \oplus v_1 \oplus \cdots \oplus v_k$, qui satisfait les propriétés demandées. Si les termes u_i sont clos mais non les v_j, alors il existe également une unique solution $x = u_1 \oplus \ldots \oplus u_n \oplus v_1 \oplus \cdots \oplus v_k$, qui satisfait les propriétés demandées. Supposons maintenant que u_1 n'est pas un terme clos et considérons $x \oplus t_1 \ldots \oplus t_k$ l'occurrence la plus extérieure de x dans u_1 (définie comme dans la preuve de la propriété 8.8). Posons $x\theta = w_1 \oplus \ldots \oplus w_m$ tels que les termes w_i soient distincts et que leur symbole de tête ne soit pas $\oplus$. De l'égalité $w_1 \oplus \ldots \oplus w_m = u_1\theta \downarrow \oplus u_2\theta \downarrow \ldots \oplus u_n\theta \downarrow v_1\theta \downarrow \oplus \ldots \oplus v_k\theta \downarrow$, nous déduisons qu'il existe un indice j tel que $w_j = u_1\theta \downarrow$. En particulier, le terme w_j n'est pas un sous-terme strict de $u_1\theta \downarrow$. Donc $w_j = t_i\theta \downarrow$ pour un certain indice i. Cela signifie que la substitution θ est solution de l'équation $t_i = u_1$. Comme t_i est un sous-terme de u_1, nous pouvons appliquer la proposition 8.8) et en déduire le résultat attendu.

Cas 4 : il reste à examiner le cas le plus compliqué : $u_i = x$ pour un certain indice i et $v_j = y$ pour un certain indice j. Nous renommons les termes u_i et v_j de manière à obtenir l'égalité : $x\theta \oplus u_1\theta \downarrow \oplus \cdots \oplus u_n \downarrow \theta = y\theta \oplus v_1\theta \downarrow \oplus \cdots \oplus v_k\theta \downarrow$. On peut supposer que les termes $u_i\theta \downarrow$ et $v_j\theta \downarrow$ sont tous distincts deux à deux sinon nous sommes ramenés au cas précédent. Si tous les termes u_i sont clos, alors toute solution θ la plus générale vérifie $y\theta = z \oplus w_1 \oplus \cdots \oplus w_l$ et $x\theta = v'_1 \oplus \cdots \oplus v'_k \oplus w'_1 \oplus \cdots \oplus w'_{l'}$ tels que les v'_i sont les v_i non clos et les w_i et w'_i sont distincts deux à deux et l'ensemble $\{w_1, \ldots, w_l, w'_1, \ldots, w'_{l'}\}$ est égal à l'union des u_i et des v_i clos. De telles solutions satisfont les conditions de la propriété. La propriété est également vérifiée si tous les v_i sont clos. Nous supposons donc que les u_1 et v_1 ne sont pas clos. Posons $x\theta = w_1 \oplus \cdots \oplus w_l \oplus s_1 \oplus \cdots s_m$ et $y\theta = w'_1 \oplus \cdots \oplus w'_{l'} \oplus s_1 \oplus \cdots s_m$ avec $l + l' = n + k$ et chaque terme w_i vérifie l'égalité $w_i = u_j\theta$ ou $w_i = v_j\theta$ pour un certain indice j, et symétriquement pour les termes w'_i. Soit $x \oplus t_1 \ldots \oplus t_k$ l'occurrence la plus extérieure de x dans u_1. Si l'un des termes w_i est égal à l'un des $t_j\theta$, alors on déduit $t_j\theta = u_s\theta$ ou $v_s\theta$ pour un certain indice s, ce qui permet d'appliquer l'hypothèse de récurrence. On conclut de la même manière si l'un des termes w'_i est égal à l'un des $t_j\theta$.

Il reste le cas où les termes w_i et $t_j\theta$ sont distincts deux à deux et les termes w'_i et $t_j\theta$ sont distincts deux à deux. Alors les termes w_i sont des sous-termes stricts de $u_1\theta$ (car nous avons considéré l'occurrence la plus extérieure de x). Cela signifie que le terme $u_1\theta$ est distinct de chacun des termes w_i et donc $u_1\theta = w'_j$ pour un certain indice j. De la même manière les termes w'_i sont des sous-termes stricts de $v_1\theta$ et $v_1\theta = w_m$ pour un certain indice m. On obtient donc les relations suivantes :

$$w_m \lhd u_1\theta = w'_j \lhd v_1\theta = w_m,$$

où $\lhd$ représente l'ordre sous-terme. Or ces relations ne peuvent être satisfaites, ce qui nous permet de conclure la démonstration. $\square$

Nous déduisons de la proposition précédente une extension au cas où u ou v sont de la forme : $u'[x \to f(x_1, \dots, x_n)]$.

Proposition 8.11 *Soient u, u' et v des termes avec variables tels que : $Var(u) \cap Var(v) = \emptyset$, $u = u'[x \to f(x_1, \dots, x_n)]$ et $Var(u') \subseteq \{x\}, Var(v) \subseteq \{y\}$. Ou bien u et v ne sont pas unifiables (modulo la théorie équationnelle du xor décrite à la figure 8.1), ou bien tout plus général unificateur θ de u et v vérifie (au renommage des variables près) :*

- *soit les $x_i\theta$ sont des sous-termes clos de u ou v et $y\theta$ est un terme clos.*
- *soit $y\theta$ est sous-terme clos de u ou v et les $x_i\theta$ sont des termes clos.*
- *soit $x_i\theta = x_i$ pour tout indice i et $y\theta = t_1 \oplus \cdots \oplus t_k \oplus u_1 \oplus \cdots u_n$, où les termes t_i sont des sous-termes clos de u ou v, $m \geqslant 1$ et les termes u_i sont des sous-termes non clos de u qui ne sont pas non plus des variables,*
- *soit symétriquement, $y\theta = z \oplus t_1 \oplus \cdots \oplus t_k$ et $x_i\theta = v_i\theta$, où les t_i sont des sous-termes clos de u ou v et $f(v_1, \dots, v_n)$ est un sous-terme de v.*

Soient u, u', v et v' des termes avec variables tels que : $Var(u) \cap Var(v) = \emptyset$, $u = u'[x \to f(x_1, \dots, x_n)]$, $v = v'[y \to g(y_1, \dots, y_k)]$ et $Var(u') \subseteq \{x\}, Var(v') \subseteq \{y\}$. Ou bien u et v ne sont pas unifiables (modulo la théorie équationnelle du xor décrite à la figure 8.1), ou bien tout plus général unificateur θ de u et v vérifie (au renommage des variables près) :

- *soit pour tous indices i, j les $x_i\theta$ et $y_j\theta$ sont des sous-termes clos de u ou v,*
- *soit $x_i\theta = x_i$ pour tout indice i et $y_j\theta = u_j$ pour tout indice j, tels que $g(u_1, \dots, u_k)$ est un sous-terme de u,*
- *soit $y_i\theta = y_i$ pour tout indice i et $x_j\theta = v_j$ pour tout indice j, tels que $f(v_1, \dots, v_n)$ est un sous-terme de v,*
- *soit $f = g$ et $x_i\theta = y_i\theta = z_i$.*

Preuve. Nous donnons ici seulement les grandes lignes de la preuve. Pour la première partie de la proposition, l'équation $u = v$ est équivalente à $u' = v \wedge x = f(x_1, \dots, x_n)$. On applique alors la proposition 8.10 à l'équation $u' = v$ et on simplifie les conclusions en tenant compte du fait que $x\theta$ doit avoir f pour symbole de tête.

Pour la deuxième partie de la proposition, l'équation $u = v$ est équivalente à $u = v' \wedge y = g(y_1, \dots, y_k)$. On applique alors la première partie de la proposition et on simplifie les conclusions en tenant compte du fait que $y\theta$ doit avoir g pour symbole de tête. □

Exemple 8.9 *Nous donnons ici deux exemples d'unification entre deux termes avec des variables distinctes.*

1. *L'équation $g(a \oplus f(y \oplus f(a)), f^n(y)) = g(x, f^n(x))$ avec n entier tel que $n \leqslant 2$, a pour unique solution : $x = y = a \oplus f(a)$. Le terme unifié obtenu a une profondeur plus grande que les termes de départ.*
2. *L'équation $y \oplus f(y \oplus f(f(a))) = x \oplus f(x \oplus a)$ a pour solution : $\{x \mapsto a \oplus f(a), y \mapsto a \oplus f(f(a))\}$. Une telle solution σ n'est pas sans réduction car $f(a) = f(y \oplus f(f(a)))\sigma \downarrow$.*

Nous avons construit l'ordre $\leqslant$ de manière à ce qu'il soit stable par substitutions sans réduction. La proposition qui suit montre que l'on peut toujours se ramener à des substitutions sans réduction. Le résultat a déjà été établi dans un contexte plus général [CLS03] (avec un nombre arbitraire de variables dans les termes) en devinant à l'avance tous les partages de variables possibles. Ici, le résultat est beaucoup plus simple mais plus précis car nous exploitons la forme particulière de nos termes.

Proposition 8.12 *Pour toute clause C de $\mathcal{C}^{\oplus}$ en forme normale, il existe un nombre fini de clauses $C_1, \ldots, C_n$ telles que :*

$$\{C\sigma \downarrow \mid V(C\sigma) = \emptyset, \sigma \in NS\} = \bigcup_{i=1}^{n} \{C_i\sigma \mid V(C_i\sigma) = \emptyset, \sigma \in CF(C_1, \ldots, C_n)\}.$$

De plus, si $C \notin S_0$, chacune des clauses C_i vérifient l'un des trois cas suivants :
- *$C_i = C$,*
- *C_i est close et $|C_i| \leqslant 2 \times |C|$,*
- *$V(C) = \{x\}$ et $C_i = C\{x' \mapsto y \oplus t_i\} \downarrow$ où t_i est une somme de sous-termes clos de C.*

Preuve. Différents cas sont possibles suivant la forme de C.

Si C est dans l'ensemble S_0 alors on pose $C_1 = C$, $C_2 \stackrel{\text{def}}{=} \neg P(x \oplus y) \vee \neg P(y) \vee P(x)$ et

$$C_3 \stackrel{\text{def}}{=} \neg P(x \oplus z) \vee \neg P(y \oplus z) \vee P(x \oplus y).$$

Montrons que ces trois clauses conviennent. Considérons une substitution normalisée σ telle que la clause $C\sigma \downarrow$ est close. Posons $x\sigma \downarrow = t_1 \oplus \ldots \oplus t_k$ et $y\sigma \downarrow = u_1 \oplus \ldots \oplus u_m$ (sans réduction) tels que les termes t_i et u_i n'ont pas $\oplus$ pour symbole de tête. Si $\{t_1, \ldots, t_k\} \subseteq \{u_1, \ldots, u_m\}$ ou $\{u_1, \ldots, u_m\} \subseteq \{t_1, \ldots, t_k\}$, on obtient une instance sans réduction de C_2, si les deux ensembles sont disjoints, on obtient une instance sans réduction de C_1 et dans le cas général, si $\{t_1, \ldots, t_k\} \cap \{u_1, \ldots, u_m\} = \{v_1, \ldots, v_n\}$ est un ensemble non vide, on obtient une instance sans réduction de C_3 (on associant par exemple $v_1 \oplus \ldots \oplus v_n$ à la variable z).

Considérons maintenant une clause C qui n'a qu'une seule variable x. Soient T l'ensemble de ses sous-termes clos et $\sigma_1, \ldots, \sigma_n$ les substitutions $\{x \mapsto x' \oplus t_1 \oplus \ldots \oplus t_m\}$ et $\{x \mapsto t_1 \oplus \ldots \oplus t_m\}$ où $\{t_1, \ldots, t_m\} \subseteq T$. Nous définissons alors $C_i \stackrel{\text{def}}{=} C\sigma_i \downarrow$. Soit σ une substitution normalisée. Nous montrons le résultat par induction sur le nombre de réductions nécessaires pour récrire $C\sigma$ en sa forme normale. Si $C\sigma$ ne contient pas de redex alors $C\sigma$ est une instance sans réduction de $C\{x \mapsto x'\}$ (avec $m = 0$). Sinon, considérons un redex apparaissant à une profondeur maximale dans $C\sigma$: $u_1\sigma \oplus \ldots \oplus u_n\sigma$ où $u_1\sigma = v_1 \oplus v_2$ et $u_2\sigma = v_1 \oplus v_3$ (v_2 et v_3 peuvent être vide) et les termes u_i n'ont le $\oplus$ comme symbole de tête. Si v_2 et v_3 sont vides, comme C est en forme normale, σ est un unificateur de u_1 et u_2. D'après la proposition 8.8, cela signifie que σ assigne à x une somme de sous-termes clos de C et donc $C\sigma \downarrow$ est égal à l'un des C_i. Si ni v_2 ni v_3 ne sont vides, comme les termes u_1 et u_2 n'ont le $\oplus$ comme symbole de tête, u_1 et u_2 sont des variables. Comme C n'a qu'une variable, cela signifie $u_1 = u_2 = x$ ce qui contredit C en forme normale. Il reste donc à considérer le cas où v_3 est vide mais non v_2. Par un même raisonnement, on déduit que $u_1 = x$. Comme nous avons choisi le redex le plus extérieur, le terme u_2 est nécessairement clos car il ne peut pas contenir strictement $x\sigma$ (sinon nous n'aurions pas $x\sigma = u_2\sigma$). On pose $\sigma_0 = \{x \mapsto v_1 \oplus x'\}$. On obtient alors $C\sigma \downarrow = (C\sigma_0) \downarrow \{x' \mapsto v_2\} \downarrow$ et la clause $C\sigma_0 \downarrow \{x \mapsto v_2\}$ contient strictement moins de redex que $C\sigma$. En appliquant l'hypothèse de récurrence à $C\sigma_0 \downarrow$ et comme l'ensemble des clauses associées à $C\sigma_0 \downarrow$ et l'ensemble des clauses associées à C sont identiques, on déduit que $C\sigma_0 \downarrow \{x \mapsto v_2\} \downarrow = C_i\sigma_i$ pour une substitution σ_i sans réduction et donc $C\sigma \downarrow = C_i\sigma_i$. De plus, par construction des clauses C_i, ou bien C_i n'est pas close et $C_i = C\{x \mapsto x' \oplus t_1 \oplus \cdots \oplus t_m\}$ ou bien C_i est close et $C_i = C\{x \mapsto t_1 \oplus \cdots \oplus t_m\}$ donc $|C_i| \leqslant 2|C|$.

Considérons enfin le cas où C est une clause de la forme $C'\{x \mapsto f(x_1, \ldots, x_n)\}$ où C' contient une unique variable x. On définit les clauses C_i par l'ensemble des clauses obtenues par unification de deux sous-termes u et v tels que $u \oplus v$ apparaît dans C puis en normalisant :

$$\{C_i\} = \{C\sigma \downarrow \mid \sigma = mgu(u, v) \text{ et } u \oplus v \text{ apparaît dans } C\}.$$

D'après la proposition 8.10, toutes les clauses C_i sont closes. Considérons une substitution normalisée σ. Tout redex de $C\sigma$ est de la forme $u\sigma \oplus v\sigma$ où $u \oplus v$ apparaît dans C (car le $\oplus$ ne peut être appliqué directement à une variable). Il suit que soit $C\sigma$ est irréductible, soit $C\sigma$ est l'une des clauses C_i définies ci-dessus. Dans ce dernier cas, C_i est close et d'après la proposition 8.10, $|x\theta| \leqslant \max(|u|, |v|)$. Comme $|C_i| = |C_i\sigma| \leqslant \max(|C|, |C|_x + |x\theta|)$, on déduit $|C_i| \leqslant 2\max(|u|, |v|) \leqslant 2|C|$. □

Notation : Soit C une clause de $\mathcal{C}^{\oplus}$, l'ensemble des clauses C_i associées à C dans la preuve de la proposition 8.12 est noté $c(C)$.

8.3.3 Résultat de décidabilité

Nous pouvons maintenant énoncer notre résultat de décidabilité.

Théorème 8.2 (décidabilité de $\mathcal{C}^{\oplus}$) *Soit S un ensemble fini de clauses tel que S appartient à l'ensemble $\mathcal{C}^{\oplus}$. La satisfaisabilité de S est décidable.*

Le reste de cette partie est dédié à la preuve de ce théorème. Il ne suffit pas de mettre en place une stratégie de réduction ordonnée sur S car une telle stratégie ne termine pas pour les clauses de S_0. Aussi nous développons une stratégie de résolution *ad hoc* dont nous montrons la terminaison et la complétude. Nous étudions enfin la complexité de notre stratégie : elle est non élémentaire.

8.3.3.1 Saturation

Notre première règle de résolution est la résolution binaire ordonnée de deux clauses C_1 et C_2 sur un littéral L si le plus général unificateur σ considéré pour obtenir l'atome résolu est sans réduction pour les littéraux de C_1 et C_2.

Nous pouvons en effet nous restreindre (d'après la proposition 8.12) à des unificateurs sans réduction à condition d'appliquer la transformation décrite dans la proposition. Cette transformation est appelée *règle de réduction*.

Cependant, il n'est pas suffisant de se restreindre à la résolution ordonnée sans réduction pour assurer la terminaison. En effet, si nous résolvons répétitivement une clause de S_0 de la forme $\neg I(x) \vee \neg I(y) \vee I(x \oplus y)$ avec elle-même (au renommage près), on obtient un ensemble infini de clauses. Aussi, nous interdisons complètement la résolution sur des clauses de S_0 et nous introduisons à la place deux règles dites d'*extension*. Dans les grandes lignes, la règle d'inférence a pour but de déduire la clause $P(s \oplus u) \vee C \vee D$ des clauses $P(s \oplus t) \vee C$ et $P(t \oplus u) \vee D$ si t est maximal parmi s, t, u. La situation est similaire au cas de la règle de transitivité pour laquelle L. Bachmair and H. Ganzinger [BG01] ont mis en place une règle spéciale d'inférence appelée « ordered chaining ».

L'ensemble des ces règles de déduction est résumé à la figure 8.2. Nous combinons comme précédemment ces règles de déduction à la règle de séparation : nous définissons à partir de S une suite d'ensembles $\mathcal{S}_i$ définie par $\mathcal{S}_0 = \{S\}$, $\mathcal{S}_{2i+1} = \{S' \in SPLIT(S) \mid S \in \mathcal{S}_{2i}\}$ et $\mathcal{S}_{2i+2} = \{R_{\leqslant}(S) \mid S \in \mathcal{S}_{2i}\}$, où $R_{\leqslant}(S)$ désigne l'ensemble des clauses qui peuvent être obtenus à partir de S en appliquant l'une des règles de déduction définies à la figure 8.2. On suppose toujours que toute clause $L \vee L \vee C$ est immédiatement remplacée par la clause $L \vee C$. Cette transformation préserve la complétude.

Ces règles sont correctes et laisse $\mathcal{C}^{\oplus}$ invariant.

Proposition 8.13 *Les règles de déduction présentées à la figure 8.2 sont correctes : l'ensemble S est satisfaisable si et seulement si l'un des ensembles de clauses de $\mathcal{S}_i$ est satisfaisable. D'autre part, pour tout entier i, si S est dans $\mathcal{C}^{\oplus}$, tout ensemble de clauses de $\mathcal{S}_i$ (obtenue à partir de S) est dans $\mathcal{C}^{\oplus}$.*

Preuve. La correction des règles est immédiate. Nous allons montrer que chaque ensemble obtenu reste dans $\mathcal{C}^{\oplus}$. Comme les règles de déduction s'appliquent à des clauses décomposées et n'appartenant pas à S_0, on peut supposer que les clauses vérifient l'une des deux propriétés suivantes :

1. la clause ne contient qu'une seule variable ;
2. les littéraux maximaux de la clause sont de la forme $\pm P_i(u)[x \mapsto f(x_1, \ldots, x_n)]$ où $\{x_1, \ldots, x_n\}$ forme l'ensemble des variables de la clause.

Nous allons montrer que pour chacune des règles de déduction, l'appartenance à l'ensemble $\mathcal{C}^{\oplus}$ est conservée.

Résolution binaire ordonnée Si les deux clauses $C_1 = \neg P(t) \vee C$ et $C_2 = P(u) \vee C'$ ne contiennent chacune qu'une seule variable, alors la proposition 8.10 nous assure que le résolvant ne contient qu'au plus une variable donc est dans $\mathcal{C}^{\oplus}$. Si l'une des deux clauses est de la forme 2, alors la proposition 8.11 permet également de conclure que le résolvant est dans $\mathcal{C}^{\oplus}$.

Factorisation D'après la proposition 8.8 et 8.9, la clause obtenue par factorisation est close donc appartient à $\mathcal{C}^{\oplus}$.

Réduction Si C est une clause ne contenant qu'une seule variable, les clauses de $c(C)$ ne contiennent également qu'une seule variable par construction. Si C est de la forme $C'[x \mapsto f(x_1, \ldots, x_n)]$ alors $c(C)$ ne contient que des clauses closes qui appartiennent donc à $\mathcal{C}^{\oplus}$.

Explosion À nouveau les clauses obtenues sont closes donc dans $\mathcal{C}^{\oplus}$.

Extension Les propositions 8.10 et 8.11 nous permettent de conclure que les clauses obtenues restent dans $\mathcal{C}^{\oplus}$. □

8.3.3.2 Terminaison

Nous pouvons maintenant montrer la terminaison de la procédure.

Proposition 8.14 *Soit S un ensemble de clauses de $\mathcal{C}^{\oplus}$ alors la suite d'ensembles $\mathcal{S}_i$ construite à partir de S est finie.*

Résolution binaire

$$\frac{\neg P(t) \vee C \quad P(u) \vee C'}{C\sigma \vee C'\sigma}$$

Si $\sigma \in mgu(t,u)$ est sans réduction pour les littéraux de C et C', et $P(t)\sigma \not\leq (C \vee C')\sigma$.

Factorisation

$$\frac{L_1 \vee L_2 \vee C}{(L_1 \vee C)\sigma}$$

Si $\sigma \in mgu(L_1, L_2)$ est sans réduction pour la clause $L_1 \vee L_2 \vee C$ et $L_1\sigma \not\leq C\sigma$.

Réduction

$$\frac{C}{C'}$$

Si $C' \in c(C)$.

Explosion

$$\frac{P(t \oplus u) \vee C}{(P(t \oplus u) \vee C)\sigma \downarrow}$$

Si t est clos, $u\sigma$ est clos et $u\sigma < t$.

Extension 1

$$\frac{\neg P(t) \vee C \quad P(u_1 \oplus u_2) \vee C'}{(C \vee C' \vee \neg P(t_2 \oplus u_2))\theta} \quad \text{Si} \begin{cases} P(x \oplus y) \vee \neg P(x) \vee \neg P(y) \in S_0 \\ t = t_1 \oplus t_2 (\text{ou } t = t_1 \text{ et } t_2 = 0) \\ V(t) = V(t_1), \theta \in mgu(t_1, u_1) \\ \theta \text{ sans réduction pour } t, u_1 \oplus u_2, t_2 \oplus u_2 \\ (C \vee C' \vee \neg P(t_2 \oplus u_2))\theta \not\geq t_1\theta. \end{cases}$$

Extension 2

$$\frac{P(t) \vee C \quad P(u_1 \oplus u_2) \vee C'}{(C \vee C' \vee P(t_2 \oplus u_2))\theta} \quad \text{Si} \begin{cases} \neg P(x) \vee \neg P(y) \vee P(x \oplus y) \in S_0 \\ t = t_1 \oplus t_2 (\text{ou } t = t_1 \text{ et } t_2 = 0) \\ V(t) = V(t_1), \theta \in mgu(t_1, u_1) \\ \theta \text{ sans réduction pour } t, u_1 \oplus u_2, t_2 \oplus u_2 \\ (C \vee C' \vee \neg P_3(t_2 \oplus u_2))\theta \not\geq t_1\theta. \end{cases}$$

Ces règles s'appliquent seulement à des clauses décomposées et n'appartenant pas à S_0.

FIGURE 8.2 - *Règles de déduction.*

Preuve. Nous allons borner la profondeur des clauses des ensembles de $\mathcal{S}_i$. La première difficulté est que la profondeur peut augmenter par résolution binaire ordonnée.

Exemple 8.10 *Considérons la clause $C_1 = P(h^n(x)) \vee P(f(x, h^n(a)))$ et la clause $C_2 = P(h^n(y)) \vee \neg P(f(y \oplus h^n(a), h^n(a)))$. Les littéraux de chacune des clauses sont incomparables. La substitution $\sigma_1 = \{x \mapsto y \oplus h^n(a)\}$ est un plus général unificateur de $P(f(x, h^n(a)))$ et $P(f(y \oplus h^n(a), h^n(a)))$. La clause résultante par résolution sur ces deux littéraux est : $C = P(h^n(y \oplus h^n(a))) \vee P(h^n(y))$. Cette clause est de profondeur strictement plus grande que C_1 et C_2.*

Nous allons voir que ce n'est plus le cas si on se limite à des résolutions binaires ordonnées pour lesquelles l'unificateur considéré est sans réduction. Mais pour pouvoir se limiter à des substitutions sans réduction, il faut appliquer la règle de réduction qui augmente la profondeur des clauses.

Posons N la profondeur maximale des clauses de S :

$$N = \max_{C \in S} |C|.$$

Posons T l'ensemble des sommes des sous-termes clos de S. Nous allons en fait montrer par induction sur i que pour toute clause C d'un ensemble de $\mathcal{S}_i$, ou bien C est close et $|C| \leqslant 2N$ ou bien il existe un terme t de T tel que $|C[x \mapsto x' \oplus t] \downarrow| \leqslant N$ (éventuellement $t = 0$).

La propriété est vraie pour $i = 0$.

Pour montrer la propriété par induction, nous allons utiliser le lemme suivant.

Lemme 8.3 *Soient L_1, L_2 deux littéraux tels que $Var(L_1) = Var(L_2) = \{x\}$ et tels que L_1 et L_2 soient des littéraux d'une clause C telle que $|C\{x \mapsto x' \oplus t\} \downarrow| \leqslant N$ pour un certain terme t de T. Soit σ une substitution sans réduction (pour L_1 et L_2) telle que $L_1\sigma \not\geqslant L_2\sigma$ et $Var(x\sigma) = \{y\}$. Alors $|L_1\sigma| \leqslant \max(N, |L_2\sigma|)$.*

De plus, considérons une deuxième substitution $\theta = \{y \mapsto x \oplus t'\}$ telle que pour tout chemin p tel que $|p| = |L_1\sigma|_y$ (resp. $|p| = |L_2\sigma|_y$) et $L_1\sigma|_p = y \oplus v$ (resp. $L_2\sigma|_p = y \oplus v$), le terme t' apparaît dans v : $v = t \oplus v'$. Un tel chemin sera appelée chemin d'occurrence de y de longueur maximale dans $|L_1\sigma|_y$ (resp. $|L_2\sigma|_y$). Alors $|L_1\sigma\theta \downarrow| \leqslant \max(N, |L_2\sigma\theta \downarrow|)$.

Preuve. Prouvons la première partie du lemme. Supposons $|L_1\sigma| > |L_2\sigma|$, alors par maximalité de $L_2\sigma$, les littéraux doivent vérifier l'inégalité $|L_1\sigma|_y \leqslant |L_2\sigma|_y$ et donc $|L_1|_x \leqslant |L_2|_x$. Alors $|L_1\sigma| = |L_1|$ car pour toute position p d'un terme de la forme $x \oplus u$ dans L_1 (avec u éventuellement égal à 0), $|p| + |x\sigma| \leqslant |L_2|_x + |x\sigma| \leqslant |L_2\sigma|$. Supposons par l'absurde que l'on ait également $|L_1\sigma| > N$ (voir figure 8.3). Alors il existe une position p telle que $L_1|_p = x \oplus u$ et $|L_1\sigma| = |L_1| = |p| + |u| > N$ car $|L_1\{x \mapsto x' \oplus t\} \downarrow| \leqslant N$. Soit p' une position de longueur maximale pour la variable x dans L_2 : $L_2|_{p'} = x \oplus v$. Ou bien $|v| \geqslant |u|$ auquel cas $|L_1\sigma| \leqslant |L_2\sigma|$, ou bien $|v| < |u|$. Alors comme $|L_1\{x \mapsto x' \oplus t\} \downarrow| \leqslant N$, on doit avoir $|t| = |u|$ donc $|L_2\{x \mapsto x' \oplus t\} \downarrow| \geqslant |L_2|_x + |t| > N$ (car $|v| < |t|$ implique $|(v \oplus t) \downarrow| = |t|$), contradiction. Nous pouvons donc conclure $|L_1\sigma| \leqslant N$, donc dans tous les cas $|L_1\sigma| \leqslant \max(N, |L_2\sigma|)$.

Montrons maintenant la deuxième partie du lemme. Comme le terme t' apparaît à toutes les occurrences maximales de la variable y, les inégalités suivantes sont vérifiées : $|L_1\sigma\theta \downarrow| \leqslant |L_1\sigma|$ et $|L_2\sigma\theta \downarrow| \leqslant |L_2\sigma|$. Supposons $|L_1\sigma| > N$ et $|L_1\sigma\theta \downarrow| > N$ (si $|L_1\sigma| \leqslant N$ ou $|L_1\sigma\theta \downarrow| \leqslant N$, nous pouvons conclure immédiatement). Si de plus $|L_2\sigma\theta \downarrow| = |L_2\sigma|$ alors la première

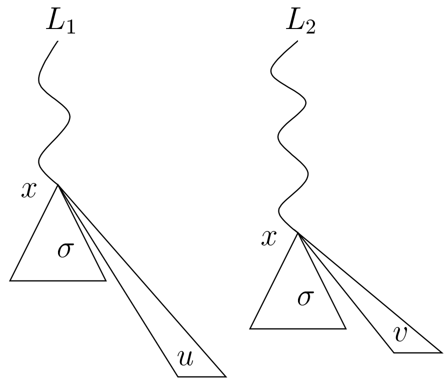


FIGURE 8.3 - *Cas* $|L_1\sigma| > |L_2\sigma|$ *et* $|L_1\sigma| > N$.

partie du lemme nous permet de conclure. Considérons donc le cas où $|L_2\sigma\theta\downarrow| < |L_2\sigma|$. Cela signifie que $|L_2\sigma| = |L_2\sigma|_y + |t|$. Supposons par l'absurde que $|L_1\sigma|_y > |L_2\sigma|_y$ alors $|L_1\sigma| \geqslant |L_1\sigma|_y + |t| > |L_2\sigma|_y + |t| = |L_2\sigma|$, ce qui contredit la maximalité de $L_2\sigma$. Donc $|L_1\sigma|_y \leqslant |L_2\sigma|_y$ ce qui implique $|L_1|_x \leqslant |L_2|_x$. Soit p une position telle que $L_2|_p = x \oplus u$ et $|p| = |L_2|_x$. Trois cas sont possibles.

- Soit la profondeur $|L_1\sigma\theta\downarrow|$ est atteinte pour un chemin prolongeant une position de x dans L_1. Alors nous pouvons conclure en utilisant l'inégalité $|L_1\sigma|_y \leqslant |L_2\sigma|_y$ que $|L_1\sigma\theta\downarrow| \leqslant |L_2\sigma\theta\downarrow|$ (voir figure 8.4).

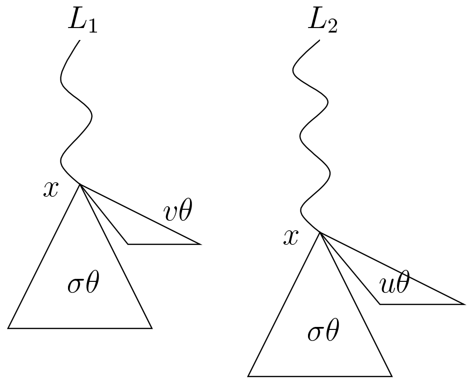


FIGURE 8.4 - *Cas* $|L_1\sigma|_y \leqslant |L_2\sigma|_y$ *et* $|L_1\sigma\theta\downarrow|$ *est atteinte pour un chemin prolongeant une position de* x *dans* L_1.

- Soit la profondeur $|L_1\sigma\theta\downarrow|$ est atteinte pour un chemin p'' prolongeant une position p' telle que $L_1\sigma|_{p'} = y \oplus v$ et $|p''| = |p'| + |v| > N$ (voir figure 8.5). Nous pouvons supposer que p'' ne prolonge pas une position de x dans L_1 sinon nous sommes ramenés au cas précédent ce qui bous permet de conclure. Donc $L_1|_{p'} = x \oplus v'$, c'est-à-dire $L_1\sigma|_{p'} = x\sigma \oplus v' = y \oplus v_1 \oplus v'$ avec $v = v_1 \oplus v'$ et $x\sigma = y \oplus v_1$. Si $|v_1| \geqslant |v'|$ alors la profondeur $|L_1\sigma\theta\downarrow|$ est également atteinte pour un chemin prolongeant une position de x dans L_1. Sinon $|v_1| < |v'|$. Comme $|L_2\{x \mapsto x' \oplus t\}\downarrow| \leqslant N$, on doit avoir $|u| = |v'|$,

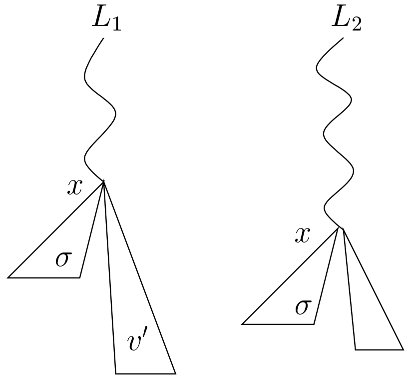


FIGURE 8.5 - *Cas $|L_1\sigma|_y \leqslant |L_2\sigma|_y$ et $|L_1\sigma\theta\downarrow|$ est atteinte pour un chemin p'' prolongeant une position p' telle que $L_1\sigma|_{p'} = y \oplus v$ et $|p''| = |p'| + |v| > N$.*

donc $|L_1\sigma\theta\downarrow| = |p'| + |v| \leqslant |p| + |u| \leqslant |L_2| \leqslant L_2\sigma\theta\downarrow$.

– Soit la profondeur $|L_1\sigma\theta\downarrow|$ est atteinte pour un chemin p'' prolongeant une position p' telle que $L_1\sigma|_{p'} = y \oplus v$, $v = v_1 \oplus u$, $t' = t_1 \oplus u$ et $|t_1| > |v_1|$ (figure 8.5). On a alors

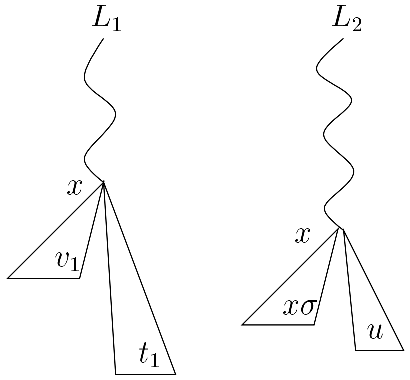


FIGURE 8.6 - *Cas $|L_1\sigma|_y \leqslant |L_2\sigma|_y$ et $|L_1\sigma\theta\downarrow|$ est atteinte pour un chemin p'' prolongeant une position p' telle que $L_1\sigma|_{p'} = y \oplus v$, $v = v_1 \oplus u$, $t' = t_1 \oplus u$ et $|t_1| > |v_1|$.*

$|L_1\sigma\theta\downarrow| = |p'| + |t_1|$. Nous pouvons supposer que p'' ne prolonge pas une position de x dans L_1 sinon nous sommes ramenés au cas précédent ce qui nous permet de conclure. Donc $L_1|_{p'} = x \oplus v'$, c'est-à-dire $L_1\sigma|_{p'} = x\sigma \oplus v' = y \oplus v_2 \oplus v'$ avec $v = v_2 \oplus v'$ et $x\sigma = y \oplus v_2$.

Comme $|L_2\{x \mapsto x' \oplus t\}\downarrow| \leqslant N$ et $|L_1\{x \mapsto x' \oplus t\}\downarrow| \leqslant N$, on déduit que $v' = v_1' \oplus t_0$, $u = u_1 \oplus t_0$ avec $t = t_0 \oplus t_0'$ et $|L_1|_x + |v_1'|, |L_1|_x + |t_0|, |L_2|_x + |u_1|, |L_2|_x + |t_0| \leqslant N$. Donc $t' = t_1' \oplus t_2' \oplus t_3'$ tels que $t_0 = t_2' \oplus t_0''$, $(v' \oplus t')\downarrow = (v_1' \oplus t_1')\downarrow \oplus t_0'' \oplus t_3'$, $(u \oplus t')\downarrow = (u_1 \oplus t_1')\downarrow \oplus t_0'' \oplus t_3'$ et $|p''| = |p'| + |t_3'|$. En utilisant à nouveau l'inégalité $|L_1\sigma|_y \leqslant |L_2\sigma|_y$ on conclut que $|L_1\sigma\theta\downarrow| \leqslant |L_2\sigma\theta\downarrow|$.

□

Nous allons maintenant examiner toutes les règles de déduction.

Réduction Si C' est une clause obtenue à partir de C par réduction alors d'après la proposition 8.12, ou bien C' est close et $|C'| \leqslant 2N$ ou bien C' n'est pas close et il existe un terme t de T tel que $|C'\{x \mapsto x' \oplus t\}| \leqslant N$.

Factorisation ou explosion Si C' est obtenu par factorisation ou explosion à partir de C, alors C' est close et d'après la proposition 8.8, $C' = C\sigma$ où $|\sigma| \leqslant N$ donc $|C'| \leqslant 2N$.

Résolution binaire

$$\frac{\neg P(t) \vee C \quad P(u) \vee C''}{C\sigma \vee C''\sigma}$$

Supposons $C' = C\sigma \vee C''\sigma$ avec $\sigma \in mgu(t, u)$. Les clauses $\neg P(t) \vee C$ et $P(u) \vee C''$ sont soit closes, soit n'ont qu'une variables, soit sont de la forme $C_1[x \to f(x_1, \ldots, x_n)] \vee \pm P_i(x_i)$. Nous allons uniquement traiter le cas où chacune des clauses ne comporte qu'une variable : $Var(t) = \{x\}$ et $Var(u) = \{y\}$. D'après la proposition 8.10, si la substitution σ est close alors $|C'| \leqslant 2N$. Si la substitution σ n'est pas close alors nous pouvons supposer sans perte de généralité que $y\sigma = z \oplus t_1 \oplus \cdots \oplus t_k$ et $x\sigma = (u_1 \oplus \cdots \oplus u_m)\sigma \oplus t'_1 \oplus \cdots \oplus t'_k$, où les t_i et les t'_i sont des sous-termes clos de u ou t, $m \leqslant 1$ et les u_i sont des sous-termes non clos de u. De plus, il existe des termes $t_1, t_2 \in T$ (éventuellement égaux à 0) tels qaue $(P(t) \vee C)\{x' \mapsto x \oplus t_1\} \downarrow\leqslant N$ et $(P(u) \vee C')\{y' \mapsto y \oplus t_2\} \downarrow\leqslant N$.

En appliquant le lemme 8.3 et en utilisant le fait que $L\sigma \not\succ P(t)\sigma$ et $L'\sigma \not\succ P(u)\sigma$, on déduit que $|L\sigma| \leq \max(N, |P(t)\sigma|)$ pour tout littéral L de C et $|L'\sigma| \leq \max(N, |P(u)\sigma|)$ pour tout littéral L' de C''. Maintenant comme $P(t)\sigma = P(u)\sigma = P(u)\{y \mapsto z \oplus t_1 \oplus \cdots \oplus t_k\}$ et que les clauses sont en forme normale, les termes t_i apparaissent à chaque occurrence de longueur maximale de la variable z dans $C\sigma \vee C''\sigma$. Considérons la substitution θ telle que $z\theta = y \oplus t_1 \oplus \cdots \oplus t_k$. On déduit de la deuxième partie du lemme 8.3 que $|L\sigma\theta \downarrow| \leq \max(N, |P(t)\sigma\theta \downarrow|)$ pour tout littéral L de C et $|L'\sigma\theta \downarrow| \leq \max(N, |P(u)\sigma\theta \downarrow|)$ for pour tout littéral L' de C''. Remarquons en fait que $P(u)\sigma\theta \downarrow = P(u)$. Supposons $|P(u)| > N$. Comme $(P(u) \vee C')\{x \mapsto x \oplus t_2\} \downarrow\leqslant N$, cela signifie qu'il existe un terme t'_2 tel que t'_2 apparaît à chaque occurrence de longueur maximale de la variable y dans C'' donc apparaît également à chaque occurrence de longueur maximale de la variable z dans $C\sigma\theta \downarrow$ et tel que $(P(u) \vee C')\{x \mapsto x \oplus t'_2\} \downarrow\leqslant N$. Considérons la substitution θ telle que $y\theta = y \oplus t'_2$. En appliquant à nouveau le lemme 8.3, on obtient $|L\sigma\theta\theta' \downarrow| \leq \max(N, |P(t)\sigma\theta\theta' \downarrow|)$ pour tout littéral L de C et $|L'\sigma\theta\theta' \downarrow| \leq \max(N, |P(u)\sigma\theta\theta' \downarrow|)$ pour tout littéral L' de C''. Comme $|P(u)\sigma\theta\theta' \downarrow| \leq N$ et $P(t)\sigma\theta\theta' \downarrow = P(u)\sigma\theta\theta' \downarrow$, on déduit $(C' = C\sigma \vee C''\sigma)\theta\theta' \downarrow\leqslant N$, d'où le résultat.

Extension Si C' est obtenue par l'une des règles d'extension, le cas est identique au cas de la résolution binaire. En effet, supposons :

$$\frac{P(t_1 \oplus t_2) \vee C \quad P(u_1 \oplus u_2) \vee C''}{C \vee C'' \vee \neg P(t_2 \oplus u_2))\theta,}$$

et $C' = (C \vee C'' \vee \neg P_3(t_2 \oplus u_2))\theta$, $\theta \in mgu(t_1, u_1)$ θ sans réduction pour $t, u_1 \oplus u_2, t_2 \oplus u_2$ et $(C \vee C' \vee \neg P(t_2 \oplus u_2))\theta \not\succ t_1\theta$. En considérons la résolution binaire (pour des clauses qui ne sont pas forcément dans $\mathcal{S}_i$!) :

$$\frac{P(t_1), P(t_2) \vee C \quad P(u_1), P(u_2) \vee C''}{C \vee C'' \vee \neg P(t_2)P(u_2))\theta,}$$

on déduit le résultat attendu en utilisant le cas précédent.

Nous venons de montrer que pour toute clause C d'un ensemble de $\mathcal{S}_i$, ou bien C est close et $|C| \leqslant 2N$ ou bien il existe un terme t de T tel que $|C[x \mapsto x' \oplus t] \downarrow| \leqslant N$ (éventuellement $t = 0$). Or si $|C[x \mapsto x' \oplus t] \downarrow| \leqslant N$, la clause C vérifie $|C| \leqslant 2N$. On a donc montré que toutes les clauses vérifient $|C| \leqslant 2N$.

Le nombre de variables distinctes par clauses est borné par l'arité maximale des symboles de fonctions. D'après la règle de factorisation, on peut supposer que tous les littéraux d'une clause sont distincts. Il suffit donc de montrer que les termes de $T(\mathcal{F} \cup \{\oplus\} \cup \{x_1, \ldots, x_k\})$ de profondeur bornée sont en nombre fini. Cela se vérifie facilement par récurrence sur la profondeur. Posons u_n le cardinal de l'ensemble de terme de profondeur inférieure ou égale à n. L'ensemble des termes de profondeur 1 est l'ensemble des constantes et des variables ainsi que des termes obtenus en appliquant le symbole $\oplus$ à un ensemble de constantes ou de variables distinctes. On en déduit que $u_1 \leqslant |\mathcal{P}(\mathcal{F} \cup \{x_1, \ldots, x_k\})| \leqslant 2^{|\mathcal{F}|+k}$. L'ensemble des termes de profondeur $n+1$ est l'ensemble des termes obtenus par application d'un symbole fonctionnel à des termes de profondeur inférieure ou égale à n ainsi que des termes obtenus en appliquant le symbole $\oplus$ à des termes de profondeur inférieure ou égale à $n+1$ et ayant pour symbole de tête un symbole différent de $\oplus$. On en déduit que u_{n+1} est fini et que : $u_{n+1} \leqslant 2^{|\mathcal{F}|u_n}$. □

8.3.3.3 Complétude

Le plus difficile dans la preuve du théorème 8.2 est de montrer la complétude de notre procédure de saturation. Soit C un ensemble de clauses de $\mathcal{C}^{\oplus}$, la suite $\mathcal{S}_i$ est finie d'après la proposition de terminaison 8.14. On définit $\mathcal{S}^* = \mathcal{S}_j$ où j est l'indice minimal tel que $\mathcal{S}_j = \mathcal{S}_i$ pour tout entier i supérieur à j.

Proposition 8.15 (complétude) *Soit S un ensemble de clauses de $\mathcal{C}^{\oplus}$ et $\mathcal{S}^*$ l'ensemble qui lui est associé par notre procédure de saturation. L'ensemble S est insatisfaisable si et seulement si $\bot \in S'$ pour tout ensemble S' de $\mathcal{S}^*$.*

Le reste de ce paragraphe est consacré à la preuve de cette proposition. Pour cela, nous commençons par étendre notre ordre $\leqslant$ en un ordre total sur les littéraux clos.

Définition 8.16 *Soit $<$ un ordre total quelconque sur les prédicats de S et sur les symboles fonctionnels de S. L'ordre $<$ est étendu sur l'ensemble $\mathcal{F} \cup \{\oplus\}$ par $\oplus < f$ pour tout symbole $f \in \mathcal{F}$. L'ordre $\widetilde{<}$ est défini inductivement sur les termes clos en forme normale de la manière suivante :*

1. *si $|t_1| < |t_2|$ alors $t_1 \widetilde{<} t_2$;*
2. *si $|t_1| = |t_2|$, $t_1 = f_1(u_1, \ldots, u_k)$ et $t_2 = f_2(v_1, \ldots, v_l)$ avec $f_1 < f_2$ alors $t_1 \widetilde{<} t_2$;*
3. *si $|t_1| = |t_2|$, $t_1 = f(u_1, \ldots, u_k)$ et $t_2 = f(v_1, \ldots, v_k)$ avec $f \in \mathcal{F}$ et s'il existe un entier i tel que $1 \leqslant i \leqslant k$, $u_i \widetilde{<} v_i$ pour tous les entiers j strictement inférieurs à i, $u_j \widetilde{\leq} v_j$ (ordre lexicographique sur les fils), alors $t_1 \widetilde{<} t_2$;*
4. *si $|t_1| = |t_2|$, $t_1 = u_1 \oplus \cdots \oplus u_k$ et $t_2 = v_1 \oplus \cdots \oplus v_l)$ et si $\{u_1, \ldots, u_k\} \widetilde{<}_{mul} \{v_1, \ldots, u_l\}$, alors $t_1 \widetilde{<} t_2$.*

L'ordre $\widetilde{<}_{mul}$ désigne l'ordre multi-ensemble associé à $\widetilde{<}$.

L'ordre $\widetilde{<}$ est étendu aux littéraux positifs : soient L_1 et L_2 deux littéraux tels que $L_1 = P_1(t_1)$ et $L_2 = P_2(t_2)$. Alors $L_1 \widetilde{<} L_2$ si :

- *$t_1 \tilde{<} t_2$;*
- *ou $t_1 = t_2$ et $P_1 < P_2$.*

Par construction de $\tilde{<}$, l'ordre $\tilde{<}$ étend l'ordre $\leqslant$. De plus, $\tilde{<}$ est un ordre total sur les termes clos et les littéraux positifs clos.

La preuve de condition nécessaire la proposition 8.15 est immédiate. Supposons en effet que $\perp \in S'$ pour tout ensemble S' de $\mathcal{S}^*$. D'après la proposition 8.13, les règles de déduction sont correctes donc l'ensemble S est insatisfaisable. Pour montrer la réciproque, nous supposons que S est insatisfaisable et nous considérons un ensemble S' de $\mathcal{S}^*$ et une interprétation partielle maximale (pour l'ordre sur les littéraux) « la plus à gauche » de $S' \cup S_0$. Nous allons montrer que cette interprétation est vide, ce qui montre : $\perp \in S'$. Précisons tout d'abord ce que nous entendons par interprétation partielle maximale « la plus à gauche ».

Définition 8.17 *Soit $\mathcal{I}$ une interprétation partielle d'un ensemble de clauses S. $\mathcal{I}$ est dite* maximale *s'il existe un littéral L, tel que pour tout littéral $L' \tilde{<} L$, L est interprété par $\mathcal{I}$ et si l'extension de $\mathcal{I}$ à $\mathcal{I}(L) = 0$ et l'extension de $\mathcal{I}$ à $\mathcal{I}(L) = 1$ falsifient toutes deux une instance d'une clause de S.*

De plus, nous définissons un ordre lexicographique $I >_{lex} J$ sur les interprétations partielles maximales I et J de la manière suivante. Soit L l'élément maximal de la base de Herbrand tel que I et J coïncident sur les littéraux strictement plus petits que L. $I >_{lex} J$ si $I(L) = 0$ et $J(L) = 1$.

Fixons un ensemble S' de $\mathcal{S}^*$. et considérons une interprétation partielle maximale $\mathcal{I}$ de $S' \cup S_0$, minimale pour l'ordre $>_{lex}$. On pose $P(t)$ le plus petit littéral non interprété par $\mathcal{I}$ et C_1 et C_2 les deux clauses de $S' \cup S_0$ falsifiées par les extensions de $\mathcal{I}$ à l'interprétation de $P(t)$: il existe deux substitutions σ_1 et σ_2 telles que $C_1\sigma_1 = P(t) \vee C_1'$ falsifie l'extension de $\mathcal{I}$ à $\mathcal{I}(P(t)) = 0$ et $C_2\sigma_2 = \neg P(t) \vee C_2'$ falsifie l'extension de $\mathcal{I}$ à $\mathcal{I}(P(t)) = 1$. Par factorisation, on peut supposer que les littéraux de C_1' et de C_2' sont tous strictement plus petits que $P(t)$. Par la règle de réduction, on peut supposer que la substitution σ_1 (respectivement σ_2) est sans réduction pour C_1 (respectivement C_2).

Nous allons montrons qu'il existe une clause de $S' \cup S_0$ falsifiée par des littéraux plus petits que $P(t)$, ce qui contredit le fait que $\mathcal{I}$ est une interprétation partielle de $S' \cup S_0$. Nous en déduisons alors que $\mathcal{I}$ est vide et donc $\perp \in S'$. Pour cela, nous distinguons quatre cas : $C_1, C_2 \in S_0$, ou $C_1 \in S_0$ et $C_2 \notin S_0$, ou $C_1 \notin S_0$ et $C_2 \in S_0$, ou enfin $C_1, C_2 \notin S_0$. Le deuxième et le troisième cas ne sont pas identiques car comme nous avons choisi l'interprétation « la plus à gauche », les rôles de littéraux $P(t)$ et $\neg P(t)$ ne sont pas symétriques.

Le cas le plus simple est lorsque les deux clauses ne sont pas dans S_0.

Lemme 8.4 (Cas $C_1, C_2 \notin S_0$) *Si $C_1, C_2 \notin S_0$, alors l'interprétation partielle $\mathcal{I}$ falsifie une instance close de $S' \cup S_0$.*

Preuve. On pose $C_1 = P(v') \vee C_1''$ et $C_2 = \neg P(v'') \vee C_2''$ avec $P(v')\sigma_1 = P(t)$ et $P(v'')\sigma_2 = P(t)$. Le littéral $P(t)$ est maximal dans C_1' pour l'ordre $\leqslant$. Comme $\leqslant$ est un ordre étroitement relevable (proposition 8.7) et que σ_1 est sans réduction, on déduit que le littéral $P(v')$ est maximal dans C_1. De même, le littéral $P(v'')$ est maximal dans C_2. Soit $\theta \in mgu(v', v'')$ tel que t est une instance de $v'\theta$. D'après la règle de réduction, la clause $C_1''\theta \vee C_2''\theta$ est dans S'. La clause $C_1' \vee C_2'$ en est une instance close, falsifiée par $\mathcal{I}$. □

Lemme 8.5 (Cas $C_1, C_2 \in S_0$) *Si $C_1, C_2 \in S_0$, alors l'interprétation partielle $\mathcal{I}$ falsifie une instance close de $S' \cup S_0$.*

Preuve. $C_1\sigma_1 = P(t) \vee \neg P(t_1) \vee \neg P(t_2)$ avec $t_1, t_2 \tilde{<} t$, $C_2 = \neg P(t) \vee P(t_3) \vee \neg P(t_4)$ avec $t_3, t_4 \tilde{<} t$. Les clauses $\neg P(t_1) \vee \neg P(t_2)$ et $P(t_3) \vee \neg P(t_4)$ sont falsifiées par $\mathcal{I}$.

Nous allons montrer un résultat intermédiaire sur l'ordre $\tilde{<}$.

Notation : Si $t = t_1 \oplus \cdots \oplus t_k$ tel que les termes t_i n'ont pas $\oplus$ pour symbole de tête, alors on note $\widetilde{t} \stackrel{\text{def}}{=} \{t_1, \ldots, t_k\}$. Si $t = f(t_1, \ldots, t_n)$, alors $\widetilde{t} \stackrel{\text{def}}{=} \{t\}$.

Proposition 8.16 *Soient t, t_1, t_2 trois termes clos en forme normale tels que $t = (t_1 \oplus t_2) \downarrow$ et $t_1, t_2 \tilde{<} t$. Considérons u le terme maximal de l'ensemble $\widetilde{t}$. Le terme u vérifie : $u \notin \widetilde{t_1}$ ou $u \notin \widetilde{t_2}$.*

En effet, supposons $u \in \widetilde{t_1}$ et $u \in \widetilde{t_2}$, alors $u \notin \widetilde{t_1 \oplus t_2} = \widetilde{t}$, contradiction.

Revenons à la preuve du lemme. On considère u le terme maximal de $\widetilde{t}$. D'après la proposition 8.16, $u \notin \widetilde{t_1}$ ou $u \notin \widetilde{t_2}$ et $u \notin \widetilde{t_3}$ ou $u \notin \widetilde{t_4}$. Comme les termes t_1 et t_2 jouent des rôles symétriques, nous pouvons supposer sans perte de généralité que $u \notin \widetilde{t_2}$.

- Supposons $u \notin \widetilde{t_3}$. Alors le terme $t_2 \oplus t_3$ vérifie $t_2 \oplus t_3 \tilde{<} u \tilde{<} t$, donc le littéral $P(t_2 \oplus t_3)$ est interprété par $\mathcal{I}$.
  - Soit $P(t_2 \oplus t_3) \in \mathcal{I}$, dans ce cas, la clause $C = P(t_3) \vee \neg P(t_2) \vee \neg P(t_2 \oplus t_3)$ est une instance d'une clause de S_0, falsifiée par $\mathcal{I}$.
  - Soit $\neg P(t_2 \oplus t_3) \in \mathcal{I}$. D'après les égalités $t_2 \oplus t_3 = t \oplus t_1 \oplus t \oplus t_4 = t_1 \oplus t_4$, la clause $C = \neg P(t_1) \vee \neg P(t_4) \vee P(t_2 \oplus t_3)$ (instance d'une clause de S_0) est falsifiée par $\mathcal{I}$.
- Supposons $u \notin \widetilde{t_4}$. Alors le terme $t_2 \oplus t_4$ vérifie $t_2 \oplus t_4 \tilde{<} u \tilde{<} t$, donc la clause $I(t_2 \oplus t_4)$ est interprétée dans $\mathcal{I}$. On conclut de la même manière qu'une des clause de S_0 est falsifiée par $\mathcal{I}$.
  - Soit $\neg P(t_2 \oplus t_4) \in \mathcal{I}$, dans ce cas, la clause $C = \neg P(t_2) \vee \neg P(t_4) \vee P(t_2 \oplus t_4)$ est une instance d'une clause de C_0, falsifiée par $\mathcal{I}$.
  - Soit $P(t_2 \oplus t_4) \in \mathcal{I}$. D'après les égalités $t_2 \oplus t_4 = t \oplus t_1 \oplus t \oplus t_3 = t_1 \oplus t_3$, la clause $C = \neg P(t_1) \vee P(t_3) \vee \neg P(t_2 \oplus t_4)$ (instance d'une clause de S_0) est falsifiée par $\mathcal{I}$. □

Les cas les plus difficiles sont lorsqu'une seule des deux clauses est dans S_0.

Lemme 8.6 (Cas $C_1 \in S_0$ et $C_2 \notin S_0$) *Si $C_1 \in S_0$ et $C_2 \notin S_0$, alors l'interprétation partielle $\mathcal{I}$ falsifie une instance close de $S' \cup S_0$.*

Preuve. $C_1 = P(x \oplus y) \vee \neg P(x) \vee \neg P(y)$. On pose $x\sigma_1 = t_1$, $y\sigma_1 = t_2$. On vérifie alors $(x\sigma_1 \oplus y\sigma_1) \downarrow = t$. Donc il existe des termes t'_1, t'_2, t' tels que $t = t'_1 \oplus t'_2$, $t_1 = t'_1 \oplus t'$ et $t_2 = t'_2 \oplus t'$ sans réduction ou $t = t_1 \oplus t_2$ sans réduction. Nous considérons uniquement le premier cas car le deuxième est similaire. Par hypothèse, $v_1 \tilde{<} v, v_2 \tilde{<} v$ donc $\mathcal{I}(P(t_1)) = \mathcal{I}(P(t_2)) = 1$. On peut supposer sans perte de généralité que $P(t_1) \tilde{<} P(t_2)$. Par minimalité de l'interprétation $\mathcal{I}$ pour l'ordre $>_{leq}$, l'interprétation $\mathcal{J}$ qui coïncide avec $\mathcal{I}$ sur les littéraux strictement plus petits que $P(t_1)$ et telle que $\mathcal{J}(P(v_1)) = 0$ falsifie une clause $C_3 = P(u) \vee C'$ de $S' \cup S_0$ (cette propriété est représentée à la figure 8.7). Nous considérons deux cas, suivant que C_3 est une clause de S_0 ou non.

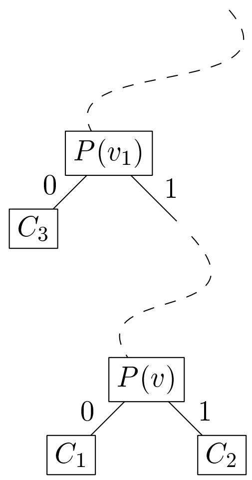


FIGURE 8.7 - *Interprétation partielle $\mathcal{I}$ la plus à gauche.*

Supposons $C_3 \notin S_0$ et que les règles de factorisation et réduction ne peuvent pas s'appliquer. Il existe une substitution σ_3 telle que $t_1 = u\sigma_3$. De plus, le littéral $P(t_1)$ est maximal dans $C_3\sigma_3$ par choix de l'interprétation partielle $\mathcal{J}$. Nous allons montrer que nous pouvons appliquer la règle **Extension 1** (éventuellement après avoir appliquée la règle **Explosion**) à C_2 et C_3 de manière à obtenir une clause falsifiée par $\mathcal{I}$. Posons $C_2 = \neg P_1(v) \vee C$. On obtient alors $t = v\sigma_2 = t'_1 \oplus t'_2$ et σ_2 est sans réduction donc $v = v_1 \oplus v_2$ avec $v_1\sigma_2 = t'_1$ et $v_2\sigma_2 = t'_2$. De la même manière, $u = u_1 \oplus u_2$ avec $u_1\sigma_3 = t'_1$ et $u_2\sigma_3 = t'$. On en déduit que t_1 et u_1 sont unifiables. En appliquant éventuellement la règle **Explosion**, nous pouvons supposer que $V(t) = V(t_1)$. Il existe une substitution $\theta \in mgu(v_1, u_1)$ telle que $\sigma_2 \uplus \sigma_3 = \theta\theta'$ pour une certaine substitution θ'. Considérons w le terme maximal de $\widetilde{t}$. Comme $t_2 \widetilde{<} t_1 \widetilde{<} t$, le terme w appartient à l'ensemble $\widetilde{t'_1}$ donc $t_2 = t'_2 \oplus t' \widetilde{<} t'_1$. Cette dernière inégalité permet de déduire que $v_2\sigma_2 \oplus u_2\sigma_3 \widetilde{<} v_1\sigma_2$, donc $(v_2 \oplus u_2)\theta\theta' \widetilde{<} v_1\sigma_2$. Il suit que $(v_2 \oplus u_2)\theta \not\succ v_1$. Comme θ est sans réduction pour les termes v, $u_1 \oplus u_2$, $v_2 \oplus u_2$ et les clauses C et C', nous pouvons appliquer la règle **Extension 1**. Nous en déduisons qu'il existe une clause $(C \vee C' \vee \neg P_3(v_2 \oplus u_2))\theta$ de S', falsifiée par $\mathcal{I}$.

Supposons maintenant que C_3 est une clause de S_0 : $C_3 = P(x' \oplus y') \vee \neg P(x') \vee \neg P(y')$. Posons $x'\sigma_3 = w_1$, $y'\sigma_3 = w_2$. On obtient alors $(x'\sigma_3 \oplus y'\sigma_3) \downarrow = t_1$. Par choix de l'interprétation $\mathcal{J}$, les termes w_1 et w_2 vérifient $w_1 \widetilde{<} t_1$ et $w_2 \widetilde{<} t_1$. Nous pouvons supposer sans perte de généralité que $w_2 \widetilde{<} w_1$. Nous savons de plus que $\mathcal{I}(P(w_1)) = 1$ et $\mathcal{I}(P(w_2)) = 1$. Considérons alors le terme $w_2 \oplus v_2$. Il vérifie $w_2 \oplus t_2 \widetilde{<} t$ car le terme maximal de $\widetilde{t}$ n'est ni dans $\widetilde{w_2}$ ni dans $\widetilde{t_2}$. On en déduit que le littéral $P(w_2 \oplus t_2)$ est interprété dans $\mathcal{I}$ et $\mathcal{I}(P(w_2 \oplus t_2)) = 1$ sinon la clause C_1 serait falsifiée par $\neg P(w_2) \vee \neg P(t_2) \vee P_0(w_2 \oplus t_2))$. Nous considérons alors la clause $C_1\sigma'_1 = \neg P(w_2 \oplus t_2) \vee \neg P(w_1) \vee P(t)$: nous sommes ramenés au cas précédent où les deux clauses C_1 et C_2 sont dans S_0. On conclut donc en appliquant le lemme 8.5. □

Il nous reste à considérer le dernier cas.

Lemme 8.7 (Cas $C_1 \notin S_0$ et $C_2 \in S_0$) *Si $C_1 \notin S_0$ et $C_2 \in S_0$, alors l'interprétation partielle $\mathcal{I}$ falsifie une instance close de $S' \cup S_0$.*

Preuve. $C_2\sigma_2 = \neg P(x)\sigma_2 \vee \neg P(y)\sigma_2 \vee P(x \oplus y)\sigma_2$ et $x\sigma_2 = v$. Nous pouvons supposer : $(x \oplus y)\sigma_2 \downarrow \widetilde{<} y\sigma_2$. En effet, si ce n'est pas le cas, on remplace alors C_2 par $C'_2 = \neg P(x') \vee \neg P(y') \vee P(x' \oplus y')$ et σ_2 par σ'_2 où la substitution σ'_2 est définie par : $x'\sigma'_2 = x\sigma_2 = v$, $y'\sigma'_2 = (x \oplus y)\sigma_2$. Nous obtenons alors $(x' \oplus y')\sigma'_2 = y\sigma_2$. Avec cette transformation, ce cas est similaire au précédent (lemme 8.6) en utilisant cette fois, la règle **Extension 2**. □

8.3.3.4 Complexité

Considérons un ensemble S de $\mathcal{C}^{\oplus}$. La borne obtenue pour montrer la terminaison de notre procédure (proposition 8.14) est une tour d'exponentielle de hauteur deux fois la profondeur maximale des termes des clauses de S. La complexité de notre algorithme de décision est donc non élémentaire.

Nous développons au paragraphe suivant un exemple de protocole dont nous montrons le secret en saturant par nos règles de déduction l'ensemble de clauses associé. L'ensemble saturé obtenu contient de l'ordre de 2^{40} clauses, ce qui n'est pas raisonnable pour un ordinateur. Cependant, il devrait être possible de raffiner la stratégie de résolution (en ajoutant des critères de redondance par exemple) de manière à éviter d'engendrer autant de clauses.

8.4 Application aux protocoles cryptographiques

L'objet de cette partie est de montrer quels protocoles peuvent être représentés par des ensembles de clauses de $\mathcal{C}$ ou $\mathcal{C}^{\oplus}$. Nous développons en particulier un exemple de protocole dont nous montrons qu'il préserve le secret. La modélisation des messages sous forme de termes est la même que celles présentées aux chapitres 3, 4 et 5, nous ne la rappellerons donc pas. De plus, les règles des protocoles sont modélisées sous forme de clauses de Horn comme au chapitre 3, nous ne nous attarderons donc pas sur ces aspects de la modélisation.

8.4.1 Expressivité

Comme annoncé en début de chapitre (partie 8.1.1), nos classes $\mathcal{C}$ et $\mathcal{C}^{\oplus}$ permettent de modéliser tous les protocoles ne permettant qu'une copie (ou test) par règle et pour lesquels les nonces sont représentés par des constantes. Comparées aux clauses de Horn contraintes utilisées au chapitre 3 pour décrire les protocoles, ces deux classes présentent deux principales restrictions :

- chaque clause ne contient qu'au plus une variable ou est de la forme $C[x \mapsto f(x_1, \ldots, x_n)]$ où C ne contient qu'une variable,
- les prédicats sont unaires : on ne tient plus compte de la trace dans la description des protocoles. La trace permettait en particulier de garantir la fraîcheur des nonces ce qui n'est plus possible au premier ordre.

La classe $\mathcal{C}^{\oplus}$ permet également de tenir compte des propriétés algébriques du xor. Remarquons que les clauses de Horn contraintes utilisées au chapitre 3 le permettait déjà en exprimant l'égalité modulo la théorie équationnelle du xor à l'aide d'un prédicat d'égalité $Egal_{\oplus}$ défini en clauses de Horn (voir figure 8.8).

Les classes $\mathcal{C}$ et $\mathcal{C}^{\oplus}$ ont été conçues en particulier pour pouvoir exprimer le pouvoir de l'intrus (voir figure 8.9). Les clauses représentant le pouvoir d'analyse et de synthèse de l'intrus sont exactement les mêmes que celles du chapitre 3 à ceci près que la deuxième composante

Nous reprenons les notations introduites au chapitre 3.

$$\begin{array}{rcll}
 & \Rightarrow & Egal_{\oplus}((x \oplus y) \oplus z, x \oplus (y \oplus z)) & \\
 & \Rightarrow & Egal_{\oplus}(x \oplus y, y \oplus x) & \\
 & \Rightarrow & Egal_{\oplus}(x \oplus 0, x) & \\
 & \Rightarrow & Egal_{\oplus}(x \oplus x, 0) & \\
 & \Rightarrow & Egal_{\oplus}(x, x) & \\
Egal_{\oplus}(x, y) & \Rightarrow & Egal_{\oplus}(y, x) & \\
Egal_{\oplus}(x_1, y_1), \ldots, Egal_{\oplus}(x_n, y_n) & \Rightarrow & Egal_{\oplus}(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n)) & f \in \mathcal{F}
\end{array}$$

FIGURE 8.8 - *Définition du prédicat d'égalité $Egal_{\oplus}$ modulo la théorie équationnelle du xor.*

correspondant à la trace est supprimée. La clause spéciale $C_0 = \neg I(x) \vee \neg I(y) \vee I(\oplus y)$ permet en particulier de représenter la capacité de l'intrus à calculer le xor de deux messages.

Remarque : Le prédicat $I_{\mathsf{inv}}(x)$ est utilisé pour signifier que l'intrus connaît l'inverse de la clef x. Nous distinguons ici le chiffrement asymétrique du chiffrement symétrique en utilisant le symbole $\{\_\}\_$ pour le chiffrement symétrique et le symbole $[\_]\_$ pour le chiffrement asymétrique. Comme nous l'avons vu au chapitre 2, une telle distinction correspond mieux à l'implémentation des protocoles. Ce choix est aussi le résultat d'une considération pratique car la clause $I(\{x\}_{\mathsf{pub}(y)}), I(\mathsf{prv}(y)) \Rightarrow I(x)$ utilisée pour exprimer le déchiffrement d'un message chiffré avec une clef asymétrique n'appartient pas à la classe $\mathcal{C}^{\oplus}$. Une conséquence de cette distinction est que les clefs symétriques utilisées dans des algorithmes de chiffrement asymétriques n'ont pas d'inverse.

Il reste à montrer comment traduire les règles d'un protocole en clauses de $\mathcal{C}$ ou $\mathcal{C}^{\oplus}$, ce que nous allons voir au travers d'un exemple au paragraphe suivant.

8.4.2 Exemple

Nous avons construit un exemple de protocole utilisant la fonction xor comme une primitive de chiffrement. Chiffrer un message en calculant son xor avec un autre message (qui peut alors être considéré comme une clef) présente l'avantage d'être une opération beaucoup plus rapide que le chiffrement à clefs symétriques avec des algorithmes classiques. Par contre, les protocoles utilisant le xor comme primitive de chiffrement sont plus facilement sujet à des attaques. Par exemple, L. Paulson a montré que le protocole de Bull [Pau97a] préserve le secret si on fait l'hypothèse du chiffrement parfait alors que ce n'est plus le cas si on tient compte des propriétés algébriques du xor.

Description du protocole. Le protocole que nous proposons a pour but de permettre l'échange d'un secret entre deux agents partageant une clef privée K_{ab}.

$$\begin{array}{lll}
A & \rightarrow B : & N_a \oplus K_{ab} \\
B & \rightarrow A : & N_b \oplus N_a \\
A & \rightarrow B : & S_{ab} \oplus N_b
\end{array}$$

$I(x), I(y)$	$\Rightarrow$	$I(< x, y >)$	L'intrus peut former la paire de deux messages connus.
$I(x), I(y)$	$\Rightarrow$	$I(\{x\}_y)$	L'intrus peut chiffrer un message connu avec une clef connue.
$I(x)$	$\Rightarrow$	$I(\mathsf{h}(x))$	L'intrus peut hacher un message.
$I(< x, y >)$	$\Rightarrow$	$I(x)$	L'intrus peut projeter sur la première ou la deuxième composante.
$I(< x, y >)$	$\Rightarrow$	$I(y)$	
$I(\{x\}_y), I(y)$	$\Rightarrow$	$I(x)$	L'intrus peut déchiffrer un message (chiffré avec un algorithme de chiffrement symétrique) s'il possède la clef.
$I([x]_y), I_{\mathsf{inv}}(y)$	$\Rightarrow$	$I(x)$	L'intrus peut déchiffrer un message (chiffré avec un algorithme de chiffrement asymétrique) s'il possède l'inverse de la clef.
$I(\mathsf{pub}(x))$	$\Rightarrow$	$I_{\mathsf{inv}}(\mathsf{prv}(x))$	Les clefs publiques et privées sont inverses l'une de l'autre.
$I(\mathsf{prv}(x))$	$\Rightarrow$	$I_{\mathsf{inv}}(\mathsf{pub}(x))$	

FIGURE 8.9 - *Le pouvoir de l'intrus exprimé en clauses de $\mathcal{C}^{\oplus}$.*

À la première étape, l'agent A envoie le xor de la clef K_{ab} avec un nonce N_a qu'il vient d'engendrer. Une telle opération ne compromet pas la clef K_{ab} car le nonce N_a est un nombre aléatoire, jamais utilisé auparavant. Une telle règle peut être représentée par la clause :

$$\Rightarrow I(n_1(a, b) \oplus K(a, b)),$$

où a et b sont des constantes représentant les agents A et B.

À la deuxième étape, l'agent B apprend le nonce N_a en calculant le xor du message reçu avec la clef K_{ab} partagée entre A et B. Il engendre alors un nonce N_b et envoie le xor de ce nonce avec le nonce N_a. À nouveau une telle règle ne semble pas compromettre N_a puisque N_b est un nombre aléatoire. Cette étape peut être représentée par la clause :

$$I(z) \Rightarrow I(z \oplus n_2(b, a) \oplus K(a, b)).$$

Enfin, l'agent A calcule le xor du message reçu avec son nonce N_a pour obtenir le nonce N_b de B. Il envoie alors le xor du secret à échanger S_{ab} avec le nonce de B.

Ce protocole est très simple et pourrait être implémenté de manière efficace car l'opération xor est peu coûteuse. Nous allons montrer que même si ce protocole utilise uniquement le xor comme primitive de chiffrement, il préserve le secret de S_{ab}. La preuve repose sur le fait que tout message $m_1 \oplus \cdots \oplus m_k$ connu de l'intrus contient toujours un nombre pair de composantes m_i (distinctes) inconnues de l'intrus.

Première règle :

$$\begin{array}{lll} \Rightarrow I(n_1(a,b) \oplus K(a,b)) & \Rightarrow I(n_1(b,a) \oplus K(a,b)) & \Rightarrow I(n_1(a,c) \oplus K(a,c)) \\ \Rightarrow I(n_1(c,a) \oplus K(a,c)) & \Rightarrow I(n_1(c,b) \oplus K(b,c)) & \Rightarrow I(n_1(b,c) \oplus K(b,c)) \end{array}$$

Deuxième règle :

$$\begin{array}{ll} I(z) \Rightarrow I(z \oplus n_2(a,b) \oplus K(a,b)) & I(z) \Rightarrow I(z \oplus n_2(b,a) \oplus K(a,b)) \\ I(z) \Rightarrow I(z \oplus n_2(a,c) \oplus K(a,c)) & I(z) \Rightarrow I(z \oplus n_2(c,a) \oplus K(a,c)) \\ I(z) \Rightarrow I(z \oplus n_2(b,c) \oplus K(b,c)) & I(z) \Rightarrow I(z \oplus n_2(c,b) \oplus K(b,c)) \end{array}$$

Troisième règle :

$$\begin{array}{ll} I(z) \Rightarrow I(z \oplus n_1(a,b) \oplus S(a,b)) & I(z) \Rightarrow I(z \oplus n_1(b,a) \oplus S(b,a)) \\ I(z) \Rightarrow I(z \oplus n_1(a,c) \oplus S(a,c)) & I(z) \Rightarrow I(z \oplus n_1(c,a) \oplus S(c,a)) \\ I(z) \Rightarrow I(z \oplus n_1(b,c) \oplus S(b,c)) & I(z) \Rightarrow I(z \oplus n_1(c,b) \oplus S(c,b)) \end{array}$$

FIGURE 8.10 - *Clauses correspondant aux règles de notre protocole pour trois agents a, b et c.*

Notre résultat de réduction du chapitre 6 assure qu'il suffit de vérifier ce protocole pour trois agents : deux honnêtes et un malhonnête. Nous supposons en effet que notre protocole ne permet pas à un agent de s'envoyer des messages à lui-même car nous trouvons cette hypothèse peu réaliste. Cependant notre protocole préserve en fait le secret que l'on suppose ou non qu'un agent puisse se parler à lui-même. Nous fixons ainsi trois constantes : a, b et c. Les constantes a et b représentent des agents honnêtes et c représente un agent malhonnête. Les clauses représentant l'ensemble des règles de notre protocole sont décrites à la figure 8.10. De plus, nous ajoutons au pouvoir de l'intrus les clauses représentant sa connaissance initiale : l'intrus connaît les clefs de l'agent malhonnête c.

$$\Rightarrow I(K(a,c)) \qquad\qquad \Rightarrow I(K(b,c)) \tag{8.1}$$

Remarque : Notre résultat de réduction à deux agents s'applique dans un cadre très large comme nous l'avons vu au chapitre 6. Une fois que les nonces sont représentés par un nombre fini de constantes, tous les protocoles ne permettant qu'une copie de message par règles peuvent être ainsi représentés par un ensemble de clauses de $\mathcal{C}$ ou $\mathcal{C}^{\oplus}$ (suivant que l'on veuille tenir compte de la théorie équationnelle du xor ou non). Comme nous l'avons expliqué au paragraphe 8.1.1, la plupart des protocoles satisfont cette hypothèse comme le protocole de Needham-Schroeder-Lowe et de nombreux protocoles de [CJ97].

Propriété de sécurité. La propriété de sécurité exprimant le secret de $S(a,b)$ se traduit tout simplement par la clause :

$$\phi_0 \stackrel{\text{def}}{=} \neg I(S(a,b)).$$

On pose $\mathcal{C}_P$ l'ensemble des clauses définies aux figures 8.9 et 8.10 ainsi qu'à l'équation 8.1.

Il existe une attaque sur notre protocole si et seulement si $\mathcal{C}_P \models \neg\phi_0$, c'est-à-dire si et seulement si l'ensemble de clauses $\mathcal{C}_P \cup \phi_0$ n'est pas satisfaisable. Nous sommes donc ramenés à un problème de satisfaction dont nous avons prouvé la décidabilité dans les parties précédentes. Nous pouvons donc appliquer notre procédure à l'ensemble $\mathcal{C}_P \cup \phi_0$.

Notons qu'on peut toujours se ramener à un problème de satisfaction d'un ensemble de clauses pour des propriétés de sécurité comme le secret et l'authentification. En effet, nous avons vu au chapitre 6 (partie 6.1.3) que de telles propriétés pouvaient toujours s'exprimer sous forme de clauses purement négatives, et pour toute clause purement négative : $\mathcal{C} \models \neg\phi$ si et seulement si l'ensemble de clauses $\mathcal{C} \cup \phi$ n'est pas satisfaisable.

Proposition 8.17 *Soient $\mathcal{C}$ un ensemble de clauses de Horn tel qu'aucune de ces clauses ne soit purement négative et ϕ une clause purement négative. Les propositions suivantes sont équivalentes :*

1. *$\mathcal{C} \models \neg\phi$ (c'est-à-dire il existe une attaque sur $\mathcal{C}$ pour la propriété ϕ),*
2. *l'ensemble $\mathcal{C} \cup \{\phi\}$ n'est pas satisfaisable.*

Preuve. On considère le plus petit modèle de Herbrand $\mathcal{H}$ de $\mathcal{C}$.

Supposons que l'ensemble de clauses $\mathcal{C} \cup \{\phi\}$ ne soit pas satisfaisable. Alors $\mathcal{H}$ n'est pas un modèle de $\mathcal{C} \cup \{\phi\}$. Comme $\mathcal{H}$ est un modèle de $\mathcal{C}$, c'est que $\mathcal{H}$ est un modèle de $\neg\phi$. Comme $\mathcal{H}$ est le plus petit modèle de Herbrand de $\mathcal{C}$, on déduit : $\mathcal{C} \models \neg\phi$.

Inversement, supposons que l'ensemble de clauses $\mathcal{C} \cup \{\phi\}$ est satisfaisable. Comme l'ensemble $\mathcal{C} \cup \{\phi\}$ est un ensemble de clauses de Horn, il existe un plus petit modèle de Herbrand $\mathcal{H}'$ de cet ensemble de clauses : $\mathcal{H}' \models \mathcal{C} \cup \{\phi\}$. Comme $\mathcal{H}'$ est *a fortiori* un modèle de $\mathcal{C}$, la relation $\mathcal{H} \subseteq \mathcal{H}'$ est vérifiée. Maintenant, comme ϕ est une clause purement négative, si $\mathcal{H}' \models \phi$, alors *a fortiori* : $\mathcal{H} \models \phi$. On peut donc conclure que : $\mathcal{C} \not\models \neg\phi$. □

Preuve du secret. Nous allons maintenant montrer que notre protocole préserve le secret de S_{ab} en prouvant que l'ensemble $\mathcal{C}_P \cup \phi_0$ est satisfaisable.

Proposition 8.18 *L'ensemble $\mathcal{C}_P \cup \phi_0$ est satisfaisable.*

Preuve. On partage l'ensemble des termes représentant les nonces et les clefs en deux ensembles. L'ensemble des données supposées secrètes est défini par :

$$\Gamma_1 = \{n_1(a,b), n_1(b,a), n_2(a,b), n_2(b,a), S(a,b), S(b,a), K(a,b)\},$$

et l'ensemble des données connues de l'intrus défini par :

$$\Gamma_2 = \bigcup_{i=1,2} \{n_i(a,c), n_i(c,a), n_i(b,c), n_i(c,b)\} \cup \{S(a,c), S(c,a), S(b,c), S(c,b), K(a,c), K(b,c)\}.$$

On considère alors l'ensemble des termes T (respectivement T') où les données supposées secrètes sont « xorées » un nombre pair de fois (respectivement un nombre impair de fois) :

$$\begin{aligned} T &\stackrel{\text{def}}{=} \{u_1 \oplus \cdots \oplus u_n \oplus t_1 \oplus \cdots \oplus t_k \mid n \text{ est pair}, u_i \in \Gamma_1, t_j \in \Gamma_2, u_i, t_j \text{ distincts}\}, \\ T' &\stackrel{\text{def}}{=} \{u_1 \oplus \cdots \oplus u_n \oplus t_1 \oplus \cdots \oplus t_k \mid n \text{ est impair}, u_i \in \Gamma_1, t_j \in \Gamma_2, u_i, t_j \text{ distincts}\}. \end{aligned}$$

En appliquant les règles de résolution ordonnée à l'ensemble $\mathcal{C}_P \cup \phi_0$, on obtient l'ensemble de clauses :

$$\begin{aligned} S^* &\stackrel{\text{def}}{=} \{I(m) \mid m \in T\} \cup \{\neg I(z \oplus m_1) \vee I(z \oplus m_2) \mid m_1 \oplus m_2 \in T\} \\ &\quad \cup \{\neg I(m_1) \vee I(m_2) \mid m_1 \oplus m_2 \in T\} \cup \{\neg I(m) \mid m \in T'\}. \end{aligned}$$

Il faut (et il suffit) donc de montrer que S^* est satisfaisable. En fait, nous allons montrer que S^* est déjà saturé pour l'ensemble de nos règles de déduction. Comme $\perp \notin S^*$, cela permet de conclure que l'ensemble $\mathcal{C}_P \cup \phi_0$ est satisfaisable. On suppose de plus qu'on applique le critère de redondance suivant : on n'ajoute pas les clauses de la forme $C_1 \vee C_2$ lorsque l'une des deux clauses C_1 ou C_2 est déjà dans l'ensemble. Considérons donc toutes les règles de déduction possibles.

- La règle de **factorisation** ne peut pas s'appliquer aux clauses de S^*.
- Par construction de S^*, la règle de **réduction** préserve l'appartenance à S^*.
- La règle d'**explosion** ne modifie pas la somme $m_1 \oplus m_2$ donc préserve l'appartenance à S^*.
- Considérons une résolution binaire ordonnée entre deux clauses avec variables C_1 et C_2 de S^*. $C_1 = \neg I(z \oplus m_1) \vee I(z \oplus m_2)$ et $C_2 = I(z' \oplus m_3) \vee \neg I(z' \oplus m_4)$, avec $m_1 \oplus m_2, m_3 \oplus m_4 \in T$. Considérons un plus général unificateur $\theta \in mgu(z \oplus m_1, z' \oplus m_3)$. Il vérifie : $z\theta = z'' \oplus m_5$, $z'\theta = z'' \oplus m_6$ et $(m_5 \oplus m_6) \downarrow = (m_1 \oplus m_3) \downarrow$. La clause résultante est : $C = \neg I(z'' \oplus m_6 \oplus m_4) \vee I(z'' \oplus m_5 \oplus m_2)$. Comme le terme $(m_6 \oplus m_4 \oplus m_5 \oplus m_2) \downarrow = (m_1 \oplus m_3 \oplus m_2 \oplus m_4) \downarrow$ est dans T, il vient que la clause C est dans S^*. Lorsque l'une des deux clauses est close, le cas est similaire.
- Considérons maintenant la règle **extension 1** (la règle **extension 2**) est similaire. Soient $C_1 = \neg I(z \oplus m_1) \vee I(z \oplus m_2)$ et $C_2 = I(z' \oplus m_3) \vee \neg I(z' \oplus m_4)$ tels que $m_1 \oplus m_2), m_3 \oplus m_4 \in T$. En reprenant les notations utilisées pour décrire la règle **extension 1**, les termes t et t_1 sont de la forme $t = z \oplus m_1 = z \oplus m_1' \oplus m_1''$ et $t_1 = z \oplus m_1'$ car $V(t) = V(t_1)$. Pour les termes u_1 et u_2, deux cas sont possibles : $u_1 = z' \oplus m_3'$ et $u_2 = m_3''$ ou $u_2 = z' \oplus m_3'$ et $u_1 = m_3''$, avec dans les deux cas : $m_3' \oplus m_3'' = m_3$. Nous ne considérons que le premier cas car le deuxième est similaire. Soit θ un plus général unificateur de t et u_1 : $\theta \in mgu(z \oplus m_1', z' \oplus m_3')$. Il vérifie : $z\theta = z'' \oplus m_5$, $z'\theta = z'' \oplus m_6$, et $(m_5 \oplus m_6) \downarrow = (m_1' \oplus m_3') \downarrow$. La clause résultante est :

  $$C = I(z'' \oplus m_5 \oplus m_2) \vee \neg I(z'' \oplus m_6 \oplus m_4) \vee \neg I(m_1'' \oplus m_3'').$$

  Deux cas sont possibles.
  - Ou bien $m_1'' \oplus m_3'' \in T$, auquel cas le terme $(m_1' \oplus m_3' \oplus m_2 \oplus m_4) \downarrow$ est dans T car $(m_1 \oplus m_2 \oplus m_3 \oplus m_4) \downarrow \in T$. Dans ce cas, $m_5 \oplus m_2 \oplus m_6 \oplus m_4 \in T$ car $m_5 \oplus m_6 = m_1' \oplus m_3'$. On obtient alors que la clause $I(x \oplus m_5 \oplus m_2) \vee I(x \oplus m_6 \oplus m_4)$ est dans S^* et C est redondante avec cette clause, on ne l'ajoute donc pas à S^*.
  - Ou bien $m_1'' \oplus m_3'' \notin T$, auquel cas $m_1'' \oplus m_3'' \in T'$ et alors par construction de S^*, la clause $\neg I(m_1'' \oplus m_3'')$ appartient à S^* et C est redondante avec cette clause.

  Si l'une des deux clauses est close, le raisonnement est similaire.

Comme l'abstraction qui consiste à remplacer les nonces par des constantes est correcte, on en déduit que notre protocole préserve le secret. □

8.5 Lien avec d'autres travaux

Comparons les résultats de décidabilité pour les classes $\mathcal{C}$ et $\mathcal{C}^{\oplus}$ aux autres travaux existants.

8.5.1 Contraintes ensemblistes avec tests d'égalité

Le résultat de décidabilité de la classe $\mathcal{C}$ peut être vu comme une conséquence de la décidabilité de la classe de contraintes ensemblistes avec tests d'égalité introduites dans [CLC03c]. Nous nous reportons à [CP97, CLC03c] pour les définitions de *contraintes ensemblistes* et *contraintes ensemblistes définies*. Dans l'article [CLC03c], nous partageons l'ensemble des variables d'ensemble en deux catégories : les variables *basiques* et les variables non basiques. Les tests d'égalité sur les variables basiques peuvent être quelconques, tandis que les tests d'égalité sur les variables non basiques ne doivent pas se « chevaucher ». Précisons un peu ces définitions. Une *contrainte d'égalité* c est une relation d'équivalence sur un ensemble fini de positions, noté $P(c)$. Les contraintes ensemblistes peuvent alors contenir des expressions de la forme $f^c(e_1, \ldots, e_n)$ où c est une contrainte d'égalité.

L'ensemble des *chemins* d'une expression e est défini inductivement de la manière suivante :

$$\begin{aligned}
\Pi(f^c(e_1, \ldots, e_n)) &\stackrel{\text{def}}{=} \{\epsilon\} \cup 1 \cdot \Pi(e_1) \cup \ldots \cup n \cdot \Pi(e_n), \\
\Pi(e_1 \cap e_2) &\stackrel{\text{def}}{=} \Pi(e_1) \cup \Pi(e_2), \\
\Pi(X) &\stackrel{\text{def}}{=} \{\epsilon\}.
\end{aligned}$$

Soit p un chemin de $\Pi(e)$. On définit $e|_p$ l'ensemble des sous-expressions de e à la position p par :

$$\begin{aligned}
e|_\epsilon &\stackrel{\text{def}}{=} \{e\}, \\
(e_1 \cap e_2)|_{i \cdot p} &\stackrel{\text{def}}{=} e_1|_{i \cdot p} \cup e_2|_{i \cdot p}, \\
(f^c(e_1, \ldots, e_n))|_{i \cdot p} &\stackrel{\text{def}}{=} e_i|_p, \\
X|_{i \cdot p} &\stackrel{\text{def}}{=} \emptyset.
\end{aligned}$$

Exemple 8.11 *Considérons l'ensemble formé des deux contraintes ensemblistes :*

$$f^{21=12}(f(Z, Y) \cap X \quad \textit{et} \quad g(X) \cap Y) \subseteq f(Y, g(X)).$$

Il a pour solution σ telle que : $\sigma = \{X \mapsto \{a, b, f(a, b)\}; Y \mapsto \{b, g(a), g(b), f(a, b)\}; Z \mapsto \{a, b\}\}$. La substitution σ est bien solution car : $[\![f^{12=21}(f(Z, Y) \cap X, g(X) \cap Y)]\!]_\sigma = \{f(f(a, b), g(b))\}$

La satisfaction d'un ensemble de contraintes ensemblistes avec tests d'égalité est indécidable en général.

Définissons maintenant la notion de variable basique.

Définition 8.18 (variable basique) *Soit S un ensemble de contraintes ensemblistes avec tests d'égalité. Un symbole de fonction g est dit* non inversible *:*

- *s'il n'apparaît pas à droite des contraintes d'inclusion de S,*
- *et si pour toute expression $f^c(e_1, \ldots, e_n)$ de S et tout préfixe strict π' d'un chemin π de $P(c)$, $e|_{\pi'}$ ne contient pas d'expressions ayant g pour symbole de tête.*

On note $\mathcal{OF}(S)$ l'ensemble des symboles non inversibles de S.

Si X est une variable de S, on note $\mathcal{R}(X)$ l'ensemble des contraintes ensemblistes telles que X apparaisse à droite de l'inclusion.

L'ensemble des variables basiques *de S est le plus grand ensemble de variables de S tel que :*

- *si X est basique alors $\mathcal{R}(X)$ ne contient que des symboles non inversibles et des variables basiques,*
- *si X est basique alors :*
  - *ou bien $\mathcal{R}(X)$ ne contient qu'une seule clause $\phi \Rightarrow e \subseteq X$ telle que X n'apparaît pas dans e,*
  - *ou bien les symboles de fonctions de $\mathcal{R}(X)$ ne peuvent apparaître que dans des $\mathcal{R}(Y)$ ou Y est une variable basique.*

Intuitivement, un symbole de fonction est non inversible si on ne peut ni projeter ni tester « en dessous » de ce symbole de fonction. De plus, les symboles de fonction utilisés récursivement pour construire les variables basiques ne peuvent pas être utilisés pour des variables non basiques.

Exemple 8.12 *L'ensemble de contraintes considéré ci-dessous ne contient que des variables basiques. On voit dans cet exemple que les variables basiques peuvent représenter des agents (honnêtes ou malhonnêtes), des nonces, des clefs ou des messages.*

$$\begin{array}{rclrcl}
0 & \subseteq & \mathsf{Nat} & succ(\mathsf{Nat}) & \subseteq & \mathsf{Nat} \\
da(\mathsf{Nat}) & \subseteq & \mathsf{DA} & ha(\mathsf{Nat}) & \subseteq & \mathsf{HA} \\
\mathsf{DA} & \subseteq & \mathsf{A} & \mathsf{HA} & \subseteq & \mathsf{A} \\
K(\mathsf{A},\mathsf{A}) & \subseteq & \mathsf{Key} & \mathsf{shrA} & \subseteq & \mathsf{M} \\
\mathsf{A} & \subseteq & \mathsf{M} & \mathsf{Key} & \subseteq & \mathsf{M} \\
<\mathsf{M},\mathsf{M}> & \subseteq & \mathsf{M} & \{\mathsf{M}\}_{\mathsf{M}} & \subseteq & \mathsf{M}
\end{array}$$

Les symboles Nat, A, DA, HA, M *et* Key *représentent des variables d'ensemble.*

Une expression e est basique si e est une variable, une intersection d'expressions basiques ou si $e = f(e_1, \ldots, e_n)$ (ou $e = f^c(e_1, \ldots, e_n)$) avec $e_1, \ldots, e_n$ expressions basiques.

Les contraintes ensemblistes que nous allons considérer doivent vérifier une *condition de basicité* : les tests sur les variables basiques sont restreints à des tests qui ne doivent pas se chevaucher.

Définition 8.19 (condition de basicité) *Une contrainte d'égalité c d'une expression $f^c(e_1, \ldots, e_n)$ satisfait la* condition de basicité *si :*

$$\left.\begin{array}{r}
p \cdot i \cdot q \sim_c p' \\
i \neq j \\
p' \not\geqslant_{pref} p \\
\forall w,\ p \cdot j \cdot w \not\sim_c p \cdot i \cdot q
\end{array}\right\} \Rightarrow
\begin{array}{l}
\textit{il existe des positions } p_1, p_2 \textit{ telles que} \\
p_1 \sim_c p', p_2 \sim_c p \cdot j \textit{ et} \\
\textit{ou } e|_{p_1} \textit{ or } e|_{p_2} \\
\textit{ne contiennent que des expressions basiques,}
\end{array}$$

où $\geqslant_{pref}$ est l'ordre préfixe sur les positions. Une expression e satisfait la condition de basicité *si pour chacune de ses sous-expressions $f^c(e_1, ..., e_n)$, la contrainte d'égalité c satisfait la condition de basicité.*

Un exemple est présenté à la figure 8.11 où au moins l'une des trois positions terminales ne doit contenir que des expressions basiques.

Remarque : si une contrainte ensembliste ne contient qu'une seule variable non basique, alors elle satisfait automatiquement la condition de basicité.

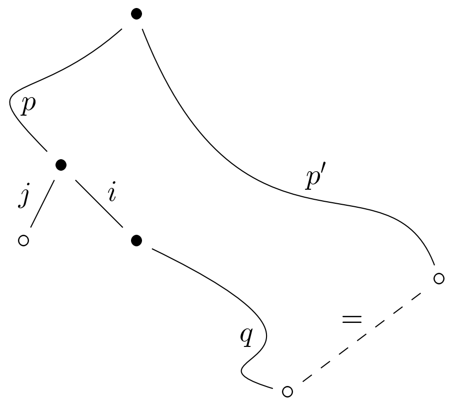


FIGURE 8.11 - *La condition de basicité.*

Nous pouvons maintenant définir notre classe de contraintes ensemblistes avec tests d'égalité, notées ET-contraintes.

Définition 8.20 (ET-contrainte) *Une* ET-contrainte *est un ensemble fini de conjonction de clauses*

$$e \subseteq e',$$

où e et e' sont des expressions construites à partir de variables d'ensemble, du symbole constant $\perp$, d'intersections et d'application de symboles de fonction $f^c(\ldots)$ ou $f(\ldots)$. Une telle clause doit de plus vérifier que :
- *e' n'a pas de sous-expressions de la forme $f^c(\ldots)$ à moins que c ne soit vide,*
- *toute sous-expression $f^c(e_1, \ldots, e_n)$ de e vérifie $P(c) \subseteq \Pi(f^c(e_1, \ldots, e_n))$,*
- *e vérifie la condition de basicité.*

Théorème 8.3 *Soit S une ET-contrainte, la satisfaction de S est décidable.*

La preuve de ce théorème s'obtient par des techniques de saturation de l'ensemble des contraintes et en introduisant une nouvelle classe d'automates à mémoire. Le contenu de la mémoire est un terme dont les sous-termes peuvent être comparés et qui peuvent être projetés. On montre alors que la décision du vide d'un tel automate à mémoire est décidable en DEXPTIME.

Notre résultat de décidabilité sur les ET-contraintes permet de déduire un résultat de décidabilité pour les protocoles cryptographiques. En effet, les deux premières règles du protocole de Needham-Schroeder-Lowe à clefs publiques peuvent par exemple être représentées par les deux contraintes définies à la figure 8.12

Ces contraintes vérifient la condition de basicité. On montre en fait que le secret est décidable pour tous les protocoles comme celui de Needham-Schroeder-Lowe, c'est-à-dire

La variable A est basique, les variables I et M ne le sont pas.
La première règle $A \to B : \{A, N_a\}_{\mathsf{pub}(B)}$ peut être représentée par la clause :

$$\{^{c_1} < A, N_1(A,B) >\}_{\mathsf{pub}A} \subseteq I,$$

où $c_1 = \{11 = 121, 122 = 21\}$.
La deuxième règle $B \to A : \{N_a, N_b, B\}_{\mathsf{pub}(A)}$ peut être représentée par la clause :

$$<^{c_2} \{< A, M >\}_{\mathsf{pub}A} \cup I, \{<< M, N_2(A,A) >, A >\}_{\mathsf{pub}(A)} > \subseteq I,$$

où $c_2 = \{111 = 21121 = 221, 121 = 21122 = 212, 112 = 2111\}$.

FIGURE 8.12 - *Les deux premières règles du protocole de Needham-Schroeder-Lowe traduites en contraintes ensemblistes.*

tels que l'ensemble des contraintes associé ne soit composé que de ET-contraintes. Aussi, le résultat de décidabilité que nous donnons pour notre classe $\mathcal{C}$ (théorème 8.1) est moins général que notre résultat de décidabilité sur les ET-contraintes sur deux aspects :
- notre classe $\mathcal{C}$ ne permet pas d'introduire des variables basiques,
- les tests d'égalité s'effectuent sur une unique variable alors que la condition de basicité est plus générale puisqu'elle permet (sous conditions) de tester des variables distinctes et qu'un nombre arbitraire de variables non testées est autorisé.

Cependant, l'intérêt de notre résultat de décidabilité pour $\mathcal{C}$ est que la preuve à l'aide des techniques classiques de résolution est nettement plus simple et permet plus facilement des extensions comme nous l'avons fait pour la théorie équationnelle du xor.

8.5.2 Autres résultats de décidabilité pour un nombre non borné de sessions

Sans le xor. Pour un nombre borné de sessions, le secret des protocoles cryptographique est un problème co-NP-complet [RT01]. Nous avons évoqué en introduction l'existence de trois autres résultats de décidabilité pour un nombre non borné de sessions.

J. Mitchell *et al.* [DLMS99] ont montré que le secret est décidable pour les protocoles sans nonce tels que la profondeur des messages est bornée.

G. Lowe [Low98] a présenté un résultat de décision pour des protocoles cryptographiques avec des nonces. Nous en avons décrit les principales restrictions au paragraphe 6.2.3 du chapitre 6. En particulier, les règles des protocoles doivent être entièrement typées, par des types atomiques et disjoints et toute donnée dont on ne demande pas le secret doit être immédiatement connue de l'intrus. De plus, il n'y a pas de clefs composées.

Avec le xor. Dans le cadre de la théorie équationnelle du xor, H. Comon-Lundh et V. Shmatikov [CLS03] se sont également intéressés à la décidabilité du secret des protocoles, mais pour un nombre de sessions borné. Ils montrent tout d'abord que décider si un terme t peut être déduit par l'intrus à partir d'un ensemble de termes clos est un problème NP-complet. Ils montrent ensuite que le secret est décidable pour un nombre borné de sessions sans faire aucune restriction sur le nombre de variables testées ou copiées. Les techniques utilisées (accessibilité pour un système de contraintes avec variables rigides) sont très différentes de celles utilisées ici.

Tableau récapitulatif Tous ces résultats sont résumés dans le tableau ci-dessous.

<table>
<tr><th rowspan="2">Nombre de sessions</th><th rowspan="2">Borné</th><th colspan="3">Non borné</th></tr>
<tr><th colspan="2">Sans nonce</th><th>Avec nonces</th></tr>
<tr><td rowspan="3">Hypothèse du chiffrement parfait</td><td rowspan="3">NP-complet
[RT01]</td><td colspan="2">Profondeur bornée :
DEXPTIME-complet
[DLMS99]</td><td>Profondeur bornée :
Indécidable
[DLMS99]</td></tr>
<tr><td>Une copie</td><td>Deux copies ou plus</td><td rowspan="2">Indécidable</td></tr>
<tr><td>3-EXPTIME
[Thèse]</td><td>Indécidable
[CLC03c]</td></tr>
<tr><td>Ajout du « ou » exclusif</td><td>NEXPTIME
[CLS03]</td><td>Décidable
[Thèse]</td><td>Indécidable</td><td>Indécidable</td></tr>
</table>

8.6 Perspectives

De nombreuses extensions de notre résultat de décision concernant les classes $\mathcal{C}$ et $\mathcal{C}^{\oplus}$ sont possibles.

- Tout d'abord, nous pourrions introduire comme dans [CLC03c] une notion de variable basique pouvant être ajoutée de manière arbitraire dans les clauses. Il semble en effet possible d'adapter la technique utilisée dans [CLC03c], à savoir calculer l'automate minimal associé aux solutions pour les variables basiques, choisir un ensemble de représentants correspondant aux états de l'automate et remplacer chacune des variables basiques par son ensemble de représentant. On obtient alors des clauses sans variables basiques qu'on peut résoudre avec les techniques précédentes. Il faut bien sûr vérifier qu'une telle transformation est correcte et complète.
  Une extension à des variables basiques permettrait de mettre en place des abstraction plus fines pour les nonces. Ainsi, au lieu de représenter les nonces par des constantes, nous pourrions comme B. Blanchet [Bla01] les représenter pour une fonction dépendant des messages reçus.
- D'autre part, les clauses « spéciales » de S_0 sont de la forme $\neg P(x) \vee \neg P(y) \vee P(x \oplus y)$. Il n'y a pas de raison théorique de considérer des prédicats tous identiques. Nous pensons donc que la classe $\mathcal{C}^{\oplus}$ étendue à des clauses de la forme $\neg P_1(x) \vee \neg P_2(y) \vee P_3(x \oplus y)$ reste décidable. En gardant le même système de déduction, la terminaison de la procédure est préservée. Pour rétablir la preuve de complétude, nous pensons qu'il est possible de mettre en œuvre un lemme de « pompage » sur les feuilles de l'arbre sémantique qui sont étiquetées par des clauses de S_0.
- Enfin, plus généralement, il serait intéressant d'explorer des extensions de $\mathcal{C}$ pour d'autres théories équationnelles comme celles des groupes abéliens ou de l'exponentielle modulaire.

Avant d'étendre et de raffiner nos résultats, il est peut-être possible de simplifier la preuve de décidabilité de la classe $\mathcal{C}^{\oplus}$ en changeant l'ordre sur les littéraux. Au lieu de définir $A \leqslant B$ par

$|A| \leqslant |B|$ et $|A|_x \leqslant |B|_x$, nous pensons définir $A \leqslant B$ par $(|A|_x, |A|) \leqslant_{leq} (|B|_x, |B|)$ où $\leqslant_{leq}$ désigne l'ordre lexicographique sur les couples d'entiers. L'ordre induit sur les littéraux clos reste le même. Ce nouvel ordre n'est plus étroitement relevable. Par contre, il est relevable pour les « grandes » substitutions, c'est-à-dire pour les substitutions θ telles que $|x\theta\downarrow| > |B| - |B|_x$. Le cas des « petites » substitutions est traité par la règle d'explosion :

$$\frac{P(t \oplus u) \vee C}{(P(t \oplus u) \vee C)\sigma \downarrow} \text{ si } t \text{ est clos, } u\sigma \text{ est clos et } u\sigma < t.$$

Ce changement d'ordre paraît simplifier la preuve car il n'y a plus lieu de se restreindre à des substitutions sans réduction. La règle de réduction devient en particulier inutile.

Notre résultat de décidabilité (notamment pour la classe $\mathcal{C}^{\oplus}$) n'est cependant pas praticable en raison de sa complexité. Nous avons ainsi vu au paragraphe 8.4, que l'ensemble saturé correspondant à notre exemple de protocole comprenait de l'ordre de 2^{40} clauses. Cependant, il est possible de raffiner la stratégie de résolution en adoptant n'importe quel ordre compatible sur les littéraux clos avec l'ordre $\widetilde{<}$ défini au paragraphe 8.3.3.3.

CHAPITRE 9

Conclusion et perspectives

La vérification des protocoles cryptographiques se heurte à trois principales difficultés : la modélisation des protocoles et de leurs propriétés, la recherche de classes décidables et la mise en œuvre pratique des méthodes de vérification. Nous avons apporté une contribution pour chacun de ces problèmes mais des améliorations et des prolongements sont possibles.

Modélisation. Nous avons proposé un modèle très général sous forme de clauses de Horn (chapitre 3). Il nous a en particulier permis de démontrer un résultat de réduction sur le nombre de participants nécessaires à une attaque. Ce résultat de réduction n'est pas une surprise et était souvent utilisé en pratique sans justification. Néanmoins, le démontrer dans un cadre très général a permis d'en donner la portée et de mettre en évidence l'importance de l'hypothèse « un agent peut se parler à lui-même ». Ce modèle paraît plus général que la plupart des modèles de traces mais il n'est pas comparable au spi-calcul où le secret est exprimé sous forme d'équivalence observationnelle.

Nous avons développé au chapitre 5 une équivalence entre l'équivalence observationnelle du spi-calcul et une équivalence sur un système de transitions, plus proche des systèmes de trace comme ceux décrits aux chapitres 3 et 4 par exemple. Nous avons montré comment exploiter ce résultat pour démontrer le secret du protocole de Needham-Schroeder-Lowe dans le cadre du spi-calcul. À notre connaissance, il s'agit de la première preuve du protocole de Needham-Schroeder-Lowe pour l'équivalence observationnelle. La méthode présentée montre que le point crucial de la preuve consiste en la recherche d'un environnement tel que sa connaissance soit invariante par application des règles du protocole. L'exemple du protocole de Needham-Schroeder-Lowe laisse à penser qu'il est possible de mettre au point des méthodes automatiques pour engendrer des environnements « candidats ». On pourrait ainsi s'inspirer des méthodes de génération d'invariants développées pour l'outil TAPS [Coh00]. Une autre approche consiste à améliorer encore notre résultat d'équivalence de manière à ramener l'équivalence observationnelle à un problème d'accessibilité. Cela nous permettrait d'utiliser les outils existants pour pouvoir prouver l'équivalence observationnelle. Ce dernier point reste assez hypothétique car il semble difficile de traduire la bisimulation en une propriété d'accessibilité. À notre connaissance, il n'existe aucun outil permettant de prouver l'équivalence observationnelle.

Plus généralement, nous nous sommes principalement intéressés dans cette thèse à des propriétés de secret ou d'authentification. Comme nous l'avons vu dans l'introduction, il existe d'autre propriétés pertinentes pour les protocoles cryptographiques comme l'anonymat ou l'équité. Il serait donc intéressant d'étendre la recherche de classes décidables à ces autres propriétés, voire de déterminer toute une classe de propriétés décidables comme nous avons commencé à le faire dans le modèle sous forme de clauses de Horn. Mais des extensions à d'autres propriétés de sécurité demandent un important travail préalable de définition. À l'heure actuelle, aucune des nombreuses propriétés d'authentification ne s'avère totalement satisfaisante et beaucoup ne sont définies que pour un protocole donné. Définir une sémantique claire pour les propriétés de sécurité, voire une logique appropriée apparaît donc comme un élément déterminant pour l'étude des protocoles.

Classes décidables. De notre modèle, nous avons extrait deux fragments décidables (les classes $\mathcal{C}$ et $\mathcal{C}^{\oplus}$ du chapitre 8) permettant de modéliser pour un nombre non borné de sessions, les protocoles sans nonces, qui ne copient qu'une partie des messages par transition. La classe $\mathcal{C}^{\oplus}$ modélise également les propriétés algébriques du « ou » exclusif. Il s'agit du premier résultat connu de décidabilité pour l'ajout du « ou » exclusif dans le cadre d'un nombre non borné de sessions. Cependant, la complexité de l'algorithme (non élémentaire) est loin d'être satisfaisante. La borne de complexité donnée n'est qu'une borne supérieure que nous pouvons peut-être améliorer. Même si la complexité théorique s'avérait élevée, il semble possible de mettre en place des stratégies de résolution plus fines pour assurer une terminaison plus rapide de la saturation, au moins en pratique.

D'autre part, nous avons discuté au chapitre 8 des extensions possibles de ces classes, notamment pour d'autres propriétés algébriques pertinentes comme celle de l'exponentielle modulaire. Mais toutes ces classes de décidabilité souffrent d'une restriction importante : les protocoles modélisés ne comportent pas de nonces ou alors les nonces sont modélisés par un nombre fini de constantes. Il s'agit bien sûr d'une abstraction correcte mais qui introduit de fausses attaques. Par ailleurs, la plupart des outils de preuve fonctionnent de manière satisfaisante sur des protocoles n'appartenant à aucune classe décidable connue. Un prolongement ambitieux de cette thèse serait donc de déterminer une classe décidable de protocoles permettant à la fois une modélisation fidèle des nonces et un nombre arbitraire de sessions. Pour ce faire, nous pourrions utiliser un résultat de réduction sur le nombre de sessions, similaire à celui obtenu pour le nombre de participants. Cependant, comme nous l'avons expliqué au paragraphe 6.2.2 du chapitre 6, notre restriction à une seule copie par transition ne suffit pas car R. Amadio et W. Charatonik [AC02] ont montré que le secret était indécidable pour des protocoles dont la taille des messages est bornée et qui n'utilisent que la primitive du chiffrement. Il nous faut donc envisager des restrictions supplémentaires. Nous pourrions ainsi considérer les protocoles qui n'utilisent qu'un seul déchiffrement et chiffrement par étape, hypothèse satisfaite par de nombreux protocoles.

Outil. L'outil Securify que nous décrivons au chapitre 7 fut l'un des premiers à *prouver* des propriétés de secret et il a permis de vérifier de nombreux protocoles de [CJ97]. Cependant, pour son bon fonctionnement, il faut que les protocoles soient suffisamment typés. De plus, les restrictions prises sur les protocoles traités (pas de clefs composées par exemple) ne sont peut-être pas nécessaires. Il faudrait donc reprendre et simplifier l'algorithme pour un modèle représentant une classe plus large de protocoles, comme le modèle sous forme de clauses de Horn (chapitre 3) par exemple. Cela permettrait dans un premier temps de mettre en évidence les hypothèses réellement pertinentes (comme peut-être le typage) pour la terminaison de l'algorithme en pratique et pour éviter de trop nombreuses « fausses attaques ». Dans un deuxième temps, nous pourrions l'améliorer d'une part en ajoutant de nouveaux « tests de base » pour garantir une meilleure terminaison et d'autre part en l'étendant à certaines théories équationnelles comme celle du « ou » exclusif.

Quatrième partie

Annexes

ANNEXE A

Correspondance entre le modèle Millen-Rueß et le modèle sous forme de clauses de Horn

Dans cette annexe, nous prouvons le théorème 4.2 énoncé au chapitre 4. Ce théorème assure qu'un protocole P est secret si et seulement si sa traduction l'est.

Nous reprenons dans ce qui suit toutes les notations introduites lors de la traduction d'un protocole en un ensemble de clauses de Horn.

Théorème (théorème 4.2) *Soit P un protocole du modèle Millen-Rueß. On suppose que pour toute instanciation des variables d'agents et pour tout historique H, toutes les règles du protocole peuvent être jouées dans un certain ordre à partir de H. Soit I_0 un ensemble de messages représentant la connaissance initiale de l'intrus, tel que I_0 puisse être défini par un ensemble de clauses de Horn. Soit $\mathcal{C}_P$ l'ensemble de clauses correspondant à P, défini au paragraphe 4.2.2 et Φ_P l'ensemble des propriétés de sécurité associées à P.*

Alors P n'est pas secret pour la connaissance initiale I_0 si et seulement si il existe une attaque sur $\mathcal{C}_P$ pour une des propriétés de Φ_P.

On rappelle que

$$\mathcal{C}_P \stackrel{\text{def}}{=} \mathcal{I}_0 \cup \mathcal{C}_{I_1} \cup \mathcal{C}_{I_2} \cup \mathcal{C}_{\mathsf{aux}} \cup C_{\mathcal{N}} \cup C_{\mathcal{K}} \cup \mathcal{C}_{\mathsf{regles}}.$$

Soit $\mathcal{H}_P$ le plus petit modèle de Herbrand de $\mathcal{C}_P$. On considère σ la substitution définie au paragraphe 4.2.1 pour établir la correspondance entre la connaissance initiale du modèle MR et celle du modèle sous forme de clauses de Horn.

On vérifie par récurrence sur la longueur de la trace que si $T(t) \in \mathcal{H}_P$, alors t est de la forme $[e_1, s_1] \cdot \ldots \cdot [e_n, s_n] \cdot \perp$. On peut donc définir $\mathcal{E}(t) = \{e_i \mid 1 \leqslant i \leqslant n\}$, l'ensemble des événements apparaissant dans t et $\mathcal{M}(t)$ l'ensemble des messages de $\mathcal{E}(t)$, pour chaque t tel que $T(t) \in \mathcal{H}_P$.

De plus, les messages que l'intrus peut déduire des messages apparaissant dans la trace t sont exactement les messages m tels que $I(m, t) \in \mathcal{H}_P$.

Lemme A.1 *Soit t tel que $T(t) \in \mathcal{H}_P$.*

$$m \in \mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0)) \text{ si et seulement si } I(m,t) \in \mathcal{H}_P.$$

Preuve. Soit $E = \{m | I(m,t) \in \mathcal{H}_P\}$. Montrons que $\mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0)) \subset E$.

- Par construction de $\mathcal{C}_{I_0}$, pour tout $m \in \sigma(I_0)$, on a $\mathcal{C}_{I_0} \vdash I'_0(m\sigma)$ et donc, en utilisant la clause 4.7, $\mathcal{C}_{I_0} \vdash I(m,t)$.
- En appliquant la clause «d'interception» 3.9 de l'ensemble $\mathcal{C}_{I_2}$, tout message $m \in \mathcal{M}(t)$ vérifie que $I(m,t) \in \mathcal{H}_P$.
- Il reste à montrer que E est clos pour les opérateurs analz et synth, ce qui est assuré par les clauses de $\mathcal{C}_{I_1}$.

Cela permet de conclure que $\mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0)) \subset E$.

Inversement, montrons par récurrence sur n que :

$$I(m,t) \in F_P^n(\emptyset) \Rightarrow m \in \mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0)),$$

où F_P^n est l'opérateur de conséquence immédiate associé à $\mathcal{C}_P$: $\mathcal{H}_P = \bigcup_{n \geqslant 0} F_P^n(\emptyset)$.

Pour $n = 0$, $F_P^0(\emptyset) = \emptyset$, il n'y a rien à démontrer.

Supposons la propriété vraie pour $k \leqslant n$. Soit m un message tel que $I(m,t) \in F_P^{n+1}(\emptyset)$. On considère la clause qui a engendré $I(m,t)$. Elle est nécessairement une clause de $\mathcal{C}_I$. Dans chaque cas, on conclut par induction.

- Si la clause 4.7 a été appliquée, alors $\mathcal{C} \vdash I'_0(m)$, et donc $m \in \sigma^{-1}(I_0)$.
- Si une des clauses de $\mathcal{C}_{I_1}$ a été appliquée, alors en utilisant l'hypothèse de récurrence et le fait que $\mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0))$ est clos par analz et synth, on déduit $m \in \mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0))$.
- Si la clause d'interception 3.9 de l'ensemble $\mathcal{C}_{I_2}$ a été appliquée, alors par construction de $\mathcal{M}(t)$, on a $m \in \mathcal{M}(t)$ et donc $m \in \mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0))$.
- Enfin, si la clause de mémorisation 3.10 de l'ensemble $\mathcal{C}_{I_2}$ a été appliquée, alors on conclut par hypothèse de récurrence.

On a donc vérifié que $E = \mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0))$. □

On peut maintenant prouver la première implication du théorème 4.2. On suppose que la substitution σ est étendue aux événements sans variables de la même façon que $\overline{\cdot}$ a été étendu aux événements avec variables. Le lemme qui suit établit une correspondance entre les historiques accessibles du protocole du modèle de MR et les traces valides du protocole sous forme de clauses.

Lemme A.2 *Soit H un historique accessible : $H \in reachable(P,I)$. Il existe une substitution $\tilde{\sigma}$ étendant σ dont le domaine est inclus dans l'ensemble des messages atomiques ($Agent \cup Nonce \cup Key$), il existe un terme t_H tel que $T(t_H) \in \mathcal{H}_P$ et, pour chaque événement e de H :*

1. *Si e est un message, alors $I(e\tilde{\sigma}, t_H) \in \mathcal{H}_P$.*
2. *Si e est un spell, alors il existe un numéro de session $s^n(0)$ tel que :*

$$\mathsf{In}([e\tilde{\sigma}, s^n(0)], t_H) \in \mathcal{H}_P.$$

3. *Si e est un état, $e = \mathsf{a}_{i,j}(m)$, alors il existe un numéro de session $s^n(0)$ tel que :*

$$\mathsf{In}([e\tilde{\sigma}, s^n(0)], t_H) \in \mathcal{H}_P \quad \text{et} \quad \mathsf{NotPlayed}(a\tilde{\sigma}, s^i(0), s^{j+1}(0), s^n(0), t_H) \in \mathcal{H}_P.$$

4. *Si $n \in Nonce$ apparaît dans t_H, alors* $\mathsf{Nonce}(n\tilde{\sigma}) \in \mathcal{H}_P$. *De même pour les clefs, si $k \in Key$ apparaît dans t_H, alors* $\mathsf{Clef}(k\tilde{\sigma}) \in \mathcal{H}_P$.

Preuve. On prouve le lemme A.2 par récurrence sur le nombre d'étapes nécessaires à l'obtention de H. Si $H = H_0 = \emptyset$, la propriété est vérifiée.

Soit H' accessible en $n+1$ étapes. Il existe un historique H tel que $global(P, I_0)(H, H')$. Par hypothèse de récurrence, il existe une substitution $\tilde{\sigma}$ et un terme t_H tels que, pour chaque événement e de H, les propriétés 1, 2, 3 et 4 sont satisfaites.

Soit H' est un successeur malhonnête de H : $H' = H \cup \{m\}$ où $m \in \mathsf{fake}(Msg(H) \cup I_0)$. On pose $t_{H'} = t_H$. Les propriétés 1 et 3 sont satisfaites pour tout événement e de H, il suffit donc de montrer que m vérifie la propriété 1.

Par hypothèse de récurrence, pour tout message m' de H, on a $I(m'\tilde{\sigma}, t_H) \in \mathcal{H}_P$. Donc, d'après le lemme A.1, pour tout message m' de H, $m'\tilde{\sigma} \in \mathsf{fake}(\mathcal{M}(t) \cup \tilde{\sigma}(I_0))$. Cela implique $\tilde{\sigma}(Msg(H)) \subseteq \mathsf{fake}(\mathcal{M}(t) \cup \tilde{\sigma}(I_0))$ et donc $\mathsf{fake}(\tilde{\sigma}(Msg(H) \cup I_0)) \subseteq \mathsf{fake}(\mathcal{M}(t) \cup \tilde{\sigma}(I_0))$. On en déduit $m\tilde{\sigma} \in \mathsf{fake}(\mathcal{M}(t) \cup \tilde{\sigma}(I_0))$. En appliquant une nouvelle fois le lemme A.1, on obtient $I(m\tilde{\sigma}, t_H) \in \mathcal{H}_P$, la propriété 1 est satisfaite.

Soit H' est un successeur honnête de H : il existe une transition t telle que $t = rl\theta$ pour une règle rl de P.

$$t = Pre(t) \xrightarrow{New(t)} \{m_1, \ldots, m_k, st_1, \ldots, st_m, e_1, \ldots, e_l\}.$$

La transition t vérifie que $Pre(t) \subseteq H$, $New(t) \subseteq H$ et

$$H' = (H \backslash (Pre(t) \cap States)) \cup Post(t).$$

On note N_1 (respectivement K_1) l'ensemble des nonces (respectivement des clefs) apparaissant dans $Pre(t)$. De même, on pose N_2 (respectivement K_2) l'ensemble des nonces (respectivement des clefs) apparaissant dans $New(t)$.

D'après l'hypothèse de récurrence, pour tout message m de H, on a $I(m\tilde{\sigma}, t_H) \in \mathcal{H}_P$ et pour tout spell c de H, il existe n_c tel que $\mathsf{In}([c\tilde{\sigma}, s^{n_c}(0)], t_H) \in \mathcal{H}_P$. De plus, pour tout nonce $n \in N_1$, on a $\mathsf{Nonce}(n\tilde{\sigma}) \in \mathcal{H}_P$, pour toute clef $k \in K_1$, on a $\mathsf{Clef}(k\tilde{\sigma}) \in \mathcal{H}_P$.

Pour les états, deux cas sont alors possibles : ou bien $Pre(t)$ est vide, ou bien $Pre(t)$ est un singleton. Dans le premier cas, on choisit n tel que $\mathsf{Fresh}(s^n(0), t_H) \in \mathcal{H}_P$. Les messages basiques introduits dans H' sont ceux de $New(t)$. Pour tout nonce $n \in New(t)$, $n = N\theta$ pour un certain $N \in New(rl)$ et on a définit $\overline{N} = n(\mathsf{numero}(N), l, s)$. On étend donc $\tilde{\sigma}$ par $\tilde{\sigma}(n) = n(\mathsf{numero}(N), l\tilde{\sigma}, s^n(0))$. De même, on étend $\tilde{\sigma}$ sur les clefs de $New(t)$. On pose :

$$\begin{aligned} t_{H'} = {} & [m_1\tilde{\sigma}, s^n(0)] \cdots [m_k\tilde{\sigma}, s^n(0)] \cdot \\ & [st_1\tilde{\sigma}, s^n(0)] \cdots [st_m\tilde{\sigma}, s^n(0)] \cdot [e_1\tilde{\sigma}, s^n(0)] \cdots [e_l\tilde{\sigma}, s^n(0)] \cdot t_H. \end{aligned}$$

En appliquant alors la clause $\overline{rl}$, on obtient que : $T(t_{H'}) \in \mathcal{H}_P$. En appliquant ensuite les règles d'interception 3.9 et de mémorisation 3.10 de l'ensemble $\mathcal{C}_{I_2}$ et les clauses $\mathcal{C}_{\mathsf{aux}}$, on vérifie que les propriétés 1, 2 et 3 sont satisfaites. Pour les états introduits, on utilise en particulier que le numéro de session utilisé est frais pour garantir que l'état successeur n'est pas déjà dans la trace. La propriété 4 est garantie par les clauses de $\mathcal{C}_{\mathcal{N}}$ et $\mathcal{C}_{\mathcal{K}}$ ajoutées lors de la traduction de la règle rl.

Si $Pre(t)$ est un singleton : $Pre(t) = \{\mathsf{a}_{i,j}(m)\}$. Alors $Post(t) = \{\mathsf{a}_{i,j+1}(m')\}$. D'après l'hypothèse de récurrence, il existe $n \in \mathbb{N}$ tel que :

$$\mathsf{In}([st(a\tilde{\sigma}, s^i(0), s^j(0), m\tilde{\sigma}), s^n(0)], t_H) \in \mathcal{H}_P \quad \text{et} \quad \mathsf{NotPlayed}(a\tilde{\sigma}, s^i(0), s^{j+1}(0), s^n(0), t_H) \in \mathcal{H}_P.$$

De la même manière que précédemment les basiques introduits dans H' sont ceux de $New(t)$. Pour tout nonce $n \in New(t)$, $n = N\theta$ pour $N \in New(rl)$ et on a définit $\overline{N} = n(\mathsf{numero}(N), l, s)$. On étend donc $\tilde{\sigma}$ par $\tilde{\sigma}(n) = n(\mathsf{numero}(N), l\tilde{\sigma}, s^n(0))$. De même, on étend $\tilde{\sigma}$ sur les clefs de $New(t)$. Enfin, on pose :

$$t_{H'} = [m_1\tilde{\sigma}, s^n(0)] \cdots [m_k\tilde{\sigma}, s^n(0)] \cdot \\ [st(a\tilde{\sigma}, s^i(0), s^{j+1}(0), m'\tilde{\sigma}), s^n(0)] \cdot [e_1\tilde{\sigma}, s^n(0)] \cdots [e_l\tilde{\sigma}, s^n(0)] \cdot t_H,$$

et les propriétés 1, 2, 3 et 4 sont vérifiées. □

Supposons maintenant que P n'est pas secret pour la connaissance initiale I_0. Alors il existe un historique accessible H et un spell $C \in H$, compatible avec I_0, tels que $\mathsf{fake}(H) \cap Sec(C) \neq \emptyset$. $C = \{n_1, \ldots, n_p\} \ddagger \{a_1, \ldots, a_q\}$. Comme S est compatible avec I_0, les a_i représentent des agents honnêtes, c'est-à-dire $a_1, \ldots, a_q \in \mathsf{HA}$. Par construction du système de contraintes, $\forall i \in \{1, \ldots, q\}$, $\mathsf{Ha}(a_i\tilde{\sigma})$ est vérifiée.

On construit $\tilde{\sigma}$ et t_H en suivant le lemme A.2. En particulier, $T(t_H) \in \mathcal{H}_P$.

- Ou bien il existe i tel que $\mathsf{prv}(a_i)$ ou $\mathsf{shr}(\sigma(a_i))$ soit dans $\mathsf{fake}(H) \cap Sec(C)$. Supposons que $\mathsf{prv}(a_i) \in \mathsf{fake}(H) \cap Sec(C)$ (le cas $\mathsf{shr}(a_i) \in \mathsf{fake}(H) \cap Sec(C)$ est traité exactement de la même façon). En appliquant les lemmes A.1 et A.2, on obtient $I(\mathsf{prv}(\tilde{\sigma}(a_i)), t_H) \in \mathcal{H}_P$ ce qui contredit la clause

$$\neg I(\mathsf{prv}(a), t) \vee \neg T(t) \mid \mathsf{Ha}(a) \quad \in \Phi_P.$$

- Ou bien il existe i tel que $n_i \in \mathsf{fake}(H) \cap Sec(C)$. De la même manière, $I(n_i\tilde{\sigma}, t_H) \in \mathcal{H}_P$ ce qui falsifie la clause

$$\neg T(t) \vee \neg I(\overline{n_i}, t) \quad \mid \mathsf{Ha}(a_1) \wedge \cdots \wedge \mathsf{Ha}(a_q) \wedge \mathsf{Num}(s) \quad \in \Phi_P.$$

Dans tous les cas, on conclut qu'il existe une attaque sur $\mathcal{C}_P$ pour une des propriétés de Φ_P.

Réciproquement, nous allons montrer que s'il existe une attaque sur $\mathcal{C}_P$ pour une des propriétés de Φ_P, alors P n'est pas secret pour la connaissance initiale I_0. Le lemme suivant montre qu'à toute trace valide de $\mathcal{H}_P$ correspond un historique accessible pour P et I_0.

Une trace valide t garde tous les états passés en mémoire, contrairement à l'historique H qui ne garde que les états «actifs», les autres étant effacés au fur et à mesure. On définit une relation de succession $\prec_t$, relative à une trace t, entre les états de la manière suivante :

$$st(a_1, k_1, s^i(0), m_1) \prec_t st(a_2, k_2, s^j(0), m_2) \Leftrightarrow a_1 = a_2, k_1 = k_2, i \leqslant j \text{ et} \\ \exists n, \mathsf{In}([st(a_1, k_1, s^i(0), m_1), s^n(0)], t), \mathsf{In}([st(a_2, k_2, s^j(0), m_2), s^n(0)], t) \in \mathcal{H}_P.$$

Lemme A.3 *Soit t un terme tel que $T(t) \in \mathcal{H}_P$. Il existe un historique accessible $H_t \in reachable(P, I_0)$ et une substitution injective $\tilde{\sigma}$ tels que $\tilde{\sigma}$ étende σ^{-1} et :*

1. *$\tilde{\sigma}(\mathcal{M}(t)) \subseteq Msg(H_t)$ et $Msg(H_t) \subseteq \mathsf{fake}(\tilde{\sigma}(\mathcal{M}(t)) \cup I_0)$,*

2. *$\tilde{\sigma}(\mathcal{E}(t)) \cap Spec = H_t \cap Spec$,*
3. *$H_t \cap States \subseteq \tilde{\sigma}(\mathcal{E}(t))$ et pour tout état $st \in \tilde{\sigma}(\mathcal{E}(t))$, si st est maximal dans $\mathcal{E}(t)$ pour $\prec_t$, alors $st \in H_t$.*
4. *Si un terme de la forme $n(i, [a_1, \ldots, a_q], s^n(0))$ apparaît dans t, alors il existe un spell $C \in H_t$ tel que $C = S \ddagger \{\tilde{\sigma}(a_1), \ldots, \tilde{\sigma}(a_q)\}$ et $n(i, [a_1, \ldots, a_q], s^n(0))\tilde{\sigma} \in S$.*

Preuve. La preuve se fait par récurrence sur l'entier minimal n tel que $T(t) \in F_P^n(\emptyset)$.

Pour $n = 0$, $F_P^n(\emptyset) = \emptyset$, il n'y a rien à prouver.

Supposons la propriété vraie pour $k \leqslant n$. Soit une trace t telle que $T(t) \in F_P^{n+1}(\emptyset)$. On considère la clause C qui a engendré $T(t)$. Soit c'est la clause 3.27 qui génère la trace vide :

$$\Rightarrow T(\bot)$$

et alors $H_t = \emptyset$ convient.

Soit c'est une des clauses de $\mathcal{C}_{\mathsf{regles}}$. Il existe une règle rl de P et une substitution θ telles que $C = C'\theta$ où $C' \in \overline{rl}^{\mathcal{E}}$. Supposons que $C' = \overline{rl}$ (les autres cas sont identiques). On suppose de plus que $Pre(rl)$ ne contient qu'un seul état $st = \mathsf{A}_{i,j}(m)$ car le cas où $Pre(rl) = \emptyset$ est plus simple et similaire. Alors :

$$rl = Pre(t) \stackrel{New(t)}{\longrightarrow} \{m_1, \ldots, m_k, \mathsf{A}_{i,j+1}(m'), e_1, \ldots, e_l\}.$$

On obtient :

$$t = ([\overline{m_1}, s] \cdots [\overline{m_k}, s] \cdot [st(A, s^i(0), s^{j+1}(0), \overline{m'}), s] \cdot [\overline{e_1}, s] \cdots [\overline{e_l}, s] \cdot t')\theta$$

et $T(t'\theta) \in \mathcal{H}_P$. Par hypothèse de récurrence, il existe un historique accessible $H_{t'\theta}$, une substitution $\tilde{\sigma}$ qui vérifie les propriétés du lemme A.3. De plus, pour tout $m \in Pre(rl) \cap Fields$, $I(\overline{m}, t)\theta \in \mathcal{H}_P$. D'après le lemme A.1, $\overline{m}\theta \in \mathsf{fake}(\mathcal{M}(t) \cup \sigma(I_0))$ et donc, d'après l'hypothèse de récurrence, $\overline{m}\theta\tilde{\sigma} \in \mathsf{fake}(H_{t'\theta} \cup I_0)$. Quitte à appliquer plusieurs fois des transitions malhonnêtes à $H_{t'\theta}$, on construit un historique accessible

$$H'_{t'\theta} = H_{t'\theta} \cup \bigcup_{m \in Pre(rl) \cap Fields} \overline{m}\theta\tilde{\sigma}.$$

De plus, toujours par hypothèse de récurrence, les spells correspondants à ceux de t sont dans $H_{t'\theta}$ et donc dans $H'_{t'\theta}$:

$$\bigcup_{C \in Pre(rl) \cap Spec} \overline{C}\theta\tilde{\sigma} \subseteq H'_{t'\theta}.$$

Enfin, comme le littéral $\mathsf{NotPlayed}(\overline{A}, s^i(0), s^{j+1}(0), s, t)\theta$ est dans $\mathcal{H}_P$, $\overline{st}\theta$ est maximal dans $\mathcal{E}(t'\theta)$ pour la relation $\prec_{t'\theta}$ et donc, en appliquant une dernière fois l'hypothèse de récurrence, $st\tilde{\sigma} \in H'_{t'\theta}$.

Soit ρ la substitution définie sur les variables de $Pre(rl)$, telle que pour toute variable X de $Pre(rl)$, $\rho(X) = \overline{X}\theta\tilde{\sigma}$. On étend alors ρ sur les variables $New(rl)$, de telle façon que $\rho(New(rl))$ soit un ensemble de messages basiques non utilisés dans $H'_{t'\theta}$. La règle $rl\rho$ est applicable à $H'_{t'\theta}$, on pose H_t le successeur honnête de $H'_{t'\theta}$ obtenu par application de $rl\rho$.

On étend alors $\tilde{\sigma}$ sur $\theta'(New(rl))$ de la manière suivante : $\tilde{\sigma}(N\theta') = \overline{N}\theta$. $H'_{t'\theta}$ et $\tilde{\sigma}$ ainsi obtenus satisfont les propriétés requises par le lemme, ce qui conclut la preuve. □

Supposons maintenant qu'il y ait une attaque sur $\mathcal{C}_P$ pour une propriété ϕ de Φ_P. Soit $\mathcal{H}_1 \subset \mathcal{H}_P$ un contre exemple pour ϕ. Montrons que P n'est pas secret pour I_0.

- Ou bien $\phi = \neg I(\mathsf{prv}(a), t) \vee \neg T(t) \mid \mathsf{Ha}(a)$ (ou $\phi = \neg I(\mathsf{shr}(a), t) \vee \neg T(t) \mid \mathsf{Ha}(a)$, cas similaire). Comme $\mathcal{H}_1 \not\models \phi$, il existe $a_0 \in \mathsf{Ha}$ et t_0 tels que $I(\mathsf{shr}(a_0), t_0), T(t_0) \in \mathcal{H}_1$. Soient $\tilde{\sigma}$ et H_{t_0} la substitution et l'historique associés à t_0 par le lemme A.3. Comme Φ_P n'est pas vide, une des règles de P au moins contient un spell $C = S \ddagger \{A_1, \ldots, A_q\}$. On a supposé que, pour toute instanciation des variables d'agents, toutes les règles du protocole pouvaient être jouées dans un certain ordre. Soit H l'historique obtenu à partir de H_{t_0} en jouant ainsi toutes les règles du protocole pour une instanciation des variables d'agents vérifiant $A_1 = \cdots = A_q = a_0\tilde{\sigma}$. Comme $I(\mathsf{shr}(a_0), t_0) \in \mathcal{H}_P$ et en appliquant le lemme A.1, on déduit que $\mathsf{shr}(\tilde{\sigma}(a_0)) \in \mathsf{fake}(H_{t_0} \cup I_0)$. Donc $H' = H \cup \{\mathsf{shr}(\tilde{\sigma}(a_0))\}$ est également un historique accessible : $H' \in \mathit{reachable}(P, I_0)$. Comme $\tilde{\sigma}(a_0) = \sigma(a_0) \in \mathsf{HA}$, C est un spell compatible avec I_0. Or :

$$\mathsf{shr}(\tilde{\sigma}(a_0)) \in \mathit{Sec}(S) \cap \mathsf{fake}(H' \cup I_0) \neq \emptyset,$$

donc P n'est pas secret pour la connaissance initiale I_0.
- Ou bien $\phi = \neg T(t) \vee \neg I(\overline{n_i}, t) \quad \mid \mathsf{Ha}(a_1) \wedge \cdots \wedge \mathsf{Ha}(a_q) \wedge \mathsf{Num}(s)$. De la même manière, il existe $a_1, \ldots, a_q \in \mathsf{Ha}$, il existe t_0 et $n \in \mathbb{N}$ tels que $I(n(i, [a_1; \ldots; a_q], s^n(0)), t_0), T(t_0) \in \mathcal{H}_1$. On considère à nouveau $\tilde{\sigma}$ et H_{t_0} la substitution et l'historique associés à t_0 par le lemme A.3. D'après le lemme A.3, il existe $C = S \ddagger \{\tilde{\sigma}(a_1), \ldots, \tilde{\sigma}(a_q)\} \in H_{t_0}$ tel que $n(i, [a_1, \ldots, a_q], s^n(0))\tilde{\sigma} \in S$. Comme les a_i sont dans HA, C est compatible avec I_0. De plus, d'après le lemme A.1, $I(n(i, [a_1; \ldots; a_q], s^n(0)), t_0) \in \mathcal{H}_P$ assure

$$\tilde{\sigma}(n(i, [a_1; \ldots; a_q], s^n(0))) \in \mathsf{fake}(H_{t_0} \cup I_0)$$

et donc P n'est pas secret pour la connaissance initiale I_0, ce qui termine la démonstration du théorème 4.2.

ANNEXE B

Description des protocoles testés par l'outil Securify

L'objet de cette annexe est de présenter les spécifications manquantes des protocoles mentionnés dans le chapitre 7.

B.1 Protocole de Needham-Schroeder-Lowe

Au chapitre 7, nous avons présenté les deux types entrées (`nsl.eva` ou `nsl.ml`) que l'on peut fournir à notre outil. La spécification du protocole en *eva* est tout d'abord traduite en une spécification en *cpl* avec le traducteur automatique *evatrans* développé dans le projet EVA. Le fichier `nsl.cpl` correspondant à la traduction du fichier `nsl.eva` est donné aux pages 218-220.

Notre outil permet de faire la preuve de ce protocole et donne les quatre arbres de preuve correspondants à chacune des branches des règles du protocole. Ces quatre arbres de preuves, générés automatiquement par notre outil, sont présentés aux figures B.1 et B.2.

Remarque : Le test des « secrets disjoints » noté ds est noté db sur les figure car c'est ainsi que nous l'avions appelé auparavant pour « disjoint book ».

B.2 Description des protocoles utilisés au chapitre de présentation de l'outil

Nous donnons aux figures B.3, B.4, B.5 et B.6 les spécifications en *eva* des huit protocoles mentionnés au paragraphe 7.2.2.1 dans le tableau des résultats obtenus par l'outil.

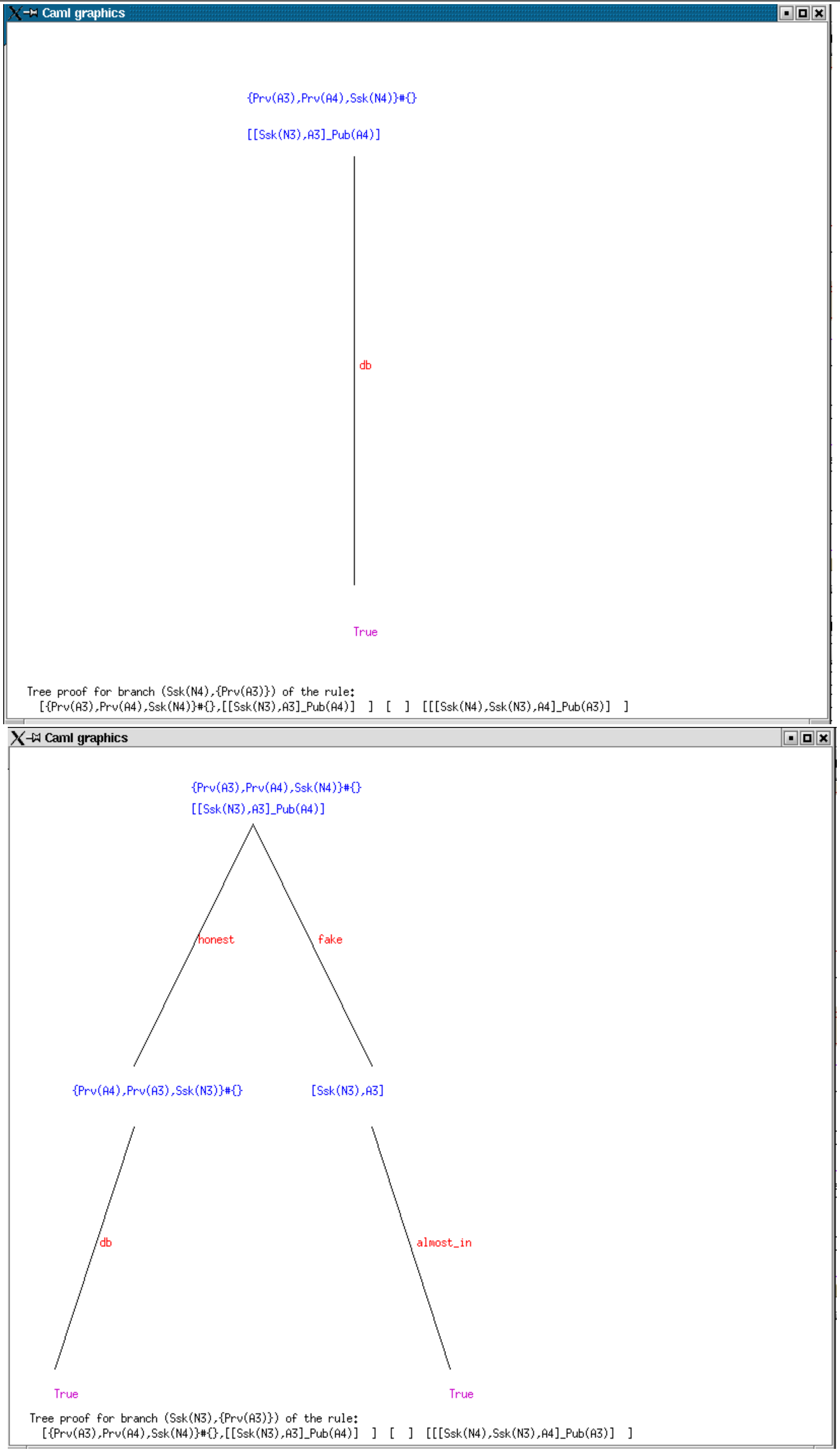


FIGURE B.1 – *Arbres de preuves obtenus pour le protocole de Needham-Schroeder-Lowe.*

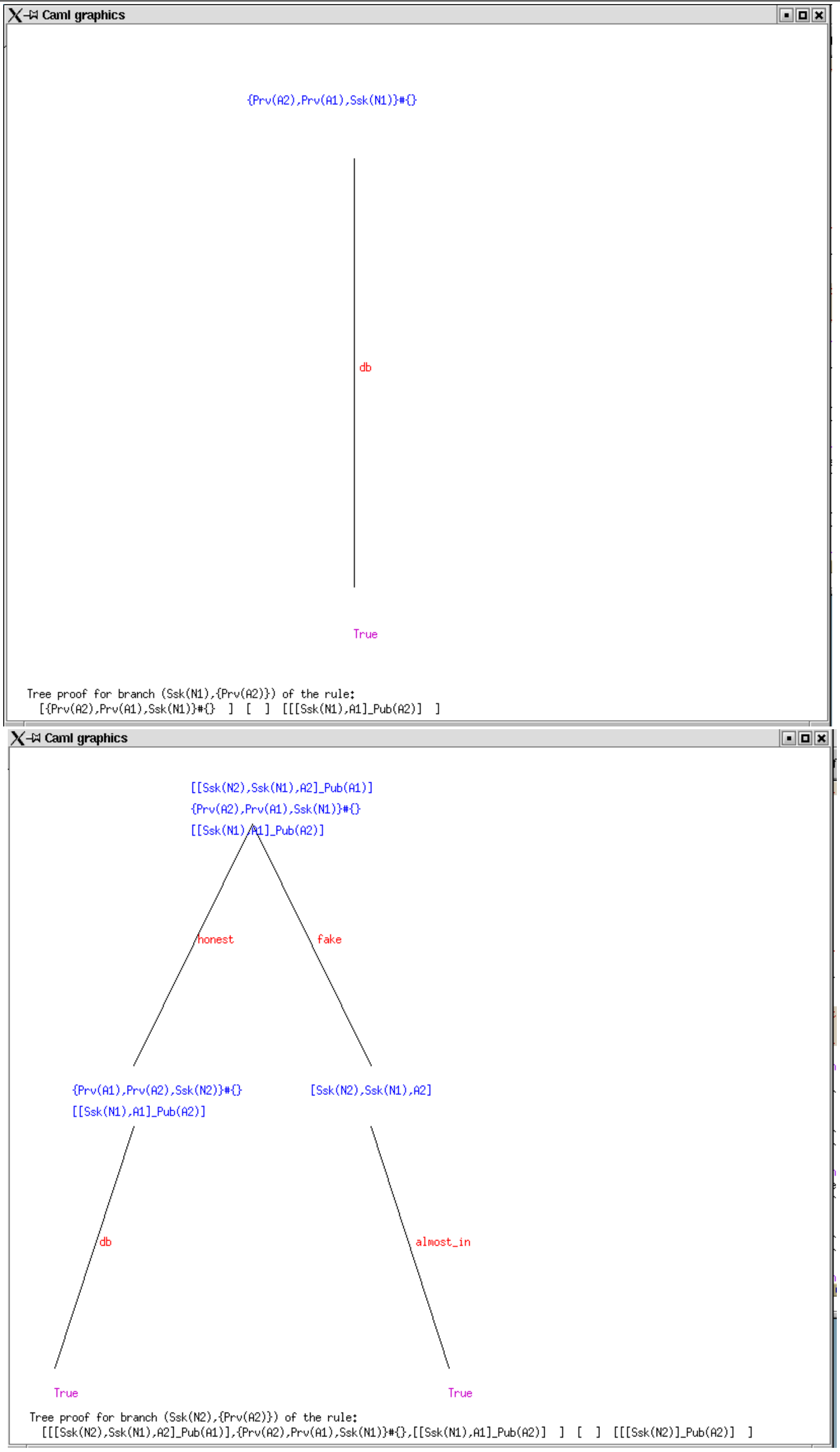


FIGURE B.2 - *Arbres de preuves obtenus pour le protocole de Needham-Schroeder-Lowe.*

Début du fichier `nsl.cpl`

```
(compiled-spec
  (types
    (type
      "**asymk**"
      (one-way-function
        "number"
        "*D*"))
    (type
      "*key_of_D*"
      (function
        "key"
        "*D*"))
    (type
      "*number_of_key*"
      (function
        "number"
        "key"))
    (type
      "A"
      "principal")
    (type
      "B"
      "principal")
    (type
      "Na"
      "key")
    (type
      "Nb"
      "key")
    (type
      "PK"
      (one-way-function
        "number"
        "principal"))
    (type
      "SK"
      (one-way-function
        "number"
        "principal"))
    (type
      "_kpf1"
      (one-way-function
        "number"
        "principal"))
    (type
      "alg"
      "asym_algo"))
  (values
    (alias
      "SK"
      (lambda-privk
        (aa
          "alg")
        "_kpf1"))
    (alias
      "PK"
      (lambda-pubk
        (aa
          "alg")
        "_kpf1")))
  (axioms)
  (system
    (repeat-process
      (private
        "A"
        "B")
      "s1.A"
      "1."
      (knows
        (as
          "alg"
          "alg")
        (as
          "Na"
          (apply
            "*key_of_D*"
            "Na-1"))
        (as
          "Nb"
          "_x3")
        (as
          "A"
          "A")
        (as
          (lambda-pubk
            (aa
              "alg")
            "_kpf1")
          (lambda-pubk
            (aa
              "alg")
            "_kpf1"))
        (as
          (hash-apply-privk
            (aa
              "alg")
            "_kpf1"
            "A")
          (hash-apply-privk
            (aa
              "alg")
            "_kpf1"
            "A"))
        (as
          "*number_of_key*"
          "*number_of_key*")
        (as
          "B"
          "B")
        (as
          "*key_of_D*"
          "*key_of_D*"))
      (trans
        "1."
        "%1."
        (new
          "Na-1"))
      (trans
        "%1."
        "2."
        (send
          (crypt
            (a
              (aa
                "alg"))
            (tuple
              (apply
                "*number_of_key*"
                (apply
                  "*key_of_D*"
                  "Na-1"))
              (p
                "A"))
            (hash-apply-pubk
              (aa
                "alg")
```

```
          "_kpf1"
          "B"))))
  (trans
    "2."
    "3."
    (recv
      (crypt
        (a
          (aa
            "alg"))
        (tuple
          (apply
            "*number_of_key*"
            "_x3")
          (exact
            (apply
              "*number_of_key*"
              (apply
                "*key_of_D*"
                "Na-1")))
          (exact
            (p
              "B")))
        (hash-apply-privk
          (aa
            "alg")
          "_kpf1"
          "A"))))
  (trans
    "3."
    "end."
    (send
      (crypt
        (a
          (aa
            "alg"))
        (apply
          "*number_of_key*"
          "_x3")
        (hash-apply-pubk
          (aa
            "alg")
          "_kpf1"
          "B")))))
(repeat-process
  (private
    "A"
    "B")
  "s1.B"
  "1."
  (knows
    (as
      "alg"
      "alg")
    (as
      (hash-apply-privk
        (aa
          "alg")
        "_kpf1"
        "B")
      (hash-apply-privk
        (aa
          "alg")
        "_kpf1"
        "B"))
    (as
      "Na"
      "_x1")
    (as
      "Nb"
      (apply
        "*key_of_D*"
        "Nb-1"))
    (as
      "A"
      "_x2")
    (as
      (lambda-pubk
        (aa
          "alg")
        "_kpf1")
      (lambda-pubk
        (aa
          "alg")
        "_kpf1"))
    (as
      "*number_of_key*"
      "*number_of_key*")
    (as
      "B"
      "B")
    (as
      "*key_of_D*"
      "*key_of_D*"))
  (trans
    "1."
    "2."
    (recv
      (crypt
        (a
          (aa
            "alg"))
        (tuple
          (apply
            "*number_of_key*"
            "_x1")
          (p
            "_x2"))
        (hash-apply-privk
          (aa
            "alg")
          "_kpf1"
          "B"))))
  (trans
    "2."
    "%4."
    (new
      "Nb-1"))
  (trans
    "%4."
    "3."
    (send
      (crypt
        (a
          (aa
            "alg"))
        (tuple
          (apply
            "*number_of_key*"
            (apply
              "*key_of_D*"
              "Nb-1"))
          (apply
            "*number_of_key*"
            "_x1")
          (p
            "B"))
        (hash-apply-pubk
          (aa
            "alg")
          "_kpf1"
          "_x2"))))
  (trans
    "3."
    "end."
    (recv
      (exact
        (crypt
          (a
            (aa
              "alg"))
          (apply
            "*number_of_key*"
            (apply
              "*key_of_D*"
              "Nb-1"))
          (hash-apply-pubk
            (aa
              "alg")
            "_kpf1"
```

```
              "B")))))))
(assume
  (secret
    (located
      "s1."
      "A"
      (hash-apply-privk
        (aa
          "alg")
        "_kpf1"
        "A")))
  (secret
    (located
      "s1."
      "B"
      (hash-apply-privk
        (aa
          "alg")
        "_kpf1"
        "B")))
  (secret
    (hash-apply-privk
      (aa
        "alg")
      "_kpf1"
      (located
        "s1."
        "A"
        "B")))
  (secret
    (hash-apply-privk
      (aa
        "alg")
      "_kpf1"
      (located
        "s1."
        "B"
        "_x2"))))
(claim
  (A
    (G
      (secret
        (located
          "s1."
          "A"
          (hash-apply-privk
            (aa
              "alg")
            "_kpf1"
            "A")))))
  (A
    (G
      (secret
        (located
          "s1."
          "B"
          (hash-apply-privk
            (aa
              "alg")
            "_kpf1"
            "B")))))
  (A
    (G
      (secret
        (apply
          "*number_of_key*"
          (located
            "s1."
            "A"
            (apply
              "*key_of_D*"
              "Na-1"))))))
  (A
    (G
      (secret
        (apply
          "*number_of_key*"
          (located
            "s1."
            "B"
            (apply
              "*key_of_D*"
              "Nb-1"))))))))
```

Fin du fichier `ns1.cpl`

```
Otway_Rees


A, B, S : principal
  Kas, Kbs, Kab : number

  basetype key
  N, Na, Nb : key

  shr (principal,principal) : number secret
  alias Kas = shr(A,S)
  alias Kbs = shr(B,S)

  A knows A, B, Kas
  B knows S, Kbs
  S knows S, shr


{
    1. A -> B : N, A, B, {Na, N, A, B}_Kas
    2. B -> S : N, A, B, {Na, N, A, B}_Kas,
                         {Nb, N, A, B}_Kbs
    3. S -> B : N, {Nb, Kab}_Kbs
    4. S -> A : N, {Na, Kab}_Kas
  }

  s1. session *{Kas, Kbs} A=A, B=B, S=S

  assume secret (Kas@s1.A), secret (Kbs@s1.B),
 secret (Kas@s1.S), secret (Kbs@s1.S)

  claim *A*G secret (Kas@s1.A),
*A*G secret (Kbs@s1.B),
*A*G secret (Kab@s1.S)
```

```
Woo_and_Lam


  A, B, S : principal

  basetype key

  Na, Nb, Kab : key

  shr (principal,principal) : number secret
  alias Kas = shr(A,S)
  alias Kbs = shr(B,S)

  A knows A, B, S, Kas
  B knows B, S, Kbs
  S knows S, shr


{
   1. A -> B : A, Na
   2. B -> A : A, Na, B, Nb
   3. A -> B : Na, Nb, {A, B, Na, Nb}_Kas
   4. B -> S : A, B, {A, B, Na, Nb}_Kas, {A, B, Na, Nb}_Kbs
   5. S -> B : {B, Na, Nb, Kab}_Kas, {A, Na, Nb, Kab}_Kbs
   6. B -> A : {B, Na, Nb, Kab}_Kas, {Na, Nb}_Kab
   7. A -> B : {Nb}_Kab
  }

  s. session *{A, B} A=A, B=B, S=S

  assume secret (Kas@s.A),
         secret (Kbs@s.B),
         secret (Kas@s.S),
         secret (Kbs@s.S)

  claim *A*G secret (Kas@s.A),
*A*G secret (Kbs@s.B),
*A*G secret (Kab@s.S)
```

FIGURE B.3 - *Spécification des protocoles d'Otway-Rees et Woo-Lam.*

```
Denning_Sacco_Key_Distribution_with_Public_Key_Protocol

  alg : asym_algo
  everybody knows alg

  A, B : principal

  basetype key
  Ka : key
  Nb : key

  keypair^alg SK, PK (principal)
  everybody knows PK

  A knows A, B, SK(A)
  B knows B, SK(B)


  {
    1. A -> B : A, {{Ka}_(SK(A))^alg}_(PK (B))^alg
    2. B -> A  : {Nb}_Ka
  }

   s. session* {Ka, Nb} A=A, B=B


  assume secret (SK(A)@s.A),
         secret (SK(B)@s.B),
         secret (SK(A@s.B))

  claim *A*G secret (Nb@s.B),
        *A*G secret (Ka@s.A)
```

```
ISO_Symmetric_Key_Two_Pass_Unilateral_Authentication

  A, B : principal

  basetype key
  Kab : key
  Text1, Text2, Text3, Nb : number

  A knows  A, B, Kab
  B knows  A, B, Kab

  {
  1. B -> A : Nb, Text1
  2. A -> B : Text3,{Nb, B,Text2}_Kab
  }

  s. session* {Kab, Text2} A=A, B=B

  assume secret (Text2@s.A),
         secret (Kab@s.A)

  claim *A*G secret (Kab@s.A),
        *A*G secret (Text2@s.A)
```

FIGURE B.4 - *Spécification du protocole de Denning-Sacco et du protocole ISO-Symmetric-Key.*

```
Needham_Schroeder_Lowe_cles_publiques

alg : asym_algo
everybody knows alg

A, B: principal

basetype key
Na, Nb : key

keypair^alg PK, SK (principal)
everybody knows PK

A knows A, B, SK(A)
B knows B, SK(B)

{
  1. A -> B : {Na, A}_(PK(B))^alg
  2. B -> A : {Na, Nb, B}_(PK(A))^alg
  3. A -> B : {Nb}_(PK(B))^alg
}

s1. session *{A,B} A=A, B=B

assume secret (SK(A)@s1.A), secret (SK(B)@s1.B),
       secret (SK(B@s1.A)), secret (SK(A@s1.B))

claim *A*G secret (SK(A)@s1.A),
      *A*G secret (SK(B)@s1.B),
      *A*G secret (Na@s1.A),
      *A*G secret (Nb@s1.B)
```

```
Needham_Schroeder_cles_publiques

alg : asym_algo
everybody knows alg

A, B: principal

basetype key
Na, Nb : key

keypair^alg PK, SK (principal)
everybody knows PK

A knows A, B, SK(A)
B knows B, SK(B)

{
  1. A -> B : {Na, A}_(PK(B))^alg
  2. B -> A : {Na, Nb}_(PK(A))^alg
  3. A -> B : {Nb}_(PK(B))^alg
}

s1. session *{A,B} A=A, B=B

assume secret (SK(A)@s1.A), secret (SK(B)@s1.B),
       secret (SK(B@s1.A)), secret (SK(A@s1.B))

claim *A*G secret (SK(A)@s1.A),
      *A*G secret (SK(B)@s1.B),
      *A*G secret (Na@s1.A),
      *A*G secret (Nb@s1.B)
```

FIGURE B.5 - *Spécification des protocoles de Needham-Schroeder et Needham-Schroeder-Lowe.*

```
Wide_Mouthed_Frog

  A, B, S : principal

  basetype key
  Na, Kab : key

  shr (principal,principal) : number secret
  alias Kas = shr(A,S)
  alias Kbs = shr(B,S)

  A knows A, B, S, Kas
  B knows B, S, Kbs
  S knows S, shr

  {
   1. A -> B : A, Na
   2. B -> S : B, {Na, A, Kab}_Kbs
   3. S -> A : {Na, B, Kab}_Kas
  }

  s. session *{A, B} A=A, B=B, S=S

  assume  secret (Kas@s.A),
          secret (Kbs@s.B),
          secret (shr(A@s.B,S@s.B)),
          secret (Kas@s.S),
          secret (Kbs@s.S)

  claim *A*G secret (Kas@s.S),
        *A*G secret (Kbs@s.S),
        *A*G secret (Kab@s.B)
```

```
KaoChow

  A, B, S : principal

  basetype key
  Na, Nb, Kab : key

  shr (principal, principal) : number secret

  A knows A, B, shr(A,S)
  B knows B, S, shr(B,S)
  S knows S, shr

  {
   1. A -> S : A, B, Na
   2. S -> B : {A, B, Na, Kab}_shr(A,S),
                    {A, B, Na, Kab}_shr(B,S)
   3. B -> A : {A, B, Na, Kab}_shr(A,S), {Na}_Kab, Nb
   4. A -> B : {Nb}_Kab
  }

  s1. session *{Kab} A=A, B=B, S=S

  assume secret (shr(A,S)@s1.A), secret (shr(B,S)@s1.B),
         secret (shr(A,S)@s1.S), secret (shr(B,S)@s1.S)

  claim *A*G secret (shr(A,S)@s1.A),
        *A*G secret (shr(B,S)@s1.B),
        *A*G secret (Kab@s1.S)
```

FIGURE B.6 - *Spécification du protocole Wide-Mouthed-Frog et du protocole de Kao-Chow.*

Bibliographie

[AB02] M. Abadi and B. Blanchet. Analysing security protocols with secrecy types and logic programs. In *Proceedings of the 29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'2002)*, pages 33–44, January 2002.

[Aba00] M. Abadi. Security protocols and their properties. In F.L. Bauer and R. Steinbrueggen, editors, *Foundations of Secure Computation*, pages 39–60, 20th Int. Summer School, Marktoberdorf, Germany, 2000. IOS Press.

[Aba02] M. Abadi. Private authentication. Proceedings of Workshop on Privacy Enhancing Technologies, 2002.

[ABB+02] A. Armando, D.A. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Bödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS security protocol analysis tool. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the 14th International Conference of Computer Aided Verification (CAV 2002)*, volume 2404 of *Lecture Notes in Computer Science*, pages 349–353, Copenhagen, Denmark, July 2002. Springer.

[AC02] R. Amadio and W. Charatonik. On name generation and set-based analysis in the dolev-yao model. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, Lecture Notes in Computer Science, pages 499–514, Brno, Czech Republic, 2002. Springer Verlag.

[AF01] 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, January 2001.

[AFG00] M. Abadi, C. Fournet, and G. Gonthier. Authentication primitives and their compilation. In *Proceedings of the 27th ACM Sympoium on Principles of Programming Languages (POPL'00)*, pages 302–315, 2000.

[AG97] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols : The spi calculus. In *proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[AG98] M. Abadi and A. Gordon. A bisimulation method for cryptographic protocols. In *Nordic Journal of Computing*, volume 5, pages 267–303. 1998.

[AL00] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 380–394, 2000.

[All93] D. M. Allester. Automatic recognition of tractability in inference relations. *Journal of the Association for Computing Machinery (ACM)*, 40(2) :284–303, April 1993.

[ALV02] R. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1) :695–740, 2002.

[AN95] R.J. Anderson and R.M. Needham. Programming Satan's computer. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441. Springer Verlag, 1995.

[And80] J.P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Compagny, Fort Washington, Pennsylvania, USA, April 1980.

[Apt90] K. R. Apt. Logic programming. In Jan Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, Formal Models and Semantics, pages 493–574. Elsevier, 1990.

[AR00] M. Abadi and P. Rogaway. Reconciling two views of cryptography. In *Proceedings of the International Conference on Theoretical Computer Science (IFIP TCS2000)*, pages 3–22, August 2000.

[ASW98] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Advances in Cryptology - EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606, 1998.

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proceedings of the Royal Society*, volume 426 of *Series A*, pages 233–271. 1989. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), 18-36.

[BCJS02] F. Butler, I. Cervesato, A. Jaggard, and Andre Scedrov. A formal analysis of some properties of kerberos 5 using MSR. In S. Schneider, editor, *Proceedings of 15th IEEE Computer Security Foundations Workshop (CSFW'15)*, pages 175–190, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society Press.

[BDNP99] M. Boreale, R. De Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *Logic in Computer Science*, pages 157–166, 1999.

[BG01] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 2. North Holland, 2001.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW'01)*, Cape Breton, Nova Scotia, Canada, June 2001. IEEE Computer Society Press.

[Bla02] B. Blanchet. From secrecy to authenticity in security protocols. In *9th International Static Analysis Symposium (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 242–259, Madrid, September 2002. Springer-Verlag.

[BLP03] L. Bozga, Y. Lakhnech, and M. Périn. Pattern-based abstraction for verifying secrecy in protocols. In *Tools and Agorithms for the Construction and Analysis of Sytems (TACAS'03), to appear*, 2003.

[BMPT90] R. Barbuti, P. Mancarella, D. Pedreshi, and F. Turini. A transformation approach to negation in logic programming. *Journal of Logic programming*, 8 :201–228, 1990.

[Bor01] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of the 28th Int. Coll. Automata, Languages, and Programming (ICALP'01)*, Crete, Greece, July 2001. Springer Verlag.

[BPW03] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003. http ://eprint.iacr.org/.

[CC01] H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. In *Research Report LSV-01-13*, december 2001.

[CCM01] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints and ping-pong protocols. In *Proceedings of the 28th Int. Coll. Automata, Languages, and Programming (ICALP'01)*, volume 2076, pages 682–693, Crete, Greece, July 2001. Springer Verlag.

[CD99] K. Compton and S. Dexter. Proof techniques for cryptographic protocols. In *Proceedings of the 26th International Colloquium on Automata, Languages, and Programming (ICALP'99)*, Prague, Czech Republic, July 1999.

[CDG$^+$97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on : `http ://www.grappa.univ-lille3.fr/tata`, 1997. release October, 1rst 2002.

[CDL$^+$99] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW'99)*, pages 55–69, Mordano, Italy, 1999. IEEE Computer Society Press.

[CJ97] J. Clark and J. Jacob. A survey of authentication protocol literature. `http ://www.cs.york.ac.uk/~jac/ papers/drareviewps.ps`, 1997.

[CKS01] R. Chadha, M.I. Kanovich, and A. Scedrov. Inductive methods and contract-signing protocols. In P. Samarati, editor, *8-th ACM Conference on Computer and Communications Security*, pages 176–185. ACM Press, November 2001.

[CL89] H. Comon and P. Lescanne. Equational Problems and Disunification. *Journal of Symbolic Computation*, 7 :371–425, 1989.

[CLC02] H. Comon-Lundh and V. Cortier. Security properties : two agents are sufficient. In *Research Report LSV-02-10*, August 2002.

[CLC03a] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. Technical Report LSV-03-3, Laboratoire Specification and Verification, ENS de Cachan, Cachan, France, January 2003. 30 pages.

[CLC03b] H. Comon-Lundh and V. Cortier. Security properties : two agents are sufficient. In *Proceedings of the 12th European Symposium On Programming (ESOP'03)*, volume 2618 of *Lecture Notes in Computer Science*, pages 99–113, Warsaw, Poland, April 2003. Springer Verlag.

[CLC03c] H. Comon-Lundh and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. In *Theoretical Computer Science, to appear*, 2003.

[CLS03] H. Comon-Lundh and V. Shmatikov. Constraint solving, exclusive or and the decision of confidentiality for security protocols assuming a bounded number of sessions. Technical Report LSV-02-3, Laboratoire Specification and Verification, ENS de Cachan, Cachan, France, January 2003. 13 pages.

[CMR01] V. Cortier, J. Millen, and H. Rueß. Proving secrecy is easy enough. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW'01)*, pages 97–108, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society Press.

[Coh00] E. Cohen. TAPS : A first-order verifier for cryptographic protocols. In *Proceedings of the 13th Computer Security Foundations Workshop (CSFW'00)*, Cambridge, England, 2000. IEEE Computer Society Press.

[Col82] A. Colmerauer. Prolog II. Manuel de référence et modèle théorique. Technical report, GIA Luminy, Marseille, Mars 1982.

[Col84] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS'84)*, pages 84–99. Tokyo, Japan, November 1984.

[Com88] H. Comon. *Unification et disunification. Théorie et applications*. PhD thesis, Institut National Polytechnique de Grenoble, mars 1988.

[Com91] H. Comon. Disunification : a survey. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic : Essays in Honor of Alan Robinson*. MIT Press, 1991.

[Com92] H. Comon. Constraints in term algebras. application to rewrite systems. *Atti Degli Incontri Di Logica Matematica*, 8 :5–17, 1992.

[Cor02a] V. Cortier. Outil de vérification SECURIFY. Rapport numéro 7 du projet RNTL EVA, May 2002. 6 pages.

[Cor02b] V. Cortier. Observational equivalence and trace equivalence in an extension of spi-calculus. application to cryptographic protocols analysis. In *Research Report LSV-02-3*, March 2002.

[CP97] W. Charatonik and A. Podelski. Set constraints with intersection. In *Logic in Computer Science*, pages 362–372, 1997.

[CS01] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not ? In *To appear in Journal of Telecommunications and Information Technology*. 2001.

[DEK83] D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. In R.L. Rivest, A. Sherman, and D. Chaum, editors, *Proceedings of CRYPTO 82*, pages 177–186, New York, 1983. Plenum Press.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 243–320. Elsevier,, 1990.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, Trento, Italia, 1999.

[DY81] D. Dolev and A.C. Yao. On the security of public key protocols. In *Proceedings of the 22nd Symp. on Foundations of ComputerScience*, pages 350–357, Nashville, Tennessee, USA, 1981. IEEE Computer Society Press.

[EFT96] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer Verlag, 1996.

[EG83] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Technical Report*. IEEE Computer Society Press, 1983.

[Eve99] Joris Evers. E-commerce encryption now vulnerable? *CNN*, August 30th 1999.

[FA01] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *Proceedings of the 14th Computer Seurity Foundation Workshop (CSFW'01)*, pages 160–173, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society Press.

[FLHT01] C.G. Fermüller, A. Leitsch, U. Hustadt, and T. Tammet. Resolution decision procedures. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 25, pages 1791–1852. Elsevier, 2001.

[GJM99] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology : Proceedings of Crypto'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 449–466. Springer Verlag, 1999.

[GK00] T. Genet and F. Klay. Rewriting for cryptographic protocol verification (extended version). In *Proceedings of the 17th International Conference on Automated Deduction (CAD'00)*, volume 1831 of *Lecture Notes in Artificial Intelligence*, Pittsburgh Pennsylvania, USA, January 2000. Springer Verlag.

[GL00] J. Goubault-Larrecq. A method for automatic cryptographic protocol verification. In *Proceedings of the 15 IPDPS 2000 Workshops*, volume 1800 of *Lecture Notes in Computer Science*, pages 977–984, Cancun, Mexico, May 2000. Springer Verlag.

[GL01] J. Goubault-Larrecq. Les syntaxes et la sémantique du langage de spécification EVA. In *Rapport numéro 3 du projet RNTL EVA*, November 2001.

[GLM97] J. Goubault-Larrecq and I. Mackie. *Proof Theory and Automated Deduction*, volume 6 of *Applied Logic Series*. Kluwer Academic, 1997.

[GLV02] J. Goubault-Larrecq and K. N. Verma. Alternating two-way ac-tree automata. In *Research Report LSV-02-11*, September 2002.

[GT01] J.D. Guttman and F.J. Thayer. Authentication tests and the structure of bundles. In *Theoretical Computer Science*. 2001.

[GTZ01] J.D. Guttman, F.J. Thayer, and L.D. Zuck. The faithfulness of abstract protocol analysis : message authentication. In *ACM Conference on Computer and Communications Security*, pages 186–195, 2001.

[HKT94] A. Herzberg, H. Krawczyk, and G. Tsudik. On travelling incognito. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, US, 1994.

[HLS00] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of the 13th Computer Security Foundations Workshop (CSFW'00)*, Cambridge, England, 2000. IEEE Computer Society Press.

[HS00] J. Heather and S. Schneider. Towards automatic verification of authentication protocols on an unbounded network. In *Proceedings of the 13th Computer Security*

Foundations Workshop (CSFW'00), pages 132–143, Cambridge, England, 2000. IEEE Computer Society Press.

[Hut02] H. Huttel. Deciding framed bisimulation. In *4th International Workshop on Verification of Infinite State Systems INFINITY'02*, pages 1–20, Brno, Czech Republic, 2002.

[JRV00] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Logic for Programming and Automated Reasoning (LPAR'00)*, volume 1955 of *Lecture Notes in Computer Science*, November 2000.

[KH69] R. Kowalski and P.J. Hayes. Semantic trees in automated theorem proving. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 87–101. Edinburgh University Press, 1969.

[KKR90] C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, pages 9–52, 1990.

[KR01] S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In K.G. Larsen and M. Nielsen, editors, *Concurrency Theory - Concur 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 551–565, Aalborg, Denmark, August 2001. Springer Verlag.

[KR02] S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In Steve Schneider, editor, *Proceedings of the 15th Computer Security Foundations Workshop (CSFW'02)*, pages 206–220, Cape Breton, Nova Scotia, Canada, June 2002. IEEE Computer Society Press.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055, pages 147–166, Passau, Germany, march 1996. Springer-Verlag, Berlin Germany. Also in Software Concepts and Tools, 17 :93-102, 1996.

[Low97a] G. Lowe. Casper : A compiler for the analysis of security protocols. In *Proceedings of 10th Computer Security Foundations Workshop (CSFW'97)*, Rockport, Massachusetts, USA, 1997. IEEE Computer Society Press. Also in Journal of Computer Security, Volume 6, pages 53-84, 1998.

[Low97b] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, Rockport, Massachusetts, USA, 1997. IEEE Computer Society Press.

[Low98] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proceedings of the 11th Computer Security Foundations Workshop (CSFW'98)*, Rockport, Massachusetts, USA, 1998. IEEE Computer Society Press.

[Low02] G. Lowe. Analysing Protocols Subject to Guessing Attacks. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS '02)*, 2002.

[LR97] G. Lowe and A.W. Roscoe. Using CSP to detect errors in the TMN protocol. In *IEEE transactions on Software Engineering*, volume 23, pages 659–669. 1997.

[Mea00a] C. Meadows. Invariant generation techniques in cryptographic protocol analysis. In *Proceedings of the 13th Computer Security Foundations Workshop (CSFW'00)*, Cambridge, England, 2000. IEEE Computer Society Press.

[Mea00b] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proceedings of "DISCEX" 2000*, pages 237–250. IEEE Comp. Society Press, 2000.

[Mer83] Michael J. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, February 1983.

[Mil93] R. Milner. The polyadic π-calculus. In F.L. Hamer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer-Verlag, 1993.

[MMS97] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murphi. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society Press, May 1997.

[Mon99] D. Monniaux. Abstracting cryptographic protocols with tree automata. In *Proceedings of the 6th International Static Analysis Symposium (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.

[Mou97] A. Mounji. *Languages and Tools for Rule-Based Distributed Intrusion Detection*. PhD thesis, Computer Science Institute, University of Namur, Belgium, September 1997.

[MPW92] R. Milner, J. Parrow, and David Walker. A calculus for mobile processes (parts i/ii). In *Information and Computation*, pages 1–40 and 41–77. Academic Press, September 1992.

[MR00] J. Millen and H. Rueß. Protocol-independent secrecy. In *Proceedings of the 21st Symposium on Research in Security and Privacy (RSP'00)*. IEEE Computer Society Press, 2000.

[MS01] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS'01)*, 2001.

[MSS98] J.C. Mitchell, V. Shmatikov, and U. Ster. Finite-state analysis of SSL 3.0. In *Proceedings of Computer Aided Verification (CAV'98)*, pages 71–76, 1998.

[MvOV97] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997.

[NGW96] P. Narendran, Q. Guo, and D.A. Wolfram. Unification and matching modulo nilpotence. In *Proceedings of the 13th International Conference on Automated Deduction, CADE'96*, volume 1104 of *Lecture Notes in Computer Science*, pages 261–274, 1996.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communication of the ACM*, 21(12) :993–999, 1978.

[Pau97a] L. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, 1997.

[Pau97b] L. Paulson. Relations between secrets : Two formal analyses of the Yahalom protocol. Technical Report TR432, University of Cambridge, Computer Laboratory, July 1997.

[Pau98] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1) :85–128, 1998.

[PW94] Birgit Pfitzmann and Michael Waidner. A general framework for formal notions of "secure" systems. Technical Report 11/94, Universität Hildesheim, April 1994.

[RB99] A.W. Roscoe and P.J. Broadfoot. Proving security protocols with model checkers by data independence techniques. In *Journal of Computer Security : Special Issue CSFW'12*, volume 7, pages 147–190. 1999.

[RDM82] N. Lynch R. Demillo and M. Merritt. Cryptographic protocols. In *Proceedings of the 14^{th} ACM SIGACT Symposium on Theory of Computing*. ACM, May 1982.

[RM00] H. Rueß and J. Millen. Local secrecy for stated-based models. In *Proceedings of the Workshop on Formal Methods in Computer Security (FMCS'2000)*, Chicago, Illinois, USA, 2000.

[RS98] P. Ryan and S. Schneider. An attack on a recursive authentication protocol : a cautionary tale. In *Information Processing Letters*, volume 65, pages 7–10. 1998.

[RSA78] R.L. Rivest, A. Shamir, and L.M. Adleman. A method for obtaining digital signatures and public key cryptosystems. In *Communications of the ACM*, volume 21, pages 120–126. ACM, February 1978.

[RSG$^+$00] P.Y.A. Ryan, S.A. Schneider, M.H. Goldsmith, G. Lowe, and A.W. Roscoe. *The modelling and analysis of security protocols : the CSP approach*. Addison-Wesley, 2000.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW'01)*, pages 174–190, Cape Breton, Nova Scotia, Canada, 2001. IEEE Computer Society Press.

[Sch96a] S.A. Schneider. Security properties and CSP. In *Proceedings of the Symposium on Security and Privacy*, pages 174–187, Oakland, 1996. IEEE Computer Society Press.

[Sch96b] B. Schneier. *Applied Cryptography Second Edition : protocols, algorithms, and source code in C*. J. Wiley & Sons, Inc., 1996.

[Sch97] S. Schneider. Verifying authentication protocols with CSP. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97)*, Rockport, Massachusetts, USA, 1997. IEEE Computer Society Press.

[SD95] U. Stern and D.L. Dill. Automatic verification of the SCI cache coherence protocol. In *Correct Hardware Design and Verification Methods : IFIP WG10.5 Advanced Research Working Conference Proceedings*, pages 21–34, 1995.

[SH02] V. Shmatikov and D.J.D Hughes. Defining Anonymity and Privacy (extended abstract). In *Proceedings of the Workshop on Issues in the Theory of Security (WITS '02)*, 2002.

[SM00a] V. Shmatikov and J. Mitchell. Analysis of a fair exchange protocol. In *Symposium on Network and Distributed Systems Security (NDSS '00)*, pages 119–128, San Diego, California, USA, February 2000. Internet Society.

[SM00b] V. Shmatikov and J. Mitchell. Analysis of abuse-free contract signing. In *Financial Cryptography '00*, volume 1962 of *Lecture Notes in Computer Science*, Anguilla, February 2000. Springer Verlag.

[SM01] V. Shmatikov and J. Mitchell. Finite-state analysis of two contract signing protocols. *Special issue of Theoretical Computer Science on security*, 2001. Accepted for publication.

[SMA95] D. Samfat, R. Molva, and N. Asokan. Untraceability in mobile networks. In *Mobile Computing and Networking*, pages 26–36, 1995.

[SMT94] D. Samfat, R. Molva, and G. Tsudik. Authentication of mobile users. Technical report, IEEE Network, Special Issue on Mobile Communications, March/April 1994.

[Son99] D. X. Song. Athena : A new efficient automatic checker for security protocol analysis. In *Proceedings of the 12th Computer Security Foundations Workshop (CSFW'99)*, Mordano, Italy, June 1999. IEEE Computer Society Press.

[SS96] S. Schneider and A. Sidiropoulos. CSP and anonymity. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, Lecture Notes in Computer Science. Springer Verlag, 1996.

[Sto01] S. Stoller. A bound on attacks on payment protocols. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 61–70, 2001.

[Sto02] S.D. Stoller. A bound on attacks on authentication protocols. In *Proc. 2nd IFIP International Conference on Theoretical Computer Science*, 2002.

[Syv94] P. Syverson. A Taxonomy of Replay Attacks. In *Proceedings of the 7th Computer Seurity Foundation Workshop (CSFW'94)*, IEEE, Computer Society Press, 1994.

[THG99] J. Thayer, J. Herzog, and J. Guttman. Strand spaces : proving security protocols correct. *IEEE Journal of Computer Security*, 7 :191–230, 1999.

[Wei98] C. Weidenbach. Sorted unification and tree automata. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction - A Basis for Applications*, volume 1, pages 291–320. Kluwer, 1998.

[Wei99] C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction, CADE'99*, volume 1632 of *Lecture Notes in Computer Science*, pages 378–382, 1999.

[Wei01] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 27. North Holland, 2001.

[WL93] T.Y.C. Woo and S.S. Lam. A semantic model for authentication protocols. In *IEEE Computer Society Symposium on Research in Security and Privacy (RSP'93)*, Oakland, CA, May 1993.

Index