UNIVERSITÉ DE MONTRÉAL

ASPiC: UN OUTIL D'ANALYSE SYMBOLIQUE AUTOMATISÉE DE PROTOCOLES CRYPTOGRAPHIQUES BASÉ SUR LE MODÈLE DE FLUX D'INFORMATION

GENEVIÈVE BASTIEN
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AOÛT 2004

© Geneviève Bastien, 2004.

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-97925-3
Our file Notre référence
ISBN: 0-612-97925-3

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

ASPiC: UN OUTIL D'ANALYSE SYMBOLIQUE AUTOMATISÉE DE PROTOCOLES CRYPTOGRAPHIQUES BASÉ SUR LE MODÈLE DE FLUX D'INFORMATION

présenté par: BASTIEN Geneviève

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. FERNANDEZ José, Ph.D., président

M. MULLINS John, Ph.D., membre et directeur de recherche

M. HAINS Gaétan, D.Phil., membre

REMERCIEMENTS

Je tiens à remercier John Mullins, mon directeur de thèse, pour avoir bien voulu partager ses connaissances théoriques, et surtout pour sa patience et pour m'avoir poussée au maximum de mes capacités. Je remercie spécialement Gaétan Hains pour son accueil au LIFO et son aide précieuse dans le développement d'ASPiC. Merci aussi à messieurs José Fernandez et Gaétan Hains pour leur participation au jury de ce mémoire. Un gros merci à Stéphane et Hamadou pour les nombreuses discussions et inspirations tout au long de ce projet et à Alaaeddine, qui s'est donné la peine de lire le mémoire en entier pour me conseiller sur la rédaction.

Sur une note plus personnelle, je remercie mes parents, Jean-Philippe et toute la famille pour leur soutien constant dans ces deux années de projet. Je remercie aussi mes amis pour les nombreuses sorties et divertissements nécessaires à une vie saine. Finalement, merci à la gang du Kung Fu qui m'ont permis de garder les deux pieds sur terre.

RÉSUMÉ

La montée en popularité d'Internet a occasionné le développement de plusieurs nouvelles applications de communication à distance. Certaines de ces applications requièrent un niveau de sécurité que l'infrastructure d'Internet ne fournit pas. L'utilisation de protocoles de communication, aidés des primitives cryptographiques appropriées est nécessaire pour garantir des propriétés telles l'authentification ou l'anonymat d'un usager, la confidentialité ou la non-répudiation d'un message, ou encore l'équité d'un échange. Une faille de sécurité dans ces protocoles peut avoir des conséquences désastreuses ; par exemple, un usager peut obtenir l'accès à de l'information confidentielle ou encore frauder un système.

Un problème auquel les chercheurs s'attaquent depuis des années est de prouver que les crypto-protocoles remplissent bien leur rôle et satisfont les propriétés désirées. Les méthodes formelles, plus spécifiquement le "model-checking", apportent une solution à ce problème. Le "model-checking" consiste à modéliser un protocole et une propriété et de faire la vérification sur ce modèle. À ce jour, il existe une panoplie de méthodes de "model-checking" et d'outils les implémentant pour vérifier une variété de plus en plus grande de protocoles. Toutefois, pour que la vérification soit automatisable, il faut que le modèle soit fini, ce qui oblige à des abstractions qui éloignent le protocole testé du protocole réel. Plus récemment, le développement de méthodes dites symboliques a permis de repousser les limites de la vérification pour englober des systèmes infinis. Il est maintenant possible de vérifier par "model-checking" des protocoles avec un nombre non-borné de sessions et de participants. Notre recherche se situe dans cette lignée.

Nous avons développé une algèbre de processus symbolique, *SPASM (Security Protocol Algebra for Symbolic Manipulations)*, pour modéliser un protocole cryptographique. C'est une algèbre réductionniste avec passage de valeurs qui regroupe toutes les transitions ne différant que par les valeurs de leurs variables en une seule transition

symbolique. Les agents du protocoles sont spécifiés dans cette algèbre. Ils sont ensuite synchronisés avec un intrus, système déductif correspondant au modèle de *Dolev-Yao*. Le système de transitions obtenu constitue le modèle du protocole.

Les propriétés à vérifier sont quant à elles spécifiées en termes de flux d'information. Plus spécifiquement, la propriété d'interférence admissible sert de base à l'expression de toutes les autres propriétés. Les actions des agents sont partitionnées selon le niveau de sécurité qu'elles représentent, soit haut niveau (information sensible), bas niveau (information publique) et déclassifiante (information sensible qu'on accepte de partager avec le public). L'interférence admissible dit qu'un observateur du bas niveau ne peut rien déduire de ce qui s'est produit au haut niveau à moins que ce ne soit permis par des actions déclassifiantes. Pour exprimer chacune des propriétés, il suffit de classer les actions au niveau de sécurité approprié selon son rôle dans le protocole. Des travaux antérieurs avaient déjà spécifié l'authentification, la confidentialité et l'anonymat. Nous ajoutons la spécification de la non-répudiation d'origine et de réception, ainsi que l'équité.

La vérification de l'interférence admissible exige la satisfaction d'une relation d'équivalence entre deux sous-ensembles du modèle initial : une bisimulation symbolique. Nous en avons défini et prouvé l'algorithme pour l'algèbre *SPASM*.

Nous proposons finalement un nouveau model-checker symbolique, *ASPiC* (*Analyse Symbolique de Protocoles Cryptographiques*), basé sur le modèle de flux d'information. Cet outil, programmé en langage CAML, vérifie l'authentification, la confidentialité, l'anonymat, la non-répudiation et l'équité à partir de la spécification d'un protocole de sécurité. Il a trouvé avec succès certains types d'attaques aux propriétés sur plusieurs crypto-protocoles. Il a aussi été utilisé pour vérifier des protocoles de commerce électronique tels *iKP* et une version abstraite de SET.

ABSTRACT

With the rapid growth of the Internet network came many new applications to allow distant communication. Some of them require a security level impossible to get from the Internet architecture alone. Mechanisms such as communication protocols, using appropriate cryptographic primitives, can help achieve properties like authentication or anonymity of a user, confidentiality or non-repudiation of a message or equity of an exchange. The consequences of a flaw in a security protocol can be disastrous. For example, a user might gain access to some confidential information or commit a fraud on a system.

To prove the correctness of a crypto-protocol and the satisfaction of its properties is a problem that researshers have been facing for years. Formal methods, especially model-checking, bring a solution to this problem. Model-checking requires to build a model of the system (protocol) on which a model of the property will be verified. There exists many model-checking methods and nearly as many tools implementing them and allowing the verification of an ever-growing class of protocols. But for the verification to be automatic, the model must be finite. Abstractions are thus necessary, creating a gap between the tested version and the original. Recently, development of symbolic methods have introduced the verification of infinite-state systems. It is now possible to model-check protocols with an unbounded number of sessions and principals. Our research follows this line.

We developped a symbolic process algebra, *SPASM (Security Protocol Algebra for Symbolic Manipulations)*, to model a crypto-protocol. This value-passing reductionnist algebra groups together all transitions differing only by the values of their variables into a unique symbolic transition. Agents specified with this algebra are then synchronised with an intruder, a lazy deductive system corresponding to the Dolev-Yao model. The transitions system thus obtained is the model of the protocol.

Properties to satisfy are then modeled in terms of information flow. More specifically,

admissible interference serves as a basis to express every other property. Actions of a system are partitionned according to their respective security level: high level (sensitive information), low level (public knowledge) and downgrading (sensible information acceptable to share with the public). Admissible interference is such that an observer observing the system from a low-level perspective cannot deduce what happened at a higher level unless it is through a downgrading action. To express each property, one only needs to classify the actions in the right security level according to its role in the protocol. Previous works already specified authentication, confidentiality and anonymity with admissible interference. We add the specification of non-repudiation of origin and reception, as well as equity of an exchange.

Verification of admissible interference requires the satisfaction of an equivalence relation between two subsystems of the original model. This relation is a symbolic bisimulation whose algorithm has been defined and proved for the *SPASM* algebra.

Finally, we propose a new information flow-based model-checker, *ASPiC (Symbolic Analysis of Crypto-Protocols)*. This tool, programmed in CAML, verifies authentication, confidentiality, anonymity, non-repudiation and equity from a SPASM specification of a security protocol. We were able to successfully find certain types of flaws on many known crypto-protocols. We also verified ecommerce protocols such as *iKP* and an abstracted version of SET.

TABLE DES MATIÈRES

REMERCIEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE DES MATIÈRES ix

LISTE DES FIGURES xiii

LISTE DES TABLEAUX xiv

LISTE DES ABBRÉVIATIONS ET DES SYMBOLES xv

INDEX DES NOTATIONS xvi

LISTE DES ANNEXES xvii

CHAPITRE 1 INTRODUCTION 1

1.1 Contexte 1

1.2 Cryptographie et crypto-protocoles 1

1.3 Propriétés de sécurité 3

1.4 Failles et attaques 4

1.5 Vérification des propriétés 7

1.6 Contenu du mémoire 9

CHAPITRE 2 MÉTHODES DE VÉRIFICATION 11

2.1 Theorem-proving 11

2.1.1 Logiques de la connaissance 12

2.1.2 Méthode inductive de Paulson avec *Isabelle/HOL* 16

2.1.3 Méthode de Bolignano avec *Coq* 19
2.1.4 Systèmes de réécriture de termes 20
2.1.4.1 Vérification avec *NRL* 21
2.1.4.2 Vérification avec *Timbuk* 22
2.1.4.3 Vérification avec *Hermes* 24
2.2 Model-checking 25
2.2.1 CCS 25
2.2.2 CSP et FDR 27
2.2.3 π-calcul, spi-calcul. 32
2.2.4 Logiques temporelles, CTL 33
2.2.5 Logiques alternantes et jeux 37
2.3 Autres méthodes formelles 39
2.3.1 Modèle "strand-space" 39
2.3.2 Les projets *AVISS* et *AVISPA* 42
2.4 Conclusion 44

CHAPITRE 3 MODÉLISATION DU PROTOCOLE 45
3.1 SPPA symbolique 45
3.2 Syntaxe de SPASM 47
3.2.1 Données 47
3.2.2 Fonctions privées et canaux 49
3.2.3 Processus 50
3.2.4 Agents 51
3.3 Valuations du système 52
3.4 Sémantique de SPASM 54
3.5 Modèle de l'intrus 57
3.6 Conclusion 59

CHAPITRE 4 SPÉCIFICATION DES PROPRIÉTÉS 60
4.1 Non-interférence et GNDC 60
4.2 Interférence admissible 65
4.3 Spécification des propriétés de sécurité 68
4.3.1 Authentification 69
4.3.2 Confidentialité et anonymat 71
4.3.3 Non-répudiation et équité 76
4.4 Conclusion 81

CHAPITRE 5 VÉRIFICATION 82
5.1 Bisimulation standard de SPPA 82
5.2 Bisimulation symbolique de SPASM 85
5.3 L'algorithme 89
5.4 Conclusion 93

CHAPITRE 6 ASPiC 94
6.1 Historique du *BNAI Analyzer* 94
6.2 Conception 96
6.2.1 Division en modules 97
6.2.2 Spécification des propriétés 98
6.2.3 Autres options 100
6.2.4 Abstractions et simplifications 100
6.2.5 Diagnostic d'erreur 101
6.3 Conclusion 104

CHAPITRE 7 ÉTUDES DE CAS 105
7.1 Protocoles simples 105
7.2 Protocoles de commerce électronique 108
7.2.1 Vérification du protocole 3KP 109

7.2.1.1 Abstraction de 3KP 109
7.2.1.2 Modèle SPASM de 3KP 110
7.2.1.3 Vérification des propriétés 113
7.2.1.4 Résultats 114
7.2.2 Résultats des protocoles testés 115
7.3 Limites et comparaison 116
7.4 Conclusion 118

CHAPITRE 8 CONCLUSION 119
8.1 Améliorations d'*ASPiC* 121
8.2 Perspectives 122

RÉFÉRENCES 124

INDEX 131

ANNEXES 132

LISTE DES FIGURES

FIGURE 2.1	Postulats de la logique de BAN	13
FIGURE 2.2	Protocole de Needham-Schroeder avec clé publique idéalisé	13
FIGURE 2.3	Sémantique opérationnelle de CCS	26
FIGURE 2.4	Exemples de processus CCS	27
FIGURE 2.5	Description de *Alice* en CSP	29
FIGURE 2.6	Description de *Bob* en CSP	29
FIGURE 2.7	Automate pour vérification avec CTL	34
FIGURE 2.8	Protocole de Needham-Schroeder exprimé en "strand-space"	41
FIGURE 2.9	Architecture du système AVISS	42
FIGURE 3.1	Sémantique opérationnelle de SPASM	56
FIGURE 3.2	Règles de déduction de l'intrus	57
FIGURE 4.1	Processus et non-interférence	66
FIGURE 5.1	Exemple de processus non-bisimilaires	83
FIGURE 5.2	Équivalence des sémantiques opérationnelles SPPA et SPASM	85
FIGURE 5.3	Graphes de transition SPASM	86
FIGURE 5.4	Algorithme de bisimulation symbolique de SPASM	91
FIGURE 6.1	Exemple de graphe de sortie de *ASPiC*	103
FIGURE 7.1	Abstraction du protocole 3kp	109
FIGURE 7.2	Description du client en SPASM	111
FIGURE 7.3	Description du marchand en SPASM	111
FIGURE 7.4	Description de la banque en SPASM	112
FIGURE 7.5	Spécification des requis de 3kp pour *ASPiC*	112

LISTE DES TABLEAUX

TABLEAU 7.1 Résultats sur protocoles simples (authentification et confidentialité) 106
TABLEAU 7.2 Résultats sur protocoles simples (non-répudiation) 107
TABLEAU 7.3 Résultats sur protocoles de ecommerce 115
TABLEAU 7.4 Comparaison avec d'autres approches 117

LISTE DES ABBRÉVIATIONS ET DES SYMBOLES

- AA : Automate alternant
- ASPiC : Analyse Symbolique de Protocoles Cryptographiques
- ATL : Alternating Temporal Logic
- BNAI : Bisimulation-based Non-deterministic Admissible Interference
- CC : Carte de crédit
- CCS : Calculus of Communicating Systems
- CRAC : Conception et Réalisation des Applications Complexes
- CSP : Communicating Sequential Process
- CTL : Computation Tree Logic
- FDR : Failure/Divergence/Refinement
- GGB : Grenouille à Grande Bouche
- NRO : Non-répudiation de l'origine
- NRR : Non-répudiation de la réception
- NSPK : Needham-Schroeder Public Key Protocol
- SPASM : Security Protocol Algebra for Symbolic Manipulations
- SPPA : Security Protocol Process Algebra
- SRT : Système de réécriture de termes
- ST : Système de transitions
- TTP : Trusted third party (tiers parti honnête)

INDEX DES NOTATIONS

$\mathcal{A}, \mathcal{V}, \mathcal{N}, \mathcal{T}$	ensembles des agents, variables, entiers, termes	47
$\mathcal{M}$	ensemble des termes clos	47
$\gamma(t)$	fonction de décomposition	48
$\gamma^{-1}(t)$	composition de termes	48
λ	fonction publique	49
λ_{gen}	fonction génératrice	49
φ	valuation d'un système de transitions symbolique	52
$\varphi[x \mapsto t]$	assignation d'un terme à une variable	52
$[\![x]\!]_\varphi$	terme clos associé à x avec variables évaluées par φ	52
$\mathcal{O}$	critère d'observation	67
$\simeq_\mathcal{O}$	équivalence observationnelle	67
ρ	valuation d'un système non-symbolique	83
$\rho \models \varphi$	correspondance de valuations symbolique et non-symbolique	84
$\_ \stackrel{\alpha}{\Longrightarrow} \_$	trace visible d'une transition	84
$\mathcal{R}^{\rho,\varrho}$	relation symétrique paramétrée sur ρ, ϱ	84
$\mathcal{B}^{\varphi,\psi}$	relation symétrique paramétrée sur φ, ψ	87

LISTE DES ANNEXES

ANNEXE I CRYPTO-PROTOCOLES 132

ANNEXE II PREUVES DE PROPOSITIONS 144

CHAPITRE 1

INTRODUCTION

1.1 Contexte

S'il est une autoroute dans ce monde qui n'éprouve pas de problèmes pour son développement, c'est définitivement l'autoroute virtuelle Internet. Ces dernières années, son débit a explosé, tout comme la variété de ses utilisations. Elle permet de relier en quelques secondes tout au plus deux points quelconques de la planète. Par contre, les véhicules qui y circulent échappent à tout contrôle policier ; elle est donc un paradis pour les mal-intentionnés. Pour cette raison, les utilisateurs honnêtes de ce réseau exigent des garanties de sécurité lorsqu'ils échangent certains types d'information, par exemple, un numéro de compte pour l'achat d'un bien électronique. Afin d'obtenir ces garanties de sécurité, on utilise la cryptographie et les crypto-protocoles.

1.2 Cryptographie et crypto-protocoles

La cryptographie est un ensemble de techniques algorithmiques qui fournit la sécurité de l'information. Elle permet de transformer un message clair en un message incompréhensible qui peut ainsi traverser un réseau non sûr comme Internet à l'abri des utilisateurs malhonnêtes(Guillem-Lessard, 2002). Le chiffrement et le hachage de données sont deux techniques cryptographiques.

Le chiffrement consiste à utiliser une clé pour encrypter un message qui pourra par la suite être décrypté uniquement par un utilisateur connaissant la clé de déchiffrement

correspondante. Il existe deux types de systèmes de chiffrement : à clé symétrique et à clé publique. On note $\{m\}_k$ le message m encrypté avec la clé k.

Les systèmes de chiffrement à clé symétrique sont ceux où la clé de décryption est facilement déductible à partir de la clé d'encryption et vice versa. Dans la plupart des cas, ces deux clés sont les mêmes.

Les systèmes de chiffrement à clé publique, aussi appelés à clés asymétriques, impliquent une paire de clés (publique / privée). La clé publique est connue de tous tandis que la clé privée est propriété de la personne à qui appartient la paire. Un message encrypté avec une clé publique n'est déchiffrable que par l'usager possédant la clé privée correspondante. De même, ce système de chiffrement permet la signature numérique : un message (ou une empreinte du message, par exemple un hash) est encrypté avec la clé privée d'un usager. Le destinataire le déchiffre avec la clé publique correspondante et est ainsi assuré de l'identité de l'expéditeur.

Quant au hachage, il consiste à exécuter une fonction à sens unique sur le message pour obtenir une empreinte de ce dernier. Cette empreinte est de taille fixe et est telle que toute altération au message entraîne une modification de son empreinte.

Cependant, ces primitives cryptographiques ont peu de sens à elles seules. Par exemple, la réception par un usager d'un message encrypté ne veut rien dire pour lui s'il ne sait pas qui l'a encrypté, quand et dans quel but. Les crypto-protocoles donnent un sens aux messages échangés. Un protocole est une séquence d'étapes de communication et de règles pour échanger de l'information(Glossary,). Un protocole se déroule habituellement entre deux participants ou acteurs : un initiateur, appelé par convention *Alice*, et un répondeur, appelé *Bob*. Parfois, d'autres participants tels des serveurs participent à l'échange. L'envoi d'un message m par *Alice* vers *Bob* dans un protocole est noté

$$Alice \rightarrow Bob \ : \ m$$

L'exécution entre deux participants de la séquence d'étapes du protocoles est appelée une session.

Les crypto-protocoles sont une classe de protocoles utilisant les primitives cryptographiques pour atteindre des objectifs de sécurité. Les protocoles d'authentification, d'échange de clés et de commerce électronique entrent dans cette catégorie. L'annexe I présente quelques exemples de crypto-protocoles dont il sera mention dans le présent travail.

1.3 Propriétés de sécurité

Les crypto-protocoles garantissent, par l'échange d'un petit nombre de messages bien formés, certaines propriétés de sécurité, autrement impossibles à assurer. En voici quelques exemples :

- l'**authentification** consiste à prouver qu'un participant au protocole est bien celui qu'il prétend être et non un intrus se donnant une fausse identité. L'authentification peut être à sens unique (un seul des participants doit s'authentifier auprès de l'autre) ou mutuelle (les deux participants doivent s'authentifier l'un auprès de l'autre)(Burrows et al., 1990).
- la **confidentialité** restreint la connaissance d'une quelconque information aux seuls participants du protocole ou encore à quelques participants seulement(Asokan et al., 1997).
- l'**anonymat** cache à un parti l'identité de son interlocuteur. Cette notion peut s'étendre vers la **non-traçabilité** qui empêche de lier deux transactions au même parti(Asokan et al., 1997).
- la **non-répudiation** empêche un parti de nier être la source d'un message et permet de le prouver à un juge externe. La non-répudiation se divise en deux : la non-

répudiation de l'origine (NRO) par laquelle un participant ne peut nier être à l'origine d'un message et la non-répudiation de la réception (NRR) par laquelle un participant ne peut nier avoir reçu un certain message. L'**équité** stipule que soit les deux participants ont une preuve que l'autre ne peut répudier, soit aucun ne la reçoit(Zhou and Gollmann, 1997).

- l'**atomicité des biens** garantit que si un paiement est effectué, la marchandise sera livrée. L'**atomicité de l'argent** garantit que si le client est débité un certain montant, alors le marchand sera crédité ce même montant, il n'y a ni destruction ni création d'argent(Heintze et al., 1996).
- la **disponibilité** garantit un accès au service et la **fiabilité** prévoit un recouvrement juste en cas de panne(Asokan et al., 1997).
- L'**intégrité** assure qu'aucun argent ne sera débité du client ou crédité au marchand sans leur autorisation respective(Asokan et al., 1997).

Les crypto-protocoles sont conçus pour satisfaire à une ou plusieurs de ces propriétés.

1.4 Failles et attaques

La principale difficulté est de développer des protocoles qui satisfont aux propriétés pour lesquelles ils sont conçus. Car bien que correct sur papier, une fois en service, un crypto-protocole subit les assauts d'un *intrus*, c'est-à-dire de toute entité malhonnête agissant sur le protocole. Dans ces circonstances, on dit un protocole contient une faille s'il ne peut répondre à une des propriétés désirées.

Par exemple, prenons le protocole d'authentification de Needham-Schroeder avec clé publique (NSPK) suivant(Needham and Schroeder, 1978) :

1. Alice $\rightarrow$ Bob : $\{N_a, Alice\}_{PK_{Bob}}$
2. Bob $\rightarrow$ Alice : $\{N_a, N_b\}_{PK_{Alice}}$
3. Alice $\rightarrow$ Bob : $\{N_b\}_{PK_{Bob}}$

Ce protocole assure l'authentification mutuelle d'*Alice* et de *Bob* par un échange de nonces N_a et N_b. La présence de N_a dans le message 2 reçu par *Alice* authentifie *Bob* alors que la présence de N_b dans le message 3 reçu par *Bob* authentifie *Alice*. Suite à l'authentification, les deux nonces N_a et N_b peuvent être utilisés comme authentificateur pour les communications subséquentes entre *Alice* et *Bob*. Plusieurs années après sa mise en fonction, on a trouvé qu'il existe un moyen d'attaquer ce protocole et créer une faille d'authentification lorsque *Alice* désire communiquer avec l'intrus (Lowe, 1996).

1. Alice $\rightarrow$ E : $\{N_a, Alice\}_{K_E}$
2. E(Alice) $\rightarrow$ Bob : $\{N_a, Alice\}_{K_{Bob}}$
3. Bob $\rightarrow$ E(Alice) : $\{N_a, N_b\}_{K_{Alice}}$
4. E $\rightarrow$ Alice : $\{N_a, N_b\}_{K_{Alice}}$
5. Alice $\rightarrow$ E : $\{N_b\}_{K_E}$
6. E(Alice) $\rightarrow$ Bob : $\{N_b\}_{K_{Bob}}$

L'intrus utilise le nonce envoyé par *Alice* pour amorcer en son nom une communication avec *Bob*. Cette attaque est due au fait que nulle part dans cet échange de message, *Alice* ne mentionne avec qui elle a l'intention de communiquer. Ceci est un exemple d'attaque *man-in-the-middle* où l'intrus s'interpose entre les deux participants et change l'intention du premier. Il existe plusieurs types d'attaques, classées selon le degré d'activité de l'intrus :

- Analyse de trafic : L'intrus analyse les messages qui passent sur le réseau pour en tirer une information confidentielle, par exemple, un numéro de compte ou un NIP.

- Inférence : L'intrus déduit à partir de certains messages, une information autrement non disponible. On y compte les attaques de type *dictionnaire* qui consistent à déduire une valeur d'un domaine borné, par exemple un mot de passe de 8 caractères.
- Répétition : L'intrus rejoue les messages qu'il voit passer sur le réseau. Par exemple, il peut tenter d'encaisser le montant d'une transaction qui ne lui est pas dédié ou payer une marchandise avec la facture d'un autre client. Il peut faire des attaques miroirs, c'est-à-dire rejouer à un participant un message reçu de ce même participant. Cette attaque peut être contrée par des stratégies qui permettent à un participant de reconnaître ses propres messages. L'intrus peut aussi rejouer un message lors d'une session ultérieure. Ce problème est contré par l'utilisation de valeurs (nonces, estampilles temporelles) garantissant la fraîcheur d'un message.
- Modification d'un message : L'intrus modifie le contenu de certains champs d'un message, par exemple en remplaçant le nom du destinataire par le sien. On met dans cette catégorie les attaques de types par lesquelles un intrus envoie pour un champ une valeur de type différent de celui attendu.
- Mascarade : L'intrus se fait passer pour un participant honnête qui n'est pas lui. Il ne possède aucune donnée personelle sur ce participant. C'est plutôt des attaques du type "man-in-the-middle". Par exemple, il peut se faire passer pour un autre participant auprès d'une institution financière et ainsi retirer de l'argent en son nom.
- Fraude : L'intrus est un participant légitime, mais malhonnête, du protocole. Un client malhonnête peut tenter de payer moins cher pour le bien qu'il demande, dépenser deux fois l'argent comptant qu'il possède. Les cas de clés compromises, que l'intrus peut utiliser à sa guise, et de serveurs corrompus entrent dans cette catégorie.

La conception d'un protocole sans faille est une tâche ardue. Parfois, des attaques très

subtiles sont indétectables en observant le protocole, mais elles existent. (Clark and Jacob, 1997) font un survol des protocoles d'authentification et des failles rencontrées. Rares sont ceux qui n'en contiennent pas. Pourtant, les conséquences de telles failles peuvent être désastreuses du point de vue économique, surtout si le protocole est utilisé à grande échelle.

1.5 Vérification des propriétés

Ceci convainc de la nécessité de trouver une méthode de vérification qui permette d'assurer qu'un protocole est sûr, même (et surtout) en présence d'un environnement hostile. Il existe plusieurs méthodes de vérification de protocoles. Parmi les plus simples, on compte les tests unitaires ou de boîte noire qui consistent simplement à exécuter le protocole avec des données précises. Mais puisque le domaine des données en entrée est souvent infini, ces tests ne peuvent tester toutes les possibilités. Or, les conséquences d'une faille dans un crypto-protocole peuvent être désastreuses. On ne peut donc se contenter de vérifier une partie du système et assumer que le tout serait correct. Ça prend des méthodes plus strictes. À l'autre extrémité du spectre des méthodes de vérification, on trouve les méthodes formelles. Celles-ci sont beaucoup mieux adaptées à la vérification des systèmes critiques.

Les méthodes formelles incluent toutes les applications des mathématiques discrètes aux différentes phases du développement logiciel. Elles impliquent habituellement la modélisation puis l'analyse d'un système, toutes deux basées sur des fondations mathématiques précises(Vienneau, 1996). Il existe deux grandes catégories de méthodes formelles de spécification et vérification, soit le "theorem-proving" et le "model-checking".

Le "theorem-proving" consiste à faire la preuve, à l'aide de méthodes d'inférence

appropriées, de la validité d'une propriété, appelée théorème. Le "theorem-proving" permet de faire une preuve complète qu'un protocole est correct, ie d'aller jusqu'à en garantir l'absence de failles, peu importe le nombre de sessions, parallèles ou séquentielles, et en présence d'un intrus. Toutefois, les langages utilisés pour décrire les protocoles font souvent appel à des logiques d'ordre supérieur, ce qui élimine la décidabilité de la preuve. L'automatisation de ces techniques est donc partielle et requiert beaucoup d'interaction de la part de l'usager pour réduire la taille du problème à une dimension vérifiable automatiquement(Burrows et al., 1990; Paulson, 1998; Bolignano, 1996).

Le "model-checking" consiste d'abord à modéliser un protocole, souvent sous la forme d'un système de transitions ou d'une machine à états, puis de modéliser la propriété à vérifier. On tente finalement de montrer qu'il y a équivalence entre les deux modèles. Si c'est le cas, la propriété est vérifiée, sinon, on trouve un contre-exemple. Les spécifications du modèle et de la propriété sont indépendantes l'une de l'autre ; on peut combiner les différentes approches selon ses besoins spécifiques. La complétude et la terminaison de ces méthodes sont garanties, ce qui permet d'implanter des outils pour lesquels l'usager n'a qu'à connaître un langage de spécification et le tour est joué ! Cependant, le "model-checking" ne s'attaque qu'aux systèmes à états finis, ce qui oblige à ne considérer qu'un nombre borné d'acteurs et de sessions du protocole. De plus, le problème de l'explosion de l'espace-état lors de la modélisation (un seul envoi de message occasionne des dizaines de transitions puisque l'on doit considérer toutes les possibilités de valeurs échangées) limite encore plus la taille des systèmes vérifiables(Lowe, 1996; Heintze et al., 1996; Panti et al., 2002).

Ces dernières années, le développement d'approches dites symboliques a permis de repousser ces limites et de gérer des systèmes à états infinis par "model-checking". Ces méthodes explorent directement un modèle infini en manipulant des contraintes qui peuvent représenter un ensemble infini d'états. Elles réduisent le problème d'ex-

plosion d'espace-état, tout en vérifiant des protocoles dans un environnement plus large(Guttman, 2001; de Alfaro et al., 2001).

Par contre, la difficulté des méthodes de "theorem-proving" et de "model-checking", symboliques ou non, décourage leur utilisation. Celles qui connaissent un certain succès pour la vérification des crypto-protocoles sont celles qui ont été implantées dans des outils effectuant la vérification sans que l'usager ait à comprendre le mécanisme sous-jacent. Notons entre autres FDR(FDR, 2000), MOCHA(Alur et al., 1998a) et CASRUL(Bouallagui et al., 2002).

1.6 Contenu du mémoire

Le présent projet de recherche a été réalisé au laboratoire de Conception et Réalisation des Applications Complexes (CRAC). Le CRAC s'intéresse au développement et à l'application des méthodes formelles en ingénierie des systèmes complexes. Les principaux axes de recherche sont la spécification du code mobile sécurisé, la conception sécurisée de logiciels directement avec UML, les méthodes de vérification de flux d'information sécurisés basées sur le "model-checking" et l'analyse des protocoles cryptographiques et de commerce électronique. Nous nous intéresserons aux deux derniers thèmes.

Les travaux du CRAC sur la vérification par flux d'information ont atteint une maturité qui lui permet de se hisser parmi les méthodes efficaces de vérification de crypto-protocoles. D'abord comptant parmi les méthodes de model-checking standard, elle a évolué pour devenir symbolique. Toutefois, à ce jour, aucun outil ne la mettait en valeur.

Ce mémoire se présente en trois parties :

- Un modèle de spécification : Nous proposons un modèle de protocoles basé sur une algèbre de processus symbolique réductionniste, *SPASM (Security Protocol Algebra for Symbolic Manipulations)*. C'est une algèbre de processus avec passage de valeurs qui permet de passer d'un système de transitions à états infinis à un système de transitions fini en regroupant des transitions ne différant que par les valeurs des paramètres échangés. Les agents du protocole à vérifier sont mis en parallèle avec un intrus correspondant au modèle de *Dolev-Yao*, sur lequel nous reviendrons plus loin (chapitre 3).
- Une méthode de vérification : Nous utilisons les flux d'information pour exprimer les différentes propriétés de sécurité à vérifier. L'authentification, la confidentialité et l'anonymat ont déjà été définies dans des travaux précédents (Lafrance and Mullins, 2002; Brlek et al., 2003). Nous reprenons cette méthodologie pour ajouter la non-répudiation et l'équité (chapitre 4). Leur vérification consiste en la satisfaction d'une relation de bisimulation symbolique entre deux systèmes de transitions, relation qui devra être définie pour l'algèbre développée et prouvée équivalente à une bisimulation entre deux systèmes non-symboliques (chapitre 5).
- L'outil de vérification : Nous présentons l'implantation de l'outil *ASPiC*[1] *(Analyse Symbolique de Protocoles Cryptographiques)* développé dans le cadre de ce projet (chapitre 6). Cet outil est un prototype de "model-checker" implantant la méthode décrite dans les chapitres 3 à 5 et vise à montrer son applicabilité à la vérification pratique des propriétés des crypto-protocoles. Nous exposons les résultats des tests faits sur divers exemples au chapitre 7.

Mais d'abord, le chapitre suivant présente une revue des méthodes formelles de vérification utilisées par différents auteurs pour l'analyse des crypto-protocoles.

[1] Le nom d'aspic est inspiré de celui d'une vipère, dont la morsure peut s'avérer mortelle.

CHAPITRE 2

MÉTHODES DE VÉRIFICATION

Les sections qui suivent présentent différentes méthodes formelles de vérification décrites dans la littérature et utilisées pour la vérification des crypto-protocoles. D'abord, nous présentons quelques méthodes de "theorem-proving" utilisées, puis quelques-unes de "model-checking". Nous distinguons finalement une troisième catégorie de méthodes dites hybrides que l'on ne peut classer dans aucune des deux catégories précédentes.

Chaque section a la structure suivante : nous décrivons brièvement la méthode puis l'illustrons par un exemple (NSPK). Nous terminons en présentant des analyses exécutées sur des crypto-protocoles, surtout des protocoles de ecommerce. Ce chapitre se veut un état de l'art des méthodes de vérification formelles et n'est en rien un pré-requis à la compréhension du reste de ce mémoire.

2.1 Theorem-proving

Les méthodes de "theorem-proving" permettent de faire une preuve complète de la correction d'un protocole, ie d'aller jusqu'à en garantir l'absence de failles, peu importe le nombre de sessions et en présence d'un intrus. Les sections qui suivent présentent quelques méthodes qui entrent dans cette catégorie : d'abord des méthodes tout-usage, puis d'autres adaptées spécialement à la vérification de protocoles.

2.1.1 Logiques de la connaissance

Les logiques de la connaissance, dont la mère est la logique de BAN, du nom de leurs inventeurs Burrows, Abadi et Needham (Burrows et al., 1990) datent de la fin des années 80. Ces logiques sont le résultat d'un effort pour formaliser un raisonnement informel sur les protocoles d'authentification. On y définit un ensemble de prédicats permettant d'exprimer les croyances et les connaissances des participants, de même que des postulats pour raisonner sur ceux-ci. La vérification consiste à s'assurer que, partant des croyances initiales et avec les messages appris en cours d'exécution, il est possible, à l'aide des postulats, d'atteindre les croyances finales souhaitées.

Nous présentons ici un sous-ensemble des prédicats de la logique de BAN(Burrows et al., 1990). Un message envoyé dans le protocole est un prédicat. Les lettres Q et R représentent d'autres prédicats.

- A *believes* Q : A croit que Q est vrai.
- A *sees* Q : Quelqu'un a envoyé le message Q à A.
- A *said* Q : A a, à un moment quelconque passé, envoyé le message Q.
- $fresh$(Q) : Q est une valeur fraîche, donc n'a pas été envoyé précédemment dans l'exécution du protocole.
- $\overset{K}{\mapsto}$ A : A a la clé publique K, donc sa clé privée K^{-1} ne pourra être découverte par aucun autre principal.
- A $\overset{Q}{\leftrightarrow}$ B : A et B partagent le secret Q avec peut-être une autre entité fiable.
- $\{Q\}_K$: Q est encrypté par la clé K.
- $\langle Q \rangle_R$: Q est combiné avec R où R est un secret, partagé entre deux acteurs, dont la présence prouve l'identité du principal qui émet $\langle Q \rangle_R$.

La figure 2.1 présente quelques postulats pour raisonner sur les énoncés. Informellement, les règles signifient :

- signification de messages : Permet de dériver de l'information sur l'origine d'un message. Si un message est accompagné d'un secret partagé entre A et B, alors A

signification de messages $\dfrac{A \textbf{ believes } A \overset{R}{\rightleftharpoons} B, A \textbf{ sees } \langle Q \rangle_R}{A \textbf{ believes } B \textbf{ said } Q}$

vérification de nonces $\dfrac{A \textbf{ believes fresh}(Q), A \textbf{ believes } B \textbf{ said } Q}{A \textbf{ believes } B \textbf{ believes } Q}$

vision1 $\dfrac{A \textbf{ sees } \langle Q \rangle_R}{A \textbf{ sees } Q}$

vision2 $\dfrac{A \textbf{ believes } A \overset{K}{\mapsto} A, A \textbf{ sees } \{Q\}_K}{A \textbf{ sees } Q}$

fraîcheur $\dfrac{A \textbf{ believes fresh}(Q)}{A \textbf{ believes fresh}(A \overset{Q}{\rightleftharpoons} B)}$

FIGURE 2.1 Postulats de la logique de BAN

1. Alice $\rightarrow$ Bob : $\{N_a\}_{PK_{Bob}}$
2. Bob $\rightarrow$ Alice : $\{\langle Alice \overset{N_b}{\rightleftharpoons} Bob \rangle_{N_a}\}_{PK_{Alice}}$
3. Alice $\rightarrow$ Bob : $\{\langle Alice \overset{N_b}{\rightleftharpoons} Bob \rangle_{N_a}\}_{PK_{Bob}}$

FIGURE 2.2 Protocole de Needham-Schroeder avec clé publique idéalisé

est en droit de croire qu'il vient de B.

- vérification de nonces : Si un nonce est récent, alors l'expéditeur y croit encore.
- vision1 et vision2 : Si A voit un message, alors il en voit les composantes que ce soit un message combiné avec un secret ou un message encrypté avec sa clé publique.
- fraîcheur : Si une partie d'un message est fraîche, alors le message complet est frais.

Exemple. Pour illustrer cette méthode, soit le protocole NSPK. L'authentification se fait par l'échange de nonces qui demeureront des secrets partagés par *Alice* et *Bob* et identifieront l'émetteur lors des communications subséquentes.

La figure 2.2 montre la version *idéalisée* du protocole, ie où chaque donnée est représentée selon son rôle dans le protocole. Les valeurs importantes sont les deux

nonces à échanger. D'abord, *Alice* envoie à *Bob* un nonce fraîchement généré encrypté avec sa clé publique. Ce nonce n'a, à ce moment, aucune signification. Si *Bob* décide d'aller de l'avant dans la session, il produit aussi un nonce qu'il envoie à *Alice*. Les deux acteurs sont donc prêts pour amorcer une communication. À partir de ce point, les nonces jouent un rôle de secrets partagés. La présence de N_a dans le deuxième message est donc signe que le deuxième nonce N_b, se veut un secret entre *Alice* et *Bob*. De même, dans le troisième message, la présence de N_b confirme que N_a est un secret partagé, puisque si les acteurs n'y avaient pas cru, les messages n'auraient pas été envoyés.

Nous sommes maintenant prêts à analyser le protocole. D'abord nous identifions les croyances initiales. Celles-ci concernent les clés publiques, la fraîcheur des nonces, ainsi que la croyance de chaque acteur que le nonce qu'il a lui-même généré est un secret avec l'autre acteur.

a. $Alice$ **believes** $\overset{PK_{Alice}}{\mapsto} Alice$ b. Bob **believes** $\overset{PK_{Alice}}{\mapsto} Alice$

c. $Alice$ **believes** $\overset{PK_{Bob}}{\mapsto} Bob$ d. Bob **believes** $\overset{PK_{Bob}}{\mapsto} Bob$

e. $Alice$ **believes fresh**(N_a) f. Bob **believes fresh**(N_b)

g. $Alice$ **believes** $Alice \overset{N_a}{\rightleftharpoons} Bob$ h. Bob **believes** $Alice \overset{N_b}{\rightleftharpoons} Bob$

Quant aux objectifs finaux à obtenir, on remarque que *Alice* et *Bob* veulent tous deux la certitude que le nonce généré par son interlocuteur l'a bien été dans le but de servir de secret. Donc, nous avons comme postulats à vérifier,

c1. $Alice$ **believes** Bob **believes** $Alice \overset{N_b}{\rightleftharpoons} Bob$

c2. Bob **believes** $Alice$ **believes** $Alice \overset{N_a}{\rightleftharpoons} Bob$

La réception du premier message ajoute aux croyances initiales

$$i.\ Bob \textbf{ sees } \{N_a\}K_{Bob}$$

Cet énoncé en conjonction avec d et en utilisant le postulat vision2 permet de déduire

$$j.\ Bob\ \textbf{sees}\ N_a$$

C'est tout ce que nous pouvons tirer du premier message. Le deuxième message additionne l'énoncé

$$k.\ Alice\ \textbf{sees}\ \{\langle Alice \overset{N_b}{\rightleftharpoons} Bob\rangle_{N_a}\}_{PK_{Alice}}$$

k et a, après application de vision2 permet de déduire

$$l.\ Alice\ \textbf{sees}\ \langle Alice \overset{N_b}{\rightleftharpoons} Bob\rangle_{N_a}$$

et par vision1,

$$m.\ Alice\ \textbf{sees}\ Alice \overset{N_b}{\rightleftharpoons} Bob$$

Ce dernier, par signification de messages et g nous dit que

$$n.\ Alice\ \textbf{believes}\ Bob\ \textbf{said}\ Alice \overset{N_b}{\rightleftharpoons} Bob$$

Par e et fraîcheur appliqué à plusieurs reprises,

$$o.\ Alice\ \textbf{believes fresh}\ Alice \overset{N_b}{\rightleftharpoons} Bob$$

o et n et vérification de nonce permettent d'arriver à $c1$.

et ainsi de suite jusqu'à ce qu'on ne puisse plus déduire quoi que ce soit avec ce message. On passe alors au troisième message et on pose un raisonnement similaire, obtenant ainsi les croyances finales. À ce point, les deux principaux sont en présence d'un secret qu'ils croient que l'autre acteur va accepter comme étant partagé par eux seuls. *Alice* et *Bob* peuvent continuer à s'échanger des messages en utilisant N_a et N_b comme preuve d'identité.

Or, nous savons aujourd'hui qu'il existe une erreur dans ce protocole. La logique de BAN a échoué dans la détection de celle-ci dû à la faiblesse de certains postulats. ◁

La logique initiale de Burrows, Abadi et Needham ne fonctionne que pour une certaine catégorie de protocoles d'authentification. Plusieurs extensions ont été développées pour augmenter le pouvoir de raisonnement de la logique, mais souvent au prix d'une plus grande complexité(Gong et al., 1990; Abadi and Tuttle, 1991; van Oorschot, 1993; Syverson and van Oorschot, 1994).

Quelques travaux de vérification de protocoles de commerce électronique ont été exécutés avec les logiques BAN et ses extensions. (Anderson, 1999) a utilisé la logique de BAN originale pour prouver le bon fonctionnement du système UEPS (annexe I.9) sauf dans le cas d'un déni de service permanent où le compte du client aura été débité sans que celui du marchand n'ait été crédité. (Zhou and Gollmann, 1998) ont utilisé une extension de BAN pour vérifier la non-répudiation d'un protocole simple, tandis que (Kailar, 1996) va plus loin en développant sa propre logique permettant de mieux raisonner sur la non-répudiation dans le commerce électronique. Il utilise avec succès sa logique sur 4 protocoles, entre autres le Internet Billing Server de Carnegie-Mellon (annexe I.8) pour lequel il montre que la non-répudiation n'est atteinte qu'advenant certaines assomptions et des étapes supplémentaires. (Kessler and Neumann, 1998) ont utilisé une logique semblable pour étudier la non-répudiation de SET (annexe I.7) et montrer que ce requis est satisfait.

2.1.2 Méthode inductive de Paulson avec *Isabelle/HOL*

Cette méthode, développée par (Paulson, 1998), utilise l'induction sur les règles pour raisonner sur un protocole. Les protocoles sont représentés comme un ensemble de toutes les traces possibles. Chaque message d'un protocole est traduit sous la forme

d'une règle qui décrit comment une trace *tr* peut être étendue par un des participants.

Un des acteurs est un attaquant qui peut voir tout le trafic sur le réseau, mascarader pour des acteurs honnêtes, de même que personnifier des acteurs compromis, dont il connaît les clés privées. Cet intrus peut créer et déduire des nouveaux messages selon les règles d'inférence du modèle de Dolev-Yao[1](Dolev and Yao, 1983).

En plus des règles propres au protocole, trois autres règles indépendantes sont nécessaires pour en compléter la modélisation.

1. La trace vide [] est une trace.
2. Règle de génération de faux messages : Si *tr* est une trace, *m* peut être déduit ou créé par l'intrus, et *Bob* est un agent différent de *Intrus*, alors la trace *tr* peut être étendue avec l'événement "*Intrus sends m to Bob*".
3. Règle de l'accident : Si *tr* est une trace, et N_a et N_b sont des nonces qui devraient être des secrets entre *Alice* et *Bob*, alors la trace *tr* peut être étendue avec l'événement "*Intrus learns* (N_a, N_b)". Cette règle modélise la perte d'un secret, d'une manière ou d'une autre, non due à une attaque sur le protocole.

La preuve d'une propriété se fait par induction sur les règles. La propriété F est d'abord définie en termes de traces et de messages. Par exemple, la confidentialité de la valeur v peut s'exprimer v *ne peut être déduite par l'intrus.* Le principe d'induction stipule que $F(tr)$ est préservé pour chaque trace *tr* pour autant que F soit préservé par toutes les règles permettant de créer des traces. On doit prouver $F([])$, puis, pour chaque règle, il faut montrer que $F(tr) \Longrightarrow F(ev\#tr)$ où $ev\#tr$ est la concaténation à la trace *tr* de l'événement *ev*.

Exemple. Les règles pour représenter le protocole NSPK sont les suivantes :

[1]Ce modèle présente l'intrus comme un système déductif qui utilise l'hypothèse du chiffrement parfait, ie que l'intrus peut encrypter et décrypter n'importe quel message qu'il connaît à condition d'avoir la clé

1. Si *tr* est une trace, N_a un nonce frais et *Bob* un agent distinct d'*Alice*, alors la trace *tr* peut être étendue avec l'événement "*Alice sends* $(\{N_a, Alice\}_{PK_{Bob}})$ *to Bob*".
2. Si la trace *tr* contient un événement de la forme "*Alice' sends* $(\{N_a, Alice\}_{PK_{Bob}})$ *to Bob*", N_b est un nonce frais et *Alice* est un agent distinct de *Bob*, alors la trace *tr* peut être étendue avec l'événement "*Bob sends* $(\{N_a, N_b\}_{PK_{Alice}})$ *to Alice*".
3. Si *tr* est une trace contenant deux événements "*Alice sends* $(\{N_a, Alice\}_{PK_{Bob}})$ *to Bob*", "*Bob' sends* $(\{N_a, N_b\}_{PK_{Alice}})$ *to Alice*", alors la trace *tr* peut être étendue avec l'événement "*Alice sends* $(\{N_b\}_{PK_{Bob}})$ *to Bob*".

Les agents *Alice'* et *Bob'* signifient qu'on ne sait pas exactement qui a envoyé le message. Pour prouver l'authentification de *Alice* par *Bob*, il suffit de prouver que N_b demeure secret. Or, la tentative de prouver le secret échoue, laissant une trace contenant un événement passé de la forme "*Alice sends* $(\{N_a, Alice\}_{PK_{Intrus}})$ *to Intrus*" et ainsi de suite pour recréer l'attaque de Lowe. ◁

Paulson a utilisé le "theorem-prover" tout-usage Isabelle/HOL pour vérifier le protocole NSPK, de même que des crypto-protocoles à clé symétrique, dont celui de Otway-Rees (annexe I.5).

D'autres auteurs ont analysé des protocoles de ecommerce. (Bella et al., 2002b) ont vérifié le sous-protocole d'enregistrement de SET (annexe I.7.1). Ils en ont prouvé 64 théorèmes pour le client et 31 pour le marchand, entre autres "le secret de la carte est confidentiel si le client envoie sa demande de certification à une autorité de certification non compromise". Dans (Bella et al., 2002a), ils se sont aussi intéressés à certaines propriétés du protocole de paiement de SET.

Toutefois, les preuves faites avec cette méthode sont longues et complexes. Un protocole peut prendre plusieurs jours à spécifier et demande une expertise de la part de l'usager qui ne rend pas cette méthode accessible à n'importe qui.

2.1.3 Méthode de Bolignano avec *Coq*

(Bolignano, 1996) se penche plutôt sur savoir quelles garanties l'on peut tirer à partir d'hypothèses de confiance. Il distingue les agents honnêtes, qui jouent le rôle prévu dans le protocole, et le monde extérieur ou intrus. Le protocole est modélisé comme un automate où chaque état est représenté par les valeurs des variables et les transitions sont les étapes du protocole. La transmission de messages est asynchrone, donc l'envoi et la réception se font en deux temps. Les propriétés, exprimées soit par des fonctions de filtration des étapes (par exemple, on peut abstraire tous les messages qui ne sont pas des échanges entre des acteurs *Alice* et *Bob* et vérifier que l'ordre des actions est bien celui attendu), soit en termes d'invariants (par exemple, une clé privée demeure toujours secrète), sont vérifiées par induction sur les états du protocole.

Exemple. Le protocole de NSPK est spécifié en 6 transitions dont les états initial et final sont décrits par les valeurs des variables des participants : $Alice : N_a, N_b, vers$, $Bob : N_a, N_b, de$. L'état s_I d'un intrus est spécifié comme l'ensemble des connaissances qu'il a acquises sur le réseau. Les variables primées représentent leur valeur après la transition. Le premier message est exprimé comme suit

1a. *Alice* envoie $msg1$ et ($Alice.N'_a$ = valeur fraîche) et
($s'_I = s_I \cup \{Alice.N'_a, Alice\}_{PK_{Bob}}$)

1b. *Bob* reçoit $msg1$ et ($\{Bob.N'_a, Bob.de\}_{PK_{Bob}}$ déductible de s_I)

et ainsi de suite pour les autres messages du protocole.

Pour exprimer l'authentification de *Alice* par *Bob*, on considère uniquement les transitions des agents honnêtes. L'authentification s'exprime en disant que chaque fois qu'*Alice* complète un cycle $1a, 2a, 3a$, alors *Bob* a exécuté un cycle $1b, 2b$ possiblement suivi de $3b$, avec les valeurs des variables respectivement cohérentes avec l'autre participant. De plus, l'ordre des actions doit être $1a, 1b, 2b, 2a, 3a, 3b$. Ensuite, des invariants se rajoutent à cette propriété, soit "si $Alice.vers \neq Bob$ alors

$Alice.N_b \neq Bob.N_b$". Cet invariant n'est pas vérifié dans le cas de l'attaque de Lowe. $\lhd$

(Bolignano, 1997) a prouvé, par sa méthode aidée du "theorem-prover" tout-usage Coq, quelques propriétés de sécurité du protocole C-SET, protocole basé sur des cartes à puces proposé par le Groupe des Cartes Bancaires français (annexe I.7.3). Son approche a de plus permis d'exprimer des propriétés de protocoles tels iKP (annexe I.12) et Millicent(annexe I.13).

La difficulté de cette méthode réside dans la spécification des propriétés qui est loin d'être intuitive. Chaque propriété est souvent un ensemble de fonctions de filtration et d'invariants, à vérifier indépendamment ou en conjonction les uns avec les autres. On obtient la version finale d'une propriété par raffinement et non par spécification directe.

2.1.4 Systèmes de réécriture de termes

Alors que les méthodes de "theorem-proving" précédentes adaptaient des approches tout-usage à l'analyse des crypto-protocoles, les méthodes suivantes sont conçues spécifiquement à cette fin.

D'abord, chacune spécifie le protocole sous une forme ou une autre d'un système de réécriture de termes (SRT). Un SRT est un système de réduction où chaque terme peut être transformé selon un ensemble fini de règles de réécriture. On dit qu'un terme g est *réduit* en un terme d lorsqu'il existe une règle de réécriture $g \longrightarrow d$ où chaque variable se retrouvant dans d est aussi dans g. $g \longrightarrow *d$ signifie que g peut être réduit à d en une étape ou plus.

Exemple. Le SRT suivant spécifie le protocole NSPK. Le symbole $\vee$ concatène les termes et le LHS représente le membre de gauche de la règle.

- $goal(x, y) \rightarrow LHS \vee msg(x, y, \{N(x, y), x\}_{PK(y)})$ correspond au premier message du protocole. x et y sont des agents et $msg(x, y, enc(pk(y), x, N(x, y)))$ est l'envoi de x vers y d'un nonce $N(x, y)$ (créé par x pour communiquer avec y) et de l'identificateur de x, encrypté avec la clé publique de y, $pk(y)$. $goal(x, y)$ est la demande de communication entre x et y.
- $msg(x, y, \{na, x2\}_{PK(y)}) \rightarrow LHS \vee msg(y, x2, \{na, N(y, x2)\}_{PK(x2)})$. Si un agent y reçoit un message encrypté avec sa clé publique et contenant un nonce et l'identité d'un agent, il renvoie sur le réseau le deuxième message du protocole, après avoir créé un nonce pour communiquer avec $x2$.
- $msg(x, y, \{N(y, z), nb\}_{PK(y)}) \rightarrow LHS \vee msg(y, z, \{nb\}_{PK(z)}) \vee a\_auth(y, z)$. À la réception du 2e message avec le nonce qu'il a créé, l'agent y envoie le 3e message. La dernière action rapporte que l'agent y croit avoir authentifié l'agent z.
- $msg(x, y, \{N(y, z)\}_{PK(y)}) \rightarrow LHS \vee b\_auth(y, z)$. À la réception du 3e message avec le nonce qu'il a créé, l'agent y rapporte qu'il croit avoir authentifié l'agent z.

Les connaissances de l'intrus sont aussi décrites par des règles de réécriture, par exemple pour décrypter un message :

$$\{y\}_{PK(Intrus)} \rightarrow LHS \vee y$$

◁

Plusieurs méthodes d'analyse ont été développées à partir de spécifications SRT. (Delzanno and Ganty, 2003) fait un survol de quelques-unes de ces méthodes.

2.1.4.1 Vérification avec *NRL*

Catherine Meadows a développé une méthode basée sur les règles de Prolog et implantée dans l'analyseur de protocoles NRL (Meadows, 1996b; Meadows, 1999). L'in-

trus contrôle le SRT et essaie d'y résoudre un problème de mots.

Pour effectuer la vérification, l'usager doit d'abord décrire un état insécure en fonction de ce que peut connaître l'intrus. L'analyseur fait ensuite une recherche arrière pour retrouver les états qui ont permis d'atteindre l'état spécifié. Si tous les chemins qui y mènent partent d'un état impossible, alors l'état insécure est non-atteignable. Dans le cas contraire, on a trouvé une attaque sur le protocole.

Exemple. (Meadows, 1996a) ont réussi à retrouver l'attaque de Lowe sur le protocole NSPK. Une fois le protocole spécifié, l'analyseur de protocole NRL doit déterminer comment atteindre un état dans lequel "un récepteur *Bob* accepte un nonce N_a comme venant d'un initiateur *Alice* à qui il répond par un nonce N_b lorsque (1) *Alice* n'a pas initié de session avec *Bob* en utilisant N_a et (2) N_b n'est pas compromis". $\triangleleft$

L'analyseur de protocoles NRL a permis de vérifier le protocole SET(Meadows and Syverson, 1998). Ici, les propriétés étaient exprimées à l'aide d'un logique temporelle, appelée NPATRL, spécifique à l'analyseur de protocole NRL. Les résultats obtenus de cette spécification ne sont toutefois pas mentionnés.

2.1.4.2 Vérification avec *Timbuk*

(Genet and Klay, 2000) utilisent des automates d'arbres pour représenter un protocole associé avec un SRT. Ensuite, ils dérivent un automate d'approximation pour vérifier la concordance avec le SRT $\mathcal{R}$ décrivant le protocole.

Soit E un ensemble de termes reconnaissables par le SRT initial. L'ensemble des configurations atteignables $\mathcal{R}^*(E)$ est l'ensemble des termes générés par clôture transitive de l'application de $\mathcal{R}$ aux termes de E.

On définit le langage accepté par un automate $\mathcal{A}$ comme

$\mathcal{L}(\mathcal{A}) = \{t | \exists q$ tel que $t \rightarrow *q$ permis par l'automate$\}$

L'automate d'approximation $\mathcal{T}_{\mathcal{R}} \uparrow (\mathcal{A})$ est obtenu à partir de $\mathcal{A}_0$ en construisant récursivement des automates $\mathcal{A}_i$ tels que pour chaque i, $\mathcal{L}(\mathcal{A}_i) \subset \mathcal{L}(\mathcal{A}_{i+1})$, jusqu'à ce qu'on atteigne un $\mathcal{A}_k$ tel que $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^*(E)$, l'automate d'approximation recherché. À chaque étape, on construit $\mathcal{A}_{i+1}$ à partir de $\mathcal{A}_i$ en trouvant un terme $s \in \mathcal{L}(\mathcal{A}_i)$ tel qu'il existe une règle $s \rightarrow t$ où $t \notin \mathcal{L}(\mathcal{A}_i)$ et en ajoutant les règles nécessaires pour obtenir un automate $\mathcal{A}_{i+1}$ tel que $t \in \mathcal{L}(\mathcal{A}_{i+1})$.

Exemple. Alors que le SRT utilise des variables pour décrire les règles, l'automate instancie ces variables. Par exemple le membre de gauche de la première règle du SRT de NSPK, $goal(x, y)$, devient dans l'automate des termes du genre $goal(Alice, Bob)$, $goal(Alice, Intrus)$, etc.

Pour illustrer la technique d'approximation, nous prenons un exemple plus simple. Soit le SRT avec l'unique règle $f(g(x)) \rightarrow g(f(x))$, où f et g sont des fonctions. Et soit l'automate $\mathcal{A}_i$ avec la règle $f(g(q)) \rightarrow q'$, avec q et q' des états. Cette situation suggère d'ajouter la règle $g(f(q)) \rightarrow q'$ à l'automate $\mathcal{A}_{i+1}$. L'idée est que puisque $f(g(q))$ représente plusieurs états $f(g(t))$ pour un t accepté via l'état q, alors parmi les états accessibles, il peut y en avoir qui sont aussi acceptés par la règle $g(f(q)) \rightarrow q'$. $\mathcal{A}_{i+1}$ est donc une sur-approximation du SRT. $\triangleleft$

La vérification du protocole se fait ensuite en montrant que l'automate d'approximation obtenu, $\mathcal{T}_{\mathcal{R}} \uparrow (\mathcal{A})$, ne contient aucun mauvais comportement. Ceci peut être vérifié en utilisant un autre automate d'arbre $\mathcal{A}_{bad}$ contenant les comportements insécures et en s'assurant que l'intersection des langages acceptés par les deux est vide. Si $\mathcal{L}(\mathcal{T}_{\mathcal{R}} \uparrow (\mathcal{A})) \cap \mathcal{L}(\mathcal{A}_{bad}) = \emptyset$ alors aucun état insécure n'est atteignable par le protocole.

L'outil *Timbuk* implante cette méthode(Genet and Tong, 2001).

2.1.4.3 Vérification avec *Hermes*

(Bozga et al., 2003) utilisent de leur côté une technique basée sur des patrons. Ils spécifient le protocole sous forme de SRT. Ensuite, pour définir la propriété à vérifier, ils utilisent des patrons $\langle \mathcal{G}, \mathcal{B} \rangle$ où $\mathcal{G}$ est l'ensemble des bons patrons (comportements sécures) et $\mathcal{B}$ l'ensemble des mauvais comportements.

Exemple. Soit $E = \langle \mathcal{G}, \mathcal{B} \rangle$ où $\mathcal{G}$ contient le singleton $\{\{x_s\}_k\}$ et $\mathcal{B}$ contient le singleton $\{\{A, x_s\}_k\}$ où l'indice s représente une position où un secret peut être présent. Ceci signifie que E consiste en tous les termes de la forme $\{m\}_k$, mais où m n'est pas de la forme (A, m'). $\lhd$

L'algorithme de vérification consiste en une recherche arrière à partir des termes pour essayer de trouver des mauvais termes. Cet algorithme part de secrets et essaie de déterminer quels mauvais termes auraient pu les révéler et quels autres termes auraient dû être gardés secrets pour ne pas que les secrets initiaux soient révélés. Si ces secrets supplémentaires ne peuvent être des secrets, par exemple parce que l'intrus aurait pu facilement le créer, alors une attaque a été trouvée.

Exemple. Le protocole NSPK débute avec les secrets $\{N_b, SK(Alice), SK(Bob)\}$, aucun mauvais patron et l'ensemble de bons patrons $\mathcal{G} = \{\{x_s\}_{PK(Alice)}, \{x_s\}_{PK(Bob)}\}$. L'application de l'algorithme à ce protocole requiert l'ajout à la liste des secrets du premier message, soit $\{N_a, Alice\}_{PK(E)}$. Comme ce message ne contient aucun secret, ce ne peut être un secret. C'est plutôt le signe d'une attaque. $\lhd$

Cette approche a été implantée dans l'outil *Hermes*.

2.2 Model-checking

Le model-cheking regroupe des méthodes qui consistent d'abord à modéliser le protocole à analyser, ensuite à spécifier la propriété à vérifier et enfin parcourir le modèle pour le montrer correct ou trouver un contre-exemple à la propriété. Les sections qui suivent présentent les duos modèle/logique, la plupart implantés dans des outils, qui ont été utilisés avec succès pour la vérification des cryptoprotocoles.

2.2.1 CCS

Les algèbres de processus sont des méthodes assez naturelles de modéliser les processus concurrents qui communiquent entre eux. Elles ont été conçues pour étudier le comportement de processus parallèles et concevoir des systèmes informatiques. Les protocoles étant des systèmes informatiques impliquant la communication entre plusieurs agents indépendants, ils peuvent être modélisés par des algèbres de processus où chaque agent est un processus(Cleaveland and Smolka., 1999).

Cette section présente l'une des premières algèbres de processus, CCS (Calculus of Communicating Systems)(Milner, 1989) pour les systèmes communicants tels les protocoles. Elle servira de véhicule pour décrire ce qu'est une algèbre de processus et quels éléments la composent.

Soit un ensemble d'actions visibles Σ et une action interne τ. L'ensemble de toutes les actions est noté Act.

Les processus sont définis par la syntaxe suivante :

$$P \quad := \quad 0 \quad | \quad \alpha.P \quad | \quad P_1 + P_2 \quad | \quad Z = P \quad | \quad P_1 | P_2 \quad | \quad P \backslash L \quad | \quad P[f]$$

De manière informelle, 0 est le processus qui ne prend aucune transition ; $\alpha.P$ est le

PRÉFIXAGE	$\frac{-}{\alpha.P \xrightarrow{\alpha} P}$	
ALTERNATIVE	$\frac{P_1 \xrightarrow{\alpha} P_1'}{P_1+P_2 \xrightarrow{\alpha} P_1'}$	$\frac{P_2 \xrightarrow{\alpha} P_2'}{P_1+P_2 \xrightarrow{\alpha} P_2'}$
DÉFINITION	$\frac{P \xrightarrow{\alpha} P' \; Z=P}{Z \xrightarrow{\alpha} P_1'}$	
PARALLÈLE	$\frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1' \mid P_2}$	$\frac{P_2 \xrightarrow{\alpha} P_2'}{P_1 \mid P_2 \xrightarrow{\alpha} P_1 \mid P_2'}$
SYNCHRONISATION	$\frac{P_1 \xrightarrow{\alpha} P_1' \; P_2 \xrightarrow{\overline{\alpha}} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$	
MASQUAGE	$\frac{P \xrightarrow{\alpha} P' \; \alpha \notin L}{P \backslash L \xrightarrow{\alpha} P' \backslash L}$	
RENOMMAGE	$\frac{P \xrightarrow{\alpha} P' \; \beta=f(\alpha)}{P[f] \xrightarrow{\beta} P'[f]}$	

FIGURE 2.3 Sémantique opérationnelle de CCS

processus qui après avoir exécuté l'action $\alpha \in Act$ se comporte comme P ; $P_1 + P_2$ est le choix non-déterministe de se comporter comme P_1 ou P_2 ; $Z = P$ définit le nom Z comme le processus se comportant à la manière de P ; $P_1|P_2$ est la composition parallèle asynchrone de P_1 et P_2 ; $P\backslash L$ interdit les actions de L ; $P[f]$ renomme les actions selon la fonction $f : Act \rightarrow Act$, par exemple $f(\alpha) = \beta$ remplace les occurences de α dans P par l'action β.

De façon plus formelle, la figure 2.3 présente la sémantique opérationnelle permettant d'obtenir le système de transitions (ST) associé à un processus. Cette sémantique se présente sous la forme $\frac{premisse}{conclusion}$ qui signifie que le système se comporte comme *conclusion* si les *premisses* sont satisfaites.

Exemple. La figure 2.4 présente des exemples de processus CCS. ◁

Tout formalisme basé sur les algèbres de processus inclut une relation d'équivalence comportementale ($\simeq$), ou d'ordonnancement ($\sqsubseteq$) dans certains cas, qui permet de comparer deux processus entre eux(Cleaveland and Smolka., 1999). Cette relation est

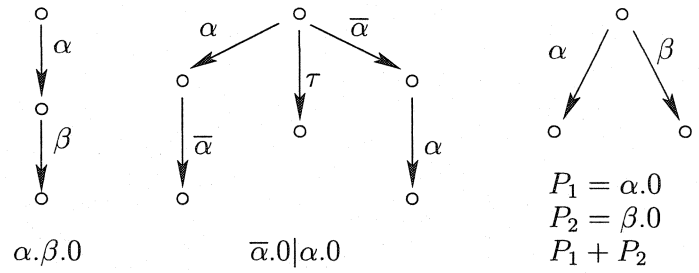


FIGURE 2.4 Exemples de processus CCS

utilisée pour la vérification des protocoles, qui revient souvent à décider de l'équivalence entre deux systèmes, par exemple une spécification et une implémentation ($spec \simeq imp$). Par contre, CCS est encore trop peu expressif pour exprimer adéquatement des protocoles de sécurité complexes. Mais il a servi de base au développement d'une panoplie d'algèbres de processus mieux adaptées à la tâche.

2.2.2 CSP et FDR

Une autre algèbre de processus amplement utilisée dans la vérification de protocoles est CSP(Communicating Sequential Processes)(Hoare, 1985), indissociable de l'outil qui l'implémente, FDR (Failure/Divergence/Refinement). La sémantique de CSP étant basée sur le même principe que CCS, elle ne sera pas détaillée ici. Nous présentons tout de même une partie de la syntaxe qui sera utile à la compréhension de l'exemple(FDR, 2000).

Un processus CSP est défini comme suit :

$$P \quad := \quad STOP \quad | \quad c\,?x\,!y \rightarrow P \quad | \quad P_1|P_2 \quad | \quad \text{if } b \text{ then } P_1 \text{ else } P_2$$

où $STOP$ est le processus où aucune transition n'est possible. $c\,?x\,!y$ est un préfixe au processus P où c est un canal, x représente une valeur reçue sur le canal et y est

une valeur envoyée. c, x et y peuvent être d'arité variable ; dans ce cas, on sépare les différentes valeurs par des "." (par exemple, $?x_1.x_2$ représente la réception de deux variables x_1 et x_2). $P_1|P_2$ est la composition parallèle asynchrone de P_1 et P_2. if b then P_1 else P_2 représente la conditionnelle telle que si l'expression booléenne b est vraie, alors le processus se comporte comme P_1, sinon, il se comporte comme P_2.

FDR représente les protocoles selon 3 modèles(FDR, 2000). Pour chacun de ces modèles on définit un raffinement $\sqsubseteq_x$ permettant d'ordonner les processus P et Q :

- le *modèle de traces* représente les processus par des séquences finies d'événements possibles, incluant la trace vide. Le raffinement correspondant est le raffinement de traces $P \sqsubseteq_T Q \Leftrightarrow traces(Q) \subseteq traces(P)$. Ce modèle est le plus simple et suffit à la vérification de la plupart des propriétés de sécurité.
- le *modèle d'échecs* représente les processus par leurs traces et leurs échecs. Les échecs sont les paires (tr, δ) où tr est une trace à la fin de laquelle les événements de l'ensemble δ sont impossibles à exécuter. Le raffinement correspondant est $P \sqsubseteq_E Q \Leftrightarrow echecs(Q) \subseteq echecs(P)$. Ce modèle est donc sensible aux blocages du système.
- le *modèle d'échecs-divergences* représente les processus par leurs échecs et leurs divergences. Les divergences sont les traces à la suite desquelles le processus peut exécuter une séquence infinie d'actions internes. Le raffinement correspondant est $P \sqsubseteq_D Q \Leftrightarrow echecs(Q) \subseteq echecs(P) \wedge divergences(Q) \subseteq divergences(P)$.

FDR et le modèle de traces ont permis de vérifier plusieurs propriétés intéressantes des crypto-protocoles. L'outil génère un modèle uniquement en fonction de la syntaxe entrée. Il appartient donc au concepteur de prévoir tout comportement déviant du système ou attaque de la part d'un intrus. Par exemple, la description d'*Alice* dans le protocole NSPK doit prévoir la possibilité où les messages 1 et 3 sont interceptés et le message 2 est généré par l'intrus. De la même façon, il faut expliciter le comportement fraudeur d'un client dans un protocole de commerce électronique. Cette façon de faire est propice aux erreurs du concepteur et ne garantit pas la précision du modèle en

$Alice(a, n_a) = user.a?b \rightarrow A\_running.a.b \rightarrow$
$comm!msg1.a.b.Encrypt.pk(b).n_a.a \rightarrow$
$comm.msg2.b.a.Encrypt.pk(a)?n'_a.n_b \rightarrow$
if $n_a = n'_a$ then $comm!msg3.a.b.Encrypt.pk(b).n_b \rightarrow$
$A\_commit.a.b \rightarrow STOP$
else $STOP$

FIGURE 2.5 Description de *Alice* en CSP

$Bob(b, n_b) = comm.msg1.a.b.Encrypt.pk(b)?n_a.a' \rightarrow$
$B\_running.a'.b \rightarrow comm!msg2.a'.b.Encrypt.pk(a').n_a.n_b \rightarrow$
$comm.msg3.a.b.Encrypt.pk(b)?n'_b \rightarrow$
if $n_b = n'_b$ then $B\_commit.a'.b \rightarrow STOP$
else $STOP$

FIGURE 2.6 Description de *Bob* en CSP

présence d'un environnement hostile.

Le protocole ainsi spécifié constitue l'implémentation IMP. La propriété désirée est spécifiée de la même façon, en CSP et constitue la spécification SPEC. Celle-ci décrit un comportement idéal du protocole tel qu'il s'exécuterait sans intrus ni fraude. On vérifie maintenant la satisfaction de la propriété : le protocole IMP rencontre la spécification SPEC si c'en est un raffinement, donc IMP $\models$ SPEC $\Leftrightarrow$ SPEC $\sqsubseteq$IMP.

Exemple. C'est grâce à cette méthode que Lowe a découvert la faille du protocole NSPK(Lowe, 1996). Les figures 2.5 et 2.6 présentent les spécifications des agents *Alice* et *Bob* respectivement. De plus, un intrus est défini avec les capacités d'intercepter des messages et en créer de nouveaux. La synchronisation des agents et de l'intrus constitue l'implémentation, IMP.

La propriété d'authentification est spécifiée comme suit : Soit la séquence d'événements à observer

$$S_0 = A\_running.a.b \rightarrow B\_commit.a.b \rightarrow S_0$$

La spécification de l'authentification SPEC est $S_0|Run(\Sigma')$, où $Run(\Sigma')$ représente

l'exécution aléatoire de tous les événements possibles de Σ sauf $A\_running.a.b$ et $B\_commit.a.b$. Donc l'événement par lequel *Bob* authentifie *Alice* est toujours précédé de l'événement par lequel *Alice* amorce une session avec *Bob* , peu importe le déroulement des autres événements.

On vérifie ensuite si l'implantation est un raffinement de la spécification, SPEC$\sqsubseteq$IMP. Il s'avère que non puisque la trace $A\_running.a.i \rightarrow B\_commit.a.b$, possible dans IMP ne fait pas partie des traces permises dans SPEC. $\lhd$

Lowe a utilisé une autre stratégie pour trouver une faille d'authentification dans le protocole TMN(Lowe and Roscoe, 1997). Il s'agit d'ajouter à l'intrus des événements pour marquer son succès à se faire authentifier en tant qu'*Alice* auprès de *Bob* et s'assurer que cet événement ne fait pas partie des traces de IMP.

Schneider s'est attardé sur la spécification de l'anonymat(Schneider and Sidiropoulos, 1996) et de la non-répudiation(Schneider, 1998) en CSP.

Pour spécifier l'anonymat, soit une fonction de renommage $f : Act \rightarrow Act$. On définit une fonction

$$f_A(x) = \begin{cases} \alpha & \text{si } x \in A \\ x & \text{sinon} \end{cases}$$

où $A \subseteq Act$ est l'ensemble des événements à garder anonymes et $\alpha \notin Act$ est un événement bidon. Un processus est anonyme si $f_A^{-1}(f_A(P)) = P$, donc $f_A^{-1}(f_A(P)) \sqsubseteq P$ et $P \sqsubseteq f_A^{-1}(f_A(P))$. Intuitivement, si tous les événements de A étaient renommés en un seul événement α (ce qui est le cas dans $f_A(P)$), alors lorsque cet événement α se produit, n'importe quel événement de A aurait pu se produire (spécifié par $f_A^{-1}(f_A(P))$). Cette propriété a été testée avec succès sur un exemple simple de don anonyme.

Quant à la non-répudiation, sa spécification est un peu plus complexe. L'étude faite

par (Schneider, 1998) concernait le protocole de Zhou et Gollmann avec tiers-parti (annexe I.6.1). Il faut ajouter un acteur, soit un juge qui ne sait pas a priori quels agents sont honnêtes ou non. On assume qu'un message signé par *Alice* garantit que *Alice* a envoyé le message. La NRO se résume au fait que si le juge reçoit de *Bob* un message signé d'*Alice* correspondant à la preuve d'origine du message, alors *Alice* a envoyé ce message. Pour la NRR par contre, ce n'est pas aussi simple puisque le message à transmettre à *Bob* passe d'abord par un TTP et donc *Alice* reçoit sa preuve de réception alors qu'elle n'est pas encore certaine que *Bob* ait reçu le message. On sait toutefois que si *Alice* reçoit sa preuve, alors le message est disponible pour *Bob*, même s'il ne le reçoit pas. Le raffinement de traces n'est plus suffisant ici, on utilise plutôt le raffinement d'échecs, qui s'assure que si *Alice* présente au juge une preuve de réception (par le TTP), alors *Bob* a reçu ou recevra le message. De même, pour vérifier l'équité de la non-répudiation, on utilise le raffinement par échecs pour s'assurer qu'il ne se produit pas de situation dans laquelle un des deux partis a reçu sa preuve et l'autre n'a plus la possibilité de recevoir la sienne.

FDR a aussi permis la vérification d'autres propriétés. (Heintze et al., 1996) ont pu vérifier le respect de l'atomicité de l'argent et des biens pour NetBill et DigiCash (annexes I.10 et I.11). Ces propriétés sont spécifiées respectivement telles que, dans une même trace, "soit le client est débité ET le marchand est crédité ou aucun ne l'est", et "un marchand reçoit un paiement si et seulement si le client reçoit le bien". On ne considère qu'une seule exécution du protocole avec un client, un marchand et une banque, aucun intrus, la fraude venant d'un des acteurs. Ils ont trouvé que NetBill satisfait les deux spécifications tandis que DigiCash ne satisfait pas l'atomicité de l'argent en cas de fraude de la part du client, si celui-ci essaie de double-dépenser une pièce. (Lu and Smolka, 1999) ont vérifié quelques propriétés d'une abstraction du protocole SET (annexe I.7.2). Ils ont prouvé que le marchand ne peut charger plus cher ou charger plus d'une fois le client, tandis qu'un client ne peut payer moins

cher ou ne peut utiliser le système s'il n'en a pas la permission. Aussi, en présence d'acteurs honnêtes, le marchand recevra éventuellement son paiement.

FDR est un "model-checker" tout usage permettant d'exprimer n'importe quel protocole. Toutefois, la syntaxe de CSP est assez limitée lorsqu'il s'agit de crypto-protocoles puisque les fonctions telles l'encryption et la signature doivent être gérées manuellement lors de la spécification. Pareillement, l'intrus ou un comportement frauduleux doivent être pris en compte par le concepteur, rendant cette étape propice à l'erreur. Toutefois, la syntaxe permet beaucoup de souplesse autant dans la spécification du protocole que des propriétés. On peut y exprimer quasiment n'importe quel comportement et on peut adapter la conception à la propriété à vérifier. Par contre, les propriétés dépendent directement du protocole. Chaque étude de cas est différente et unique et varie même d'un concepteur à l'autre, ce qui n'est pas idéal lors d'une vérification formelle...

2.2.3 π-calcul, spi-calcul.

Le π-calcul bat en expresivité les algèbres de processus précédentes en traitant de façon similaire les canaux et les données. *Alice* peut donc envoyer un canal en paramètre à *Bob* et ainsi le pourvoir d'un moyen de communication qui lui était interdit auparavant. Cette particularité permet de déplacer la *portée* d'un système, changer sa connectivité(Parrow, 2001).

La connectivité des autres algèbres de processus est déclarée de façon statique. On sait au départ qu'*Alice* partage un canal avec *Bob* et un autre avec *Charles*, mais si *Bob* et *Charles* ne partagent pas de canal, ils n'ont et n'auront jamais moyen de communiquer. Par contre, le π-calcul permet la mobilité des canaux. *Alice* peut servir d'intermédiaire entre *Bob* et *Charles* et envoyer à chacun un nouveau canal spécifique à eux deux. *Bob* et *Charles* peuvent maintenant se parler sans passer par *Alice* . La

configuration de communication a changé de façon dynamique, la portée du nouveau canal aussi. D'origine appartenant à *Alice* seulement, il a élargi ses utilisateurs pour englober *Bob* et *Charles* .

Le spi-calcul(Abadi and Gordon, 1997) étend le π-calcul avec les primitives d'encryption et signature permettant d'étudier les crypto-protocoles. Comme pour les algèbres de processus, le spi-calcul définit une sémantique opérationnelle permettant d'obtenir un ST. De même, la vérification se fait par vérification d'équivalence. Pour l'authentification, on définit d'abord deux systèmes : une implémentation et une spécification. L'implantation consiste en un système où les acteurs reçoivent n'importe quelle valeur qui passe sur le réseau (sys), alors que la spécification est le système qui ne reçoit que la valeur attendue de l'autre acteur (sys_{spec}). On vérifie ensuite que $sys \simeq sys_{spec}$. Quant à la confidentialité, soit le système $sys(M)$ un protocole où la valeur M doit rester secrète. On doit s'assurer que $\forall M' \neq M$, les deux exécutions du protocole ne peuvent être discriminées, donc $sys(M) \simeq sys(M')$.

Toutefois, à notre connaissance, aucun outil n'implante le spi-calcul et les exemples décrits dans les articles ne concernent que des protocoles de sécurité simples avec des failles connues.

2.2.4 Logiques temporelles, CTL

Les logiques temporelles constituent une autre méthode de spécification de propriétés de systèmes. Elles sont spécialisées dans les énoncés et raisonnements faisant intervenir la notion d'ordonnancement dans le temps(Schnoebelen et al., 1999). Nous parlerons plus spécifiquement de CTL (Computation Tree Logic). La syntaxe de CTL est la suivante :

$$\sigma \quad := \quad P \quad | \quad \neg\sigma \quad | \quad \sigma_1(\wedge | \vee | \Rightarrow | \Leftrightarrow)\sigma_2 \quad | \quad (\forall|\exists)(X|F|G|\sigma_1 U)\sigma_2$$

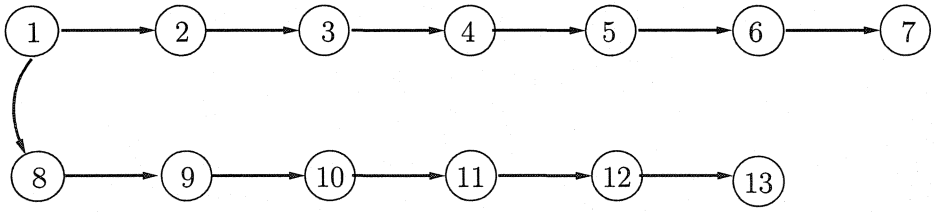


FIGURE 2.7 Automate pour vérification avec CTL

où P est une proposition atomique d'un état. Les formules CTL se combinent avec les combinateurs booléens $\wedge, \vee, \Rightarrow$ ou $\Leftrightarrow$. Les combinateurs temporels $X\sigma$ ("dans l'état suivant, σ est vraie"), $F\sigma$ ("dans le futur, σ sera vraie"), $G\sigma$ ("dans tous les états suivants, σ est vraie") et $\sigma_1 U \sigma_2$ ("σ_1 sera vraie jusqu'à ce que σ_2 le devienne) sont toujours précédés d'un quantificateur $\forall$ ("toutes les exécutions") ou $\exists$ ("il existe une exécution"). Cette dernière caractéristique fait de CTL une logique d'états : la vérité des formules ne dépend que de l'état courant et de ce qu'elle permet d'atteindre et non du chemin parcouru pour s'y rendre.

Le "model-checking" d'une formule CTL suppose la construction préalable d'un automate $\mathcal{A} = \langle \mathcal{S}, S_0, T, Prop, \rho \rangle$ modélisant le protocole où

- $\mathcal{S}$ est un ensemble d'états
- S_0 est l'ensemble des états initiaux
- $T \subseteq \mathcal{S} \times \mathcal{S}$ l'ensemble des transitions d'un état vers un autre état
- $Prop$ est un ensemble de propriétés
- $\rho : \mathcal{S} \rightarrow 2^{Prop}$ évalue les propositions atomiques pour chaque état.

On définit $[\![\phi]\!]_{\mathcal{A}} = \{s \in \mathcal{S} : s \models \phi\}$ l'ensemble des états de $\mathcal{A}$ pour lesquels ϕ est vérifiée. La vérification se fait par induction sur la syntaxe de la formule. Certaines formules, par exemple celles contenant un G, requièrent le calcul de points fixes. La satisfaction d'une formule est le résultat de l'inéquation $S_0 \subseteq [\![\phi]\!]_{\mathcal{A}}$.

Exemple. L'automate $\mathcal{A}$ de la figure 2.7 correspond à une petite partie du protocole

NSPK, avec intrus, où

$\mathcal{S} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$

$S_0 = \{1\}$

$T = \{(1, 2), (1, 8), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (8, 9),$
$(9, 10), (10, 11), (11, 12), (12, 13)\}$

$Prop = \{Alice\_a\_Bob, Bob\_confirme\_Alice\}$

$$\rho = \begin{cases} 1, 8, 9, 10, 11, 12, 13 \rightarrow \{\} \\ 2, 3, 4, 5, 6 \rightarrow \{Alice\_a\_Bob\} \\ 13 \rightarrow \{Bob\_confirme\_Alice\} \\ 7 \rightarrow \{Alice\_a\_Bob, Bob\_confirme\_Alice\} \end{cases}$$

La propriété d'authentification peut s'exprimer ainsi :

$$\forall G(Bob\_confirme\_Alice \Rightarrow Alice\_a\_Bob$$

Pour toutes les exécutions, toujours, si *Bob* authentifie *Alice* ($Bob\_confirme\_Alice$ est vrai), alors *Alice* désirait parler à *Bob* ($Alice\_a\_Bob$ est vrai).

On calcule maintenant $[\![\forall G(Bob\_confirme\_Alice \Rightarrow Alice\_a\_Bob]\!] = \{2, 3, 4, 5, 6, 7\}$. Puisque $S_0 \not\subseteq \{2, 3, 4, 5, 6, 7\}$, la propriété n'est pas vérifiée. $\lhd$

Le "model-checker" NuSMV permet de vérifier une formule CTL. (Panti et al., 2002) l'ont utilisé avec succès pour prouver certaines propriétés du protocole abstrait de SET par Lu et Smolka(Lu and Smolka, 1999). Ils ont utilisé la propriété de correspondance pour formaliser les propriétés d'authentification, confidentialité et intégrité. La propriété de correspondance entre deux événements dépendants α et β dit que si un événement α se produit n fois, alors un événement β s'est produit au moins n fois. On utilise des compteurs pour certains événements. L'authentification s'exprime en disant que si *Bob* a terminé n sessions avec *Alice* , alors *Alice* a dû dans le passé initier au moins n sessions avec *Bob* . En CTL $\forall G(Alice.begin(Bob) \geq Bob.end(Alice))$. La

confidentialité s'exprime en disant que l'intrus ne peut jamais recevoir un terme t plus souvent que *Alice* ne lui a envoyé et *Bob* ne peut être convaincu d'accepter un terme qui ne lui a pas été envoyé, ou, en CTL $\forall G(Alice.send(E,t) \geq E.receive(Alice,t)) \wedge \forall G(Alice.send(Bob,t) \geq Bob.receive(Alice,t))$. Ces propriétés ne sont pas respectées pour ce protocole ; un intrus peut conclure une transaction avec un marchand en faisant débiter le compte d'un autre client.

(Benerecetti and Giunchiglia, 2000) ont travaillé sur une extension de CTL en y ajoutant une logique de la connaissance. Les propositions atomiques sont maintenant des expressions de croyances comme celles décrites à la section 2.1.1.

Le système est modélisé par un ST multi-agents qui permet différents niveaux de "vues" des croyances du système. Un premier niveau représente la vue du système au complet, puis un second niveau comporte une vue pour les croyances de chaque agent. Un troisième niveau contient pour chaque agent, les vues des croyances qu'il a concernant chacun des autres agents, etc. L'avantage de cette méthode est que, tout comme c'était le cas avec les logiques de la connaissance, la prise en compte d'un intrus n'est pas nécessaire, on peut trouver directement la présence d'une attaque. Toutefois, le diagnostic de l'erreur s'avère difficile à trouver.

Nous n'entrerons pas dans les détails de l'algorithme de vérification des propriétés, semblable à celui de CTL. L'outil NuMAS implante cet algorithme et (Benerecetti et al., 2002) ont effectué la vérification du protocole de Lu et Smolka. Ils ont entre autres vérifié le requis que le client reçoit bien la preuve de l'autorisation de paiement par la banque, soit $C : \forall G($ **sees** $res \longrightarrow Bank$ **says** $res)$ ce qui signifie que, dans le vue du client (premier niveau), si **sees** res est vraie (le client reçoit la réponse de la banque), alors $Bank$ **says** res est vraie (la banque a envoyé la réponse à ce client). Ce requis n'est pas satisfait.

2.2.5 Logiques alternantes et jeux

On peut aussi voir un protocole comme un jeu de stratégie entre un système et un environnement(Alur et al., 1998b). C'est ce que l'on appelle un automate alternant (AA). La définition de cet AA rajoute à l'automate normal une fonction de transition $\delta : \mathcal{S} \times \mathcal{P} \rightarrow 2^{2^S}$, où $\mathcal{S}$ est l'ensemble des états et $\mathcal{P}$ l'ensemble des participants au protocole. Cette fonction associe un état et un acteur à un ensemble non-vide de choix où chaque choix est un ensemble d'états suivants possibles. Lorsque le système est dans un état s_i, chaque acteur p choisit un ensemble $S_p \in \delta(s_i, p)$. L'intersection des choix des principaux détermine l'état suivant $s_{i+1} = \bigcap_{p \in \mathcal{P}} S_p$.

On associe à l'AA la logique temporelle alternante (ATL). C'est une extension de CTL, qui en plus de répondre aux deux questions d'universalité ($\forall$) et d'existentialité ($\exists$), ajoute une troisième question : "existe-t-il un moyen pour un acteur A d'obtenir la satisfaction d'une propriété peu importe ce que les autres acteurs feront". Ce nouveau quantificateur s'écrit $\ll A \gg$. La signification de $\ll A \gg \phi$ est littéralement "il existe une stratégie pour les principaux de A pour rendre vraie la proposition ϕ".

Le "model-checking" d'une formule ATL utilise le même algorithme que CTL, à la différence que la fonction allant chercher les états précédents un état s doit maintenant sélectionner uniquement ceux à partir desquels les principaux de A peuvent coopérer pour atteindre s et non tous les états menant à s.

Exemple. Prenons l'automate de l'exemple de la section 2.2.4(fig.2.7), modèle du protocole NSPK. On rend cet automate alternant en ajoutant la fonction de transition δ :

$\delta(1, Alice) = \{\{2\}, \{8\}\}$ $\quad \delta(1, Bob) = \delta(1, Intrus) = \{\mathcal{S}\}$

$\delta(2, Intrus) = \{\{3\}\}$ $\quad \delta(2, Bob) = \delta(2, Alice) = \{\mathcal{S}\}$

etc...

L'authentification s'exprime en ATL $\ll\gg G(Bob\_confirme\_Alice \Rightarrow Alice\_a\_Bob)$, qui est l'équivalent de la formule CTL. Cette propriété n'est pas vérifiée. Par contre, sa version plus faible, $\ll Alice, Bob \gg G(Bob\_confirme\_Alice \Rightarrow Alice\_a\_Bob)$, l'est. On veut savoir s'il existe une stratégie pour *Alice* et *Bob* pour atteindre l'authentification. La réponse est oui, soit lorsque *Alice* n'amorce aucune communication ou lorsqu'elle ne parle pas à l'intrus. L'affaiblissement de la propriété aide à trouver un raffinement correct au protocole incorrect et ainsi à cerner l'erreur. $\lhd$

L'outil Mocha(Alur et al., 1998a) implante le "model-checking" d'un système avec ATL. La modélisation consiste en l'interaction de modules réactifs. Toutefois, il faut bien expliciter pour chaque module, incluant l'intrus, toutes les situations possibles et les choix offerts. La taille des systèmes à vérifier doit être, de ce fait, limitée.

(Alur et al., 1998a) ont vérifié l'authentification dans le protocole NSPK. Ils ont décrit le protocole en 3 modules *Alice*, *Bob* et *Intrus* et spécifié l'authentification d'*Alice* par *Bob* comme $\ll\gg G((Bob.state = COMMIT \wedge Bob.talk = Alice) \rightarrow Alice.request = Bob)$, et dans sa version faible $\ll Alice, Bob \gg G((Bob.state = COMMIT \wedge Bob.talk = Alice) \rightarrow Alice.request = Bob)$. Comme dans l'exemple, la version forte de la propriété est erronée, alors que la version faible est correcte.

Aussi, (Kremer and Raskin, 2001) ont vérifié l'équité de la non-répudiation du protocole optimiste de Zhou et Gollmann, ainsi que d'autres protocoles de non-répudiation optimistes, avec ATL. Il n'est pas nécessaire dans ce cas de tenir compte d'un intrus, le danger provenant d'acteurs malhonnêtes. On peut exprimer cette propriété en disant "qu'il n'existe pas de stratégie pour *Bob* et les canaux de communication pour atteindre un état où *Bob* a sa preuve d'origine sans que *Alice* ait sa preuve de réception" $(\neg \ll B, Com \gg F(EOO \wedge (EOO_k \vee Con_k) \wedge \neg \ll A \gg F(EOR \wedge (EOR_k \vee Con_k))))$ et vice versa pour *Alice* . De même, on peut montrer qu'en présence de deux agents honnêtes, *Alice* et *Bob* peuvent coopérer pour obtenir

l'équité ($\ll A, B \gg F((NRR \wedge NRO) \vee (\neg NRR \wedge \neg NRO))$).

Cette méthode de spécification a l'avantage de permettre la spécification de propriétés de façon intuitive. On peut facilement traduire en formule ATL une intuition du langage naturel en termes de jeux. Toutefois, le hic se trouve dans la modélisation qui requiert beaucoup d'efforts.

Récemment, (de Alfaro et al., 2001) ont développé des heuristiques symboliques pour étudier les jeux à états infinis et ont réussi à étendre la variété de jeux pouvant être résolus de façon symbolique.

2.3 Autres méthodes formelles

Alors que plusieurs méthodes peuvent être classées dans les catégories "model-checking" ou "theorem-proving", d'autres, plus récentes, peuvent entrer dans l'une ou l'autre de ces catégories selon les critères de classification utilisés. Nous les présentons donc dans une section à part.

2.3.1 Modèle "strand-space"

Les méthodes décrites jusqu'à présent sont basées sur un modèle de traces puisqu'on définit un système de transitions par des états et les transitions possibles à partir de ces états. Il existe un autre modèle, celui des "strand-spaces"(Guttman, 2001). Les définitions suivantes aideront à comprendre cette approche.

- Un "strand" est une séquence de transmissions et réceptions de termes par un acteur.
- Un événement $+m$ représente l"'envoi du message m" tandis que l'événement $-m$

signifie la "réception du message m".

- La relation $p \Rightarrow p'$ indique que l'événement p' suit immédiatement l'événement p dans un même "strand".
- Un espace de "strands" est un ensemble de "strands" dont la structure est imposée par la relation $p \rightarrow p'$ qui associe deux événements d'envoi et de réception de message par deux "strands" distincts.
- Un "bundle" $\mathcal{C}$ est un espace de "strands" qui représente une exécution du protocole. C'est une collection finie de noeuds et de flèches bien-fondée telle que lorsqu'un "strand" reçoit un message m, il y a un unique noeud qui a envoyé le message m immédiatement avant. Par contraste, dans un espace de "strands", si un "strand" envoie un message, aucun ou plusieurs "strands" peuvent le recevoir immédiatement.

On compte deux types de "strands", soit les "strands" réguliers qui représentent les actions d'un acteur pendant une session du protocole, et les "strands" pénétrateurs qui simulent le comportement d'un intrus. L'ensemble des données apparaissant dans les termes envoyés et reçus par chaque acteur constitue les paramètres pour cet acteur. La notion de "strands" paramétrés sera utile pour la vérification.

Exemple. La figure 2.8 montre l'attaque découverte par Lowe sur le protocole de NSPK exprimée en notation des "strand spaces". Cette figure est un exemple de "bundle" composé de deux "strands" réguliers A et B et un "strand" pénétrator P. On a les deux "strands" paramétrés $A[n_A, A, P, n_B]$ et $B[n_A, A, B, n_B]$. Les flèches doubles relient un noeud au noeud suivant du même "strand", tandis que les flèches simples montrent l'échange d'un message entre deux "strands". $\triangleleft$

Les propriétés des protocoles sont exprimées logiquement pour un "bundle". Par exemple, on spécifie l'authentification ainsi $\forall \mathcal{C}.Resp(\overrightarrow{x}) \in C \implies Init(\overrightarrow{x}) \in C$. Intuitivement, cela signifie que pour tout "bundle" $\mathcal{C}$ contenant un "strand" $Resp$ avec les paramètres $\overrightarrow{x}$, alors $\mathcal{C}$ doit contenir un "strand" $Init$ avec les mêmes paramètres,

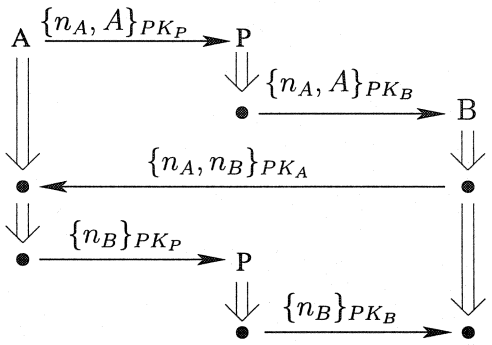


FIGURE 2.8 Attaque de Lowe sur le protocole de Needham-Schroeder exprimé en "strand space"

donc si un acteur termine une session en tant que répondeur, alors un autre acteur a initié cette session avec lui.

Quant à la confidentialité, on suppose un secret m partagé entre les strands s_1 et s_2. Soit $\mathcal{L}$ l'ensemble des termes obtenus par l'intrus dans une session du protocole. La valeur m ne doit pas faire partie de $\mathcal{L}$ pour tous les "bundles" contenant les strands s_1 et s_2. $\forall \mathcal{C}.\{s_1, s_2\} \in \mathcal{C} \Longrightarrow m \notin \mathcal{L}$.

À partir de la description d'un protocole et de la propriété à vérifier, un état initial est généré. Un état (symbolique) est une collection de propriétés de "bundles" et on dit que l'état *représente* l'ensemble des "bundles" satisfaisant ces propriétés. L'état initial est l'ensemble des "bundles" qui satisfont le protocole et la propriété. L'heuristique de preuve utilise les concepts de *but* (réception d'un terme) et d'*attachement de but* (association d'un but à l'envoi du terme correspondant). À l'état initial, aucun but n'est attaché. On crée les états suivants en choisissant un des buts et en l'attachant de toutes les façons possibles, avec les "strands" réguliers ou les "strands" pénétrateurs appropriés. On continue cette procédure jusqu'à ce que tous les buts soient attachés ou qu'il soit impossible de générer des états suivants à partir de l'état actuel. Cette heuristique peut ou non terminer puisque chaque attachement de but

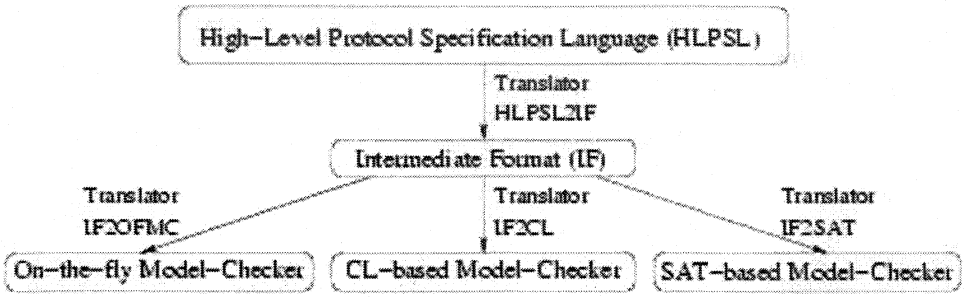


FIGURE 2.9 Architecture du système AVISS[3]

peut récursivement ajouter des buts dans la liste à attacher.

Le nombre d'états à parcourir dépend de la formule à vérifier, mais est en général très petit puisque chaque état inclut un grand nombre de possibilités et d'entrelâcements d'exécution, et qu'il n'y a pas une infinité de façons d'attacher un but à un protocole. Cette heuristique a été implantée dans un outil, *Athena*, et le nombre d'états obtenus en moyenne pour un protocole d'authentification est autour d'une vingtaine, comparativement à plusieurs centaines pour les méthodes de "model-checking" basées sur les traces(Song et al., 2001).

2.3.2 Les projets *AVISS* et *AVISPA*

Le projet *AVISS*[2] avait pour objectif de développer un outil d'analyse de protocoles de sécurité permettant l'intégration de plusieurs modules de vérification(Armando et al., 2002). La figure 2.9 montre l'architecture de l'outil développé.

[2]Automated Verification of Infinite State Systems : http ://www.informatik.uni-freiburg.de/ softech/research/projects/aviss/

[3]image tirée de (Armando et al., 2002)

D'abord, un compilateur, CASRUL(Bouallagui et al., 2002), prend en entrée la description d'un protocole en langage de haut niveau, de type *Alice&Bob*, et la transforme en un langage intermédiaire basé sur des règles de réécriture. L'avantage d'un tel traducteur de protocole est qu'il permet d'éviter les erreurs dues à la complexité d'un langage de bas niveau tel les SRT. De plus, il part d'une spécification objective du protocole dans un langage de haut niveau.

Ce langage intermédiaire adopte une stratégie paresseuse pour spécifier le comportement d'un attaquant, ie que contrairement au modèle de Dolev-Yao original où l'intrus peut créer un nombre infini de messages dont il faut borner la taille pour assurer la terminaison, cette stratégie ne crée qu'au besoin des messages, selon la structure attendue du participant.

Ensuite, l'outil d'*AVISS* comprend 3 modules de vérifications :

- Un "model-checker" à la volée déroule les règles à partir d'un état initial et produit un arbre infini qui est vérifié à la volée. Cette méthode a l'avantage de ne pas nécessiter la construction d'un modèle complet avant la vérification. Les sections de l'arbre sont construites sur demande.
- Un "model-checker" basé sur une logique des contraintes, *daTac*(Chevalier and Vigneron, 2002), et utilise un algorithme de résolution de contraintes pour chercher des incohérences et contradictions dans le système.
- Un "model-checker" basé sur SAT construit une formule propositionnelle encodant le déroulement des transitions, l'état initial et l'ensemble des états ne respectant pas les propriétés de sécurité. Cette méthode utilise la rapidité des résolveurs SAT.

Ces 3 modules trouvent des attaques sur les protocoles, mais ne prouvent pas la correction d'un protocole jugé sans erreur. Le succès de ce projet a ouvert la porte

au projet *AVISPA*[4], financé par la Commission Européenne, ayant pour objectif le développement d'une technologie commerciale pour l'analyse d'applications et de protocoles internet à grande échelle.

2.4 Conclusion

Ceci conclut cette revue des méthodes formelles pour l'analyse de crypto-protocoles. Notre formalisme se situe dans la catégorie "model-checking", dans la lignée des algèbres de processus. La nôtre est une algèbre symbolique dérivée de CCS avec passage de valeurs. Elle spécifie des processus avec des comportements plus complexes que CCS, et de façon plus intuitive que CSP. La spécification des propriétés par flux d'information n'a pas été abordée dans ce chapitre, mais fera l'objet du chapitre 4. La vérification de ces propriétés utilise la relation d'équivalence associée à notre algèbre, la bisimulation symbolique.

[4]Automated Validation of Internet Security Protocols and Applications : http ://www.avispa-project.org

CHAPITRE 3

MODÉLISATION DU PROTOCOLE

Les algèbres de processus sont une méthode assez intuitive d'exprimer les différents agents d'un protocole de communication. Leur sémantique permet d'obtenir facilement un système de transitions représentant le comportement du protocole. Toutefois, l'un des inconvénients majeurs de cette approche est l'explosion de l'espace-état lors de la synchronisation des agents. Cela est d'autant plus vrai lorsqu'un intrus peut s'infiltrer dans la communication. C'est pourquoi nous choisissons une approche symbolique.

3.1 SPPA symbolique

(Lafrance and Mullins, 2003a) définissent l'algèbre de processus SPPA symbolique (*Security Protocol Process Algebra*), une extension symbolique à CCS avec passage de valeurs et appels de fonction et implémentant un intrus comme un système déductif, conformément au modèle de Dolev-Yao(Dolev and Yao, 1983).

Chaque état d'un processus est défini par un terme de l'algèbre, ainsi que par une valeur ϕ représentant une garde booléenne. Cette garde dicte les contraintes sur le domaine des différentes variables utilisées jusqu'à présent dans le processus. La sémantique opérationnelle associée à SPPA symbolique indique comment mettre à jour la garde ϕ à chaque transition.

Exemple. Soit un processus simple exécutant un envoi de message $P = \overline{c}(a).0$. Au départ, soit une garde fixée à $\phi = true$. La transition symbolique correspondante serait

$\langle \overline{c}(a).0, \phi \rangle \xrightarrow{\overline{c}(a)} \langle 0, \phi \wedge a \in dom(c) \rangle$ où $dom(c)$ est le domaine des valeurs acceptées par le canal c, par exemple, un entier. $\triangleleft$

Toutefois, la relation d'équivalence associée au modèle SPPA ne tient pas compte du contexte symbolique et cette prise en compte occasionnerait une complexité des algorithmes qui éliminerait tous les avantages du modèle symbolique. Nous nous contentons d'un système symbolique *réductionniste par affectation.* Au lieu d'avoir une variable représentée par un domaine de valeurs possibles, nous assignons à la variable une seule valeur du domaine qui jouera un rôle symbolique. Une approche semblable a été abordée dans (Lin, 1998) : Chaque noeud d'un système de transitions contient un ensemble de variables libres. Ce sont les transitions qui définissent les valeurs de ces variables : lorsqu'une variable doit être (ré)assignée dans le modèle, on lui donne la plus petite valeur du domaine qui n'a pas encore été assignée. Toutefois, cette approche n'est valable que pour une classe restreinte de processus : elle demande des domaines bornés pour que chaque valeur du domaine puisse être assignée au moins une fois, ainsi que des processus récursifs pour qu'une action puisse être appelée plusieurs fois avec des valeurs différentes.

Par contre, le contexte des protocoles se prête bien à une approche par affectation : on s'intéresse rarement à la valeur exacte d'une variable, mais plutôt à sa valeur relativement à un autre terme. Par exemple, il importe peu qu'un nonce ait la valeur de 2 ou 1654, il faut que la valeur envoyée au départ par *Alice* soit la même que celle envoyée par la suite par *Bob* . Nous assignons donc une valeur symbolique, disons $n1$, au nonce et comparons par la suite avec toutes les valeurs possibles à une étape du protocole. Par comparaison, la méthode initiale de Mullins et Lafrance aurait associé à ce nonce la garde $n \in \mathcal{N}$.

Les sections suivantes définissent l'algèbre que nous utilisons pour les manipulations symboliques, soit la *Security Protocol Algebra for Symbolic Manipulations* (SPASM).

3.2 Syntaxe de SPASM

SPASM est une algèbre de processus symbolique avec passage de valeurs. De ce fait, sa syntaxe est riche et complexe, et permet de définir le comportement privé de chaque agent, de même que ses interactions avec les autres agents.

3.2.1 Données

Les processus spécifiés en SPASM s'échangent des valeurs et exécutent des fonctions sur ces valeurs. Cette section montre comment ces valeurs sont gérées.

Paramètres et termes. Un paramètre utilisé dans l'algèbre est défini par la grammaire suivante :

$p, q \quad := \quad Id$ (identificateur d'agent) $\mid \_x$ (variable) $\mid n$ (entier)

Ces paramètres font partie de trois catégories syntaxiques distinctes : les agents, les variables et les entiers, dénotés $\mathcal{A}$, $\mathcal{V}$ et $\mathcal{N}$, dont l'union (disjointe) est représentée par l'ensemble $\mathcal{Q}$ des paramètres. Lors de la modélisation d'un processus, les identificateurs d'agent prennent la valeur de chacun des principaux associés (voir 3.2.4), tandis que les variables sont associées à des termes. L'ensemble des termes $\mathcal{T}$ est construit comme suit :

t	:=	s	(string)	$\mid [t_1, ..., t_n]_{id}$	(signature)	$\mid Key$	(clé)
	$\mid$	n	(entier)	$\mid \{t_1, ..., t_n\}_t$	(encryption)	$\mid h(t_1, ..., t_n)$	(hash)
	$\mid$	$\_x$	(variable)	$\mid Choice(\{t_1, ..., t_n\})$		(choix)	

Un choix affecté à une variable représente un ensemble de valeurs possibles pour cette variable. Pour tout terme t, on dénote $fv(t)$ l'ensemble des variables libres dans t. Un *message* est un terme clos (ie $fv(m) = \emptyset$). L'ensemble des termes clos est noté $\mathcal{M}$.

L'ensemble $\mathcal{Q}$ des paramètres est un sous-ensemble de $\mathcal{T}$ utilisé pour la description syntaxique des processus.

Définition 3.1. Nous définissons une fonction de décomposition $\gamma : \mathcal{T} \rightarrow P(\mathcal{T})$ qui retourne l'ensemble des termes possibles pour un terme donné comme suit :

$$\gamma(t) = \begin{cases} T' & \text{si } t = Choice(T') \\ \{t\} & \text{sinon} \end{cases}$$

Son inverse est $\gamma^{-1} : P(\mathcal{T}) \rightarrow \mathcal{T}$ telle que

$$\gamma^{-1}(ts) = \begin{cases} t & \text{si } card(ts) = 1 \\ Choice(ts) & \text{si } card(ts) > 1 \\ \text{indéfini} & \text{si } ts = \emptyset \end{cases}$$

Clés d'encryption. Techniquement, n'importe quel terme peut servir de clé d'encryption. Certaines clés sont prédéfinies, soit les clés partagées, publiques et privées.

$$\mathcal{K}_d \quad := \quad Shared(t_{ag}, t_{ag}) \quad | \quad Public(t_{ag}) \quad | \quad Secret(t_{ag})$$

où t_{ag} est un terme représentant un agent.

Pour les autres termes utilisés comme clés, un sous-ensemble $\mathcal{K}_o \subseteq \mathcal{M}$ est défini comme un terme atomique (nonce, message ou nombre, ensemble dénoté $Atom$). $\mathcal{K}_o = Atom$. Soit l'ensemble de toutes les clés $\mathcal{K} = \mathcal{K}_o \bigcup \mathcal{K}_d$

Sauf dans le cas d'encryption avec clé publique où $Public(y)^{-1} = Secret(y)$ et vice versa, nous avons $k^{-1} = k$, soit de l'encryption symétrique.

3.2.2 Fonctions privées et canaux

Cette section présente les actions exécutées par les agents pour manipuler les termes.

Fonctions privées. On considère un ensemble Λ de fonctions privées pouvant être exécutées par les agents du protocole et créant des termes selon les règles ci-dessous. Ces fonctions se divisent en 2 catégories : les fonctions privées, prenant des valeurs symboliques en paramètres et créant de nouvelles valeurs, et les fonctions génératrices, ne prenant aucun paramètre, mais retournant une valeur fraîche.

Les règles de création de messages pour les fonctions privées $\lambda \in \Lambda$ sont

- $\text{encrypt}(k, t_1, ..., t_n) = \{t_1, ..., t_n\}_k$ (fonction d'encryption avec domaine $\mathcal{K} \times \mathcal{T}^n$)
- $\text{decrypt}(k^{-1}, \{t_1, ..., t_n\}_k) = (t_1, ..., t_m)$ (fonction de décryption avec domaine $\{(k^{-1}, \{t_1, ..., t_n\}_k) | k \in \mathcal{K}$ et $t_1, ..., t_n \in \mathcal{T}$ et $n = m\}$)
- $\text{hash}(t_1, ..., t_n) = h(t_1, ..., t_n)$ (fonction de hachage avec domaine $\mathcal{T}^n$)
- $\text{sign}_{Id}(t_1, ..., t_n) = [t_1, ..., t_n]_{Id}$ (fonction de signature avec domaine $\mathcal{T}^n$)[1]
- $\text{verifsign}(t_{ag}, [t_1, ..., t_n]_{Id}, u_1, ..., u_m)$ (fonction de vérification de signature avec domaine $t_{ag} \in \mathcal{T}, t_j, u_j \in \mathcal{T}$ et $n = m$ et $\forall j, t_j = u_j$)

Les fonctions génératrices $\lambda_{gen} \in \Lambda$, quant à elles, créent un message aléatoire ou une valeur fraîche selon les règles suivantes :

[1] L'algorithme de signature varie dans la littérature (voir section1.2. La fonction *sign* correspond à la signature d'une empreinte du message et sa vérification avec la fonction *verifsign* nécessite de connaître le message en clair. Pour signer un message complet, il vaut mieux utiliser les fonctions *encrypt(Secret(ag),...)* et *decrypt(Public(ag),...)*

- newnumber() = i (génère un nombre aléatoire avec $i \in \mathcal{N}$)
- newnonce() = n (génère un nonce frais avec $n \in \mathsf{String}$)
- newmsg() = m (génère un message simple aléatoire avec $m \in \mathsf{String}$)
- newkey() = k (génère une clé fraîche avec $k \in \mathcal{K}_o$)

Canaux. On considère un ensemble $\mathcal{C}$ de canaux de communication publics, propres au protocole, soit ceux permettant l'échange de messages entre les agents du protocole.

La syntaxe pour utiliser ces canaux dans le protocole est la suivante :

$$c \quad := \quad 'C(t_{ag}, cname; t_1, ..., t_n) \quad | \quad C(t_{ag}, cname; x_1, ..., x_m)$$

où $'C(t_{ag}, cname; t_1, ..., t_n)$ représente l'envoi d'un message $t_1, ..., t_n$ identifié *cname* vers l'agent t_{ag} et $C(t_{ag}, cname; x_1, ..., x_m)$ représente la réception d'un message identifié *cname* de l'agent t_{ag} où les valeurs reçues sont assignées aux variables $x_1, ..., x_m$.

La syntaxe des canaux publics, quoique peu naturelle en algèbre de processus, a été adoptée ici afin de différencier les canaux appartenant au protocole lui-même, sur lesquels les agents se synchronisent, des canaux externes, qui eux, gardent la syntaxe classique.

$$c_{ext} \quad := \quad cname(t_1, ..., t_m)$$

3.2.3 Processus

Les processus définissent le comportement d'un agent. Les préfixes μ du protocole sont obtenus ainsi :

$$\begin{aligned} \mu \quad := \quad & 'C(t_{ag}, cname; t_1, ..., t_n)(\text{canal out}) \quad | \quad C(t_{ag}, cname; x_1, ..., x_m) \text{ (canal in)} \\ | \quad & x_1, ..., x_m := \lambda(t_1, ..., t_n)(\text{fonction}) \quad | \quad x := \lambda_{gen}()(\text{fonction génératrice}) \\ | \quad & cname(t_1, ..., t_n) \text{ (canal externe)} \end{aligned}$$

Le comportement d'un agent est défini par les termes algébriques suivants :

$P, P\prime$	$:=$	$\mathbf{0}$	(agent vide)
	$\mid$	(P)	(parenthèse)
	$\mid$	$\mu.P$	(action d'un agent)
	$\mid$	$P + P\prime$	(alternative d'un agent)
	$\mid$	$P \mid P\prime$	(synchronisation de deux agents)
	$\mid$	$P \backslash L$	(restriction des actions de L)
	$\mid$	$[p = q]P$	(garde)
	$\mid$	$P/\mathcal{O}$	($\mathcal{O}$-observation)
	$\mid$	$(Aname(m_1, ..., m_n), id)$	(instanciation d'un agent)

où L est un ensemble d'actions et $\mathcal{O}$ une fonction d'observation (section 4.2) et $m_1, ..., m_n \in \mathcal{M}$.

3.2.4 Agents

Les agents d'un protocole sont identifiés par un nom, leurs connaissances de base, leur fiabilité et s'ils sont utilisés comme serveur ou non.

Un agent est décrit comme suit :

A	$:=$	P
	$\mid$	let $[trusted][server]Aname[(x_1, ..., x_n)] = P$ in P'

L'expression *trusted* décrit un agent fiable, c'est-à-dire avec lequel la communication se fait par des canaux privés qui ne peuvent être écoutés par un intrus, tandis que *server* indique que l'agent en question agit comme serveur, donc peut recevoir en parallèle de plusieurs principaux. Les termes $(x_1, ..., x_n)$ constituent les connaissances initiales de l'agent, qui lui seront allouées lors de l'instanciation. Toutefois, cette description ne représente que le comportement d'un agent, un rôle symbolique. Il ne devient une

entité physique, ou "principal", que lorsqu'il est décrit explicitement, par exemple en mettant des agents en parallèle. Ils sont alors décrits par leur entité symbolique, et un identificateur physique qui différenciera deux principaux de même agent.

L'instanciation d'une entité physique se fait de la façon suivante : $(Aname(m_1, ..., m_n), idA)$ signifiant que le principal ayant l'identité idA remplit le rôle décrit par $Aname$ pour lequel les valeurs $(m_1, ..., m_n)$ sont assignées aux variables $(x_1, ..., x_n)$ de idA.

3.3 Valuations du système

On associe à chaque processus de l'algèbre une valuation φ. Cette valuation assigne un terme à chacune des variables utilisées.

Définition 3.2. Une *valuation* φ est une fonction partielle $\mathcal{V} \rightarrow \mathcal{T}$. Chaque valuation φ est définie inductivement comme suit :

- $\frac{dom(\varphi)=\emptyset \;\; t \in \mathcal{M}}{\varphi[x \mapsto t]}$
- $\frac{dom(\varphi)=X \;\; fv(t) \subseteq X \;\; y \notin X}{\varphi[y \mapsto t]}$

La notation $\varphi[x \mapsto t]$ indique que l'on affecte dans φ une entrée telle que $\varphi(x) = t$. Nous utilisons $[\![t]\!]_\varphi$ pour signifier le terme clos après évaluation des variables de t par φ.

Proposition 3.1. Soit φ une valuation. Pour tout terme t, si $fv(t) \subseteq dom(\varphi)$ alors $[\![t]\!]_\varphi \in \mathcal{M}$.

Preuve. Il suffit de montrer que pour chaque $v \in dom(\varphi).[\![v]\!]_\varphi \in \mathcal{M}$. Pour cela, nous procéderons par induction sur les règles. Soit la propriété $P(\varphi) := \forall v \in dom(\varphi).[\![v]\!]_\varphi \in \mathcal{M}$. Prenons chacune des règles de création de φ.

- $\frac{dom(\varphi)=\emptyset \;\; t \in \mathcal{M}}{\varphi[x \mapsto t]}$

  Trivialement, $P(\varphi)$. Puisque $t \in \mathcal{M}$, alors $P(\varphi[x \mapsto t])$.

– $\frac{dom(\varphi)=X \;\; fv(t)\subseteq X \;\; y\notin X}{\varphi[y\mapsto t]}$

Supposons $P(\varphi)$. Puisque $fv(t) \subseteq dom(\varphi)$, alors pour chaque $v \in fv(t).[\![v]\!]_\varphi \in \mathcal{M}$, donc $[\![t]\!]_\varphi \in \mathcal{M}$. La variable y ne faisant pas encore partie de $dom(\varphi)$ nous avons $P(\varphi[y \mapsto t])$. □

Exemple. Soit $\varphi[x \mapsto \{y, z\}_k][y \mapsto 3][z \mapsto Alice]$, alors $[\![h(x)]\!]_\varphi = h(\{3, Alice\}_k)$. ◁

Soit la fonction $\overline{\varphi} : \mathcal{V} \to P(\mathcal{T})$ telle que $\overline{\varphi}(x) = \gamma(\varphi(x))$. De même, nous écrivons $[\![t]\!]_{\overline{\varphi}} = \gamma([\![t]\!]_\varphi)$.

Puisque chaque principal a ses propres variables, nous pouvons sectionner φ en φ_{id} pour chaque principal id tels que $\bigcup_{id\in\mathcal{P}} \varphi_{id} = \varphi$ et $\forall id1, id2 \in \mathcal{P}, id1 \neq id2$ implique $dom(\varphi_{id1}) \cap dom(\varphi_{id2}) = \emptyset$.

Avant d'aller plus loin, il convient d'éclaircir la sémantique concernant les valuations du système.

Les opérations sur les valuations φ sont :

- *Inclusion* $(\varphi_1 \subseteq \varphi_2)$: $\forall x_i \in dom(\varphi_1) \cap dom(\varphi_2).\overline{\varphi_1}(x_i) \subseteq \overline{\varphi_2}(x_i)$.
- *Intersection* $(\varphi_1 \wedge \varphi_2)$:
  $\forall x_i \in dom(\varphi_1) \cap dom(\varphi_2).\varphi'(x_i) = \gamma^{-1}(\overline{\varphi_1}(x_i) \cap \overline{\varphi_2}(x_i))$
  Si φ' total, alors φ' sinon $\perp$.
- *Union* $(\varphi_1 \vee \varphi_2)$:
  Si $\varphi_i \subseteq \varphi_j$ alors φ_j
  sinon si φ_1 et φ_2 ne diffèrent que sur un seul élément du domaine, x_i, alors $\varphi' = \varphi_1[x_i \mapsto \gamma^{-1}(\overline{\varphi_1}(x_i) \cup \overline{\varphi_2}(x_i))]$
  sinon $\varphi_1 \vee \varphi_2$ (union irréductible).
  Cet opérateur est plus compliqué puisque l'union ne peut pas toujours être efficace. Il arrive des situations où l'on doive garder l'union tel quel, car on ne peut le réduire en un seul φ'. Par exemple, soit $\varphi_1 = \varphi[x \mapsto Choice(0, 1, 2)][y \mapsto 2][z \mapsto 1]$

et $\varphi_2 = \varphi[x \mapsto Choice(0,1,2)][y \mapsto 0][z \mapsto 0]$, les variables y et z se trouvent liées l'une à l'autre, de sorte qu'il serait impossible d'avoir une valuation telle que $\varphi(y) = 2$ et $\varphi(z) = 0$.

- *Différence* $(\varphi_1 \backslash \varphi_2)$:

  $\bigvee_{x_i \in dom(\varphi_1) \cap dom(\varphi_2)} \varphi'_i$

  où $\varphi'_i =$

  $\varphi''_i = (\varphi_1[x_i \mapsto \gamma^{-1}(\overline{\varphi_1}(x_i) \backslash \overline{\varphi_2}(x_i))]$

  si φ''_i total, alors φ''_i sinon $\bot$.

Proposition 3.2. Pour simplifier un opérateur d'union irréductible, nous utiliserons les propriétés suivantes :

- $(\varphi_1 \vee \varphi_2) \wedge (\varphi_3 \vee \varphi_4) = (\varphi_1 \wedge \varphi_3) \vee (\varphi_1 \wedge \varphi_4) \vee (\varphi_2 \wedge \varphi_3) \vee (\varphi_2 \wedge \varphi_4)$.
- $(\varphi_1 \vee \varphi_2) \backslash (\varphi_3) = (\varphi_1 \backslash \varphi_3) \vee (\varphi_2 \backslash \varphi_3)$.
- $\varphi_1 \backslash (\varphi_2 \vee \varphi_3) = (\varphi_1 \backslash \varphi_2) \backslash \varphi_3$.

3.4 Sémantique de SPASM

Cette section présente la sémantique opérationnelle de SPASM à partir de laquelle on obtient un système de transitions.

Marqueurs. Avant de décrire la sémantique de SPASM, nous introduisons la notion de marqueurs. En algèbre de processus, la synchronisation entre deux processus est exprimée en remplaçant les actions *out* et *in* par une action silencieuse τ. Ceci mène à une perte d'information importante quant au contenu du message échangé et son rôle dans le protocole. Pour pallier à ce problème, on utilise des marqueurs. Ceux-ci ne font pas partie de la syntaxe de l'algèbre, mais sont utilisés pour identifier une communication entre les processus. Par exemple, soit un canal interne $'C_{Alice}(\_Bob, \_msg1; a_1, a_2)$ une communication envoyée par *Alice*. Elle est reçue par *Bob* avec l'action $C_{Bob}(\_Alice, \_msg1; x_1, x_2)$. Plutôt que se transformer en τ, cette

action devient un marqueur $msg1(Alice, a_1, a_2, Bob, x_1, x_2)$ appartenant à *Alice* et *Bob*.

Sémantique opérationnelle. La sémantique opérationnelle de SPASM est une extension symbolique de celle de CCS avec passage de valeurs. Les règles sont décrites à la figure 3.1. Soit $P, P_1, ..., P_n$ des processus, φ, φ' des valuations, $t, u \in \mathcal{T}$ et $x \in \mathcal{V}$.

Informellement, les règles de la sémantique ont la signification suivante :

- La règle Externe représente des canaux externes qui font plutôt office de "piliers" marquant certains points du protocole importants pour le concepteur ; par exemple, la fin d'un protocole d'authentification pourrait contenir l'action *auth* pour marquer une identification réussie.
- Les règles Encryption, Decryption, Verifsign et Fonction permettent d'exécuter les fonctions du système seulement si les paramètres sont à l'intérieur du domaine.
- La règle de choix non-déterministe et la conditionnelle (Alternative, Match) sont définies de la façon habituelle.
- Le protocole est considéré comme un système synchrone : un message envoyé sur des canaux publics sera obligatoirement reçu, parfois par un intrus. Nous distinguons deux règles pour la concurrence des processus, Synchronisation et Parallellisation respectivement pour les canaux publics et les autres préfixes.
- La règle Restriction interprète l'opérateur $\backslash L$ comme étant le système de transitions permettant seulement les actions $\notin L$.
- La règle $\mathcal{O}$-Observation interprète l'observation d'un processus du point de vue de l'observateur $\mathcal{O}$.
- Finalement, la règle Instanciation définit le comportement d'un principal idA.

Externe
$$\frac{-}{\langle cname(p).P,\varphi\rangle \overset{cname(p)}{\longrightarrow} \langle P,\varphi\rangle}$$

Encryption
$$\frac{\forall k_i \in [\![k]\!]_{\overline{\varphi}}.k_i \in Key_{id} \text{ and } u=encrypt(k_i,t)}{\langle y:=encrypt_{Id}(k,t).P,\varphi\rangle \overset{u:=encrypt_{Id}(k_i,t)}{\longrightarrow} \langle P,\varphi[k\mapsto k_i][y\mapsto u]\rangle}$$

Decryption
$$\frac{\forall k_i \in [\![k]\!]_{\overline{\varphi}}.k_i \in Key_{id} \text{ and } u=decrypt(k_i,t)}{\langle y:=decrypt_{Id}(k,t).P,\varphi\rangle \overset{u:=decrypt_{Id}(k_i,t)}{\longrightarrow} \langle P,\varphi[k\mapsto k_i][y\mapsto u]\rangle}$$

Verifsign
$$\frac{\forall id_i \in [\![id]\!]_{\overline{\varphi}}.verifsign(id_i,Sign(t,id_i),u) \text{ and } z=[\![t]\!]_{\overline{\varphi}} \cap [\![u]\!]_{\overline{\varphi}}}{\langle verifsign_{Id}(id,u_0,u).P,\varphi\rangle \overset{verifsign_{Id}(id_i,Sign(t,id_i),u)}{\longrightarrow} \langle P,\varphi[id\mapsto id_i][u\mapsto \gamma^{-1}(z)]\rangle}$$

Fonction
$$\frac{u=\lambda_{Id}(t)}{\langle y:=\lambda_{Id}(t).P,\varphi\rangle \overset{u:=\lambda_{Id}(t)}{\longrightarrow} \langle P,\varphi[y\mapsto u]\rangle}$$

Alternative
$$\frac{\langle P_j,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P'_j,\varphi'\rangle}{\langle P_1+P_2,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P'_j,\varphi'\rangle} \text{ for } j=1,2$$

Garde
$$\frac{\langle P,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P',\varphi'\rangle \text{ and } z=[\![t]\!]_{\overline{\varphi}} \cap [\![u]\!]_{\overline{\varphi}} \neq \emptyset}{\langle [t=u]P,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P',\varphi'[t\mapsto \gamma^{-1}(z)][u\mapsto \gamma^{-1}(z)]\rangle}$$

Parallellisation
$$\frac{\langle P_j,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P'_j,\varphi'\rangle \text{ and } \alpha \notin \mathcal{C}}{\langle P_1|...|P_k,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P'_1|...|P'_k,\varphi'\rangle}$$

où $P'_i = \begin{cases} P'_j & \text{si } i=j \\ P_i & \text{sinon} \end{cases}$

Synchronisation
$$\frac{\langle P_j,\varphi\rangle \overset{'C_{P_j}(\_P_k,cn;t)}{\longrightarrow} \langle P'_j,\varphi'\rangle \text{ and } \langle P_k,\varphi\rangle \overset{C_{P_k}(\_x_{id_j},cn;x)}{\longrightarrow} \langle P'_k,\varphi\rangle \text{ and } t\in dom(cn)}{\langle P_1|...|P_n,\varphi\rangle \overset{cn(P_j,t,P_k,x)}{\longrightarrow} \langle P'_1|...|P'_n,\varphi[x\mapsto t]\rangle}$$

où $P'_i = \begin{cases} P'_j & \text{si } i=j \\ P'_k & \text{si } i=k \\ P_i & \text{sinon} \end{cases}$

Restriction
$$\frac{\langle P,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P',\varphi'\rangle \text{ and name}(\alpha)\notin L}{\langle P\backslash L,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P',\varphi'\rangle}$$

Restriction partielle
$$\frac{\langle P,\varphi\rangle \overset{\alpha(t)}{\longrightarrow} \langle P',\varphi'\rangle \text{ and } \alpha(u)\in L \text{ and } z=[\![t]\!]_{\overline{\varphi}} \backslash [\![u]\!]_{\overline{\varphi}} \neq \emptyset}{\langle P\backslash L,\varphi\rangle \overset{\alpha(t)}{\longrightarrow} \langle P',\varphi'[t\mapsto \gamma^{-1}(z)]\rangle}$$

$\mathcal{O}$-Observation
$$\frac{\langle P,\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P',\varphi'\rangle \text{ and } \alpha\in\mathcal{O}^{-1}(\alpha)}{\langle P/\mathcal{O},\varphi\rangle \overset{\alpha}{\longrightarrow} \langle P'/\mathcal{O},\varphi'\rangle}$$

Instanciation
$$\frac{\langle \text{let } A(x)=P \text{ in } P_1(A(t),idA)P_2,\varphi\rangle}{\langle P_1 P_{idA} P_2,\varphi[x\mapsto t]\rangle}$$

FIGURE 3.1 Sémantique opérationnelle de SPASM

Axiome $\frac{t \in L}{L \vdash t}$

Analyse

Déchiffre $\frac{L \vdash \{t\}_k \; L \vdash k^{-1}}{L \vdash t}$ GetSign $\frac{L \vdash [t]_{ag} \; L \vdash PK(ag)}{L \vdash t}$

ChoiceDecomp $\frac{L \vdash Choice(ts)}{\forall t \in ts \; L \vdash t}$

Synthèse

Chiffre $\frac{L \vdash t \; L \vdash k}{L \vdash \{t\}_k}$ Signe $\frac{L \vdash t}{L \vdash [t]_e}$

Hash $\frac{L \vdash t}{L \vdash h(t)}$ ChoiceComp $\frac{ts \subseteq L}{L \vdash Choice(ts)}$

FIGURE 3.2 Règles de déduction de l'intrus

3.5 Modèle de l'intrus

En plus des acteurs honnêtes décrits dans la syntaxe d'entrée, un système de transitions SPASM inclut un intrus conforme au modèle de Dolev-Yao (Dolev and Yao, 1983). Cet intrus est un système déductif qui peut construire des messages à partir de messages connus, signer ces messages, les hacher, chiffrer et déchiffrer des messages s'il connaît la clé. Ce modèle repose sur l'hypothèse du chiffrement parfait, selon laquelle on ne peut obtenir d'information sur un message chiffré sans en posséder la clé.

Le système déductif de l'intrus est composé de deux fonctions : *analyse* et *synthèse*. *Analyse* $:P(\mathcal{T}) \rightarrow P(\mathcal{T})$ obtient tous les termes que l'intrus peut déduire à partir des connaissances qu'il a et *synthèse* $:P(\mathcal{T}) \rightarrow P(\mathcal{T})$ crée des nouveaux termes.

Nous définissons le prédicat $L \vdash m$ comme "à partir de l'ensemble L de ses connaissances, l'intrus peut déduire m". La figure 3.2 décrit les règles d'analyse et de synthèse de messages.

Initialement, l'intrus connaît les clés publiques de tous les principaux du système, sa clé privée, de même que les clés qu'il partage avec tous les principaux. Il connaît aussi pour chaque type de donnée élémentaire (nonce, message, clé), une valeur symbolique aléatoire différente de celles qui seront générées par les acteurs. Il possède une identité propre qu'il peut utiliser pour communiquer avec les autres acteurs. Il peut aussi personnifier n'importe quel autre acteur honnête dont il connaît l'identité.

$$L_{init} = \{e, n_e, msg_e, k_e, SK(e), \forall p \in \mathcal{P}.(id_p, PK(p), Shared(e,p))\}$$

Ce L_{init} constitue les connaissances par défaut d'un intrus pour n'importe quel protocole.

Le processus associé à l'intrus est le processus ennemi générique. La sommation $\sum$ représente une série de choix non-déterministes, gérés par la règle Alternative[2].

$$E(L) := \sum_{c \in \mathcal{C}} (C(\_, c; y).E(L \cup \{y\}) + \sum_{id \in (\mathcal{P} \cup \{e\})} \sum_{c \in \mathcal{C}} ('C(id, c; \gamma^{-1}(L))).E(L))$$

Le première sommation signifie qu'il peut lire tout message qui passe sur un canal public et l'ajouter à sa base de connaissances. La seconde sommation lui permet d'envoyer tout message qu'il peut générer en se faisant passer pour n'importe quel agent dont il connaît l'identité.

L'exécution des règles d'analyse termine puisque chaque règle déduit des sous-termes d'un terme connu, et l'ensemble des termes connus est fini. Les nouveaux sous-termes déduits pouvant récursivement permettre de déduire d'autres sous-termes, $Analyse(L)$ est le résultat d'un calcul de points fixes. Par contre, les règles de synthèse de messages ont une structure infinie. Pour éviter le problème de non-terminaison qui

[2] $\sum_{i \in \{1,\ldots,4\}} P_i = P_1 + P_2 + P_3 + P_4$

en résulte, nous pouvons borner le comportement de l'intrus, par exemple, en limitant la profondeur maximale des messages (pas plus de 3 encryptions). Nous avons plutôt opté pour l'exécution paresseuse des opérations de l'intrus, ie seulement au besoin. Au lieu de calculer, à chaque envoi de l'intrus, l'ensemble des messages possibles, on le calcule seulement au moment où un participant nécessite une certaine information.

Exemple. Soit le processus synchronisé suivant :

$$\cdots .msg1(e, \gamma^{-1}(L), Alice, \_x).\_y := decrypt(SK(Alice), \_x). \cdots$$

avec $L = L_{init} \cup \{\{m\}_{PK_{Alice}}\}$. Après l'exécution du marqueur, nous avons $\varphi[\_x_{Alice} \mapsto \gamma^{-1}(L)]$.

La fonction de décryption suggère que $\_x_{Alice} = \{t\}_{PK(Alice)}$. On effectue d'abord $analyse(L)$ et on vérifie si un terme de cette forme est présent. Nous trouvons le terme $\{m\}_{PK_{Alice}}$. On le décrypte pour obtenir la valeur m.

On ajoute ensuite les termes de cette forme qui auraient pu être générés par l'intrus. Nous avons que, puisque $L \vdash PK(Alice)$, $\forall t \in L.L \vdash \{t\}_{PK(Alice)}$.

Donc, après la décryption, $\varphi[\_x_{Alice} \mapsto \gamma^{-1}(L)][\_y_{Alice} \mapsto \gamma^{-1}(L \cup \{m\})]$. $\lhd$

3.6 Conclusion

La syntaxe de SPASM permet de décrire le comportement des agents d'un protocole. La sémantique opérationnelle de cette algèbre gère symboliquement les manipulations de données des agents mis en concurrence, possiblement avec un intrus paresseux. Le système de transitions ainsi obtenu constitue le modèle du protocole à analyser. Il faut maintenant spécifier les propriétés qu'il doit satisfaire. Nous nous baserons sur la théorie du flux d'information.

CHAPITRE 4

SPÉCIFICATION DES PROPRIÉTÉS

La deuxième étape après la modélisation du protocole est la modélisation des propriétés à vérifier. Pour ce faire, nous utiliserons la méthode de flux d'information développée au CRAC. Nous expliquons les notions de flux d'information et les propriétés de non-interférence qui en découlent. Nous présentons ensuite la propriété d'interférence admissible puis la spécification des propriétés de sécurité basée sur l'interférence admissible.

4.1 Non-interférence et GNDC

Bien qu'on retrouve des références aux méthodes par flux d'information dans les années 80, la principale contribution à son développement vient de Focardi et Gorrieri dans le milieu des années 90. (Focardi and Gorrieri, 2001; Focardi, 2001) résument le travail accompli sur ce sujet.

La méthode par flux d'information a d'abord été conçue pour assurer la confidentialité d'information classifiée. Cela exige un contrôle du flux d'information pour éviter qu'il y ait des fuites dans le système, d'un usager ayant accès à l'information vers un usager n'y ayant pas accès. La non-interférence capture bien cette idée.

Définition 4.1 (Non-interférence). La *non-interférence* est la propriété par laquelle l'action d'un usager d'un système ne doit pas affecter l'observation qu'un autre usager peut faire du système. (Focardi and Gorrieri, 2001)

Focardi et Gorrieri distinguent deux niveaux d'usagers, soit le haut niveau ('hi') et

le bas niveau ('lo'). L'usager de haut niveau ne veut pas que ses activités interfèrent avec celles de l'usager 'lo'. Dans un système informatique fermé, des politiques de contrôle d'accès régissant l'interaction de l'usager avec le système garantissent un certain niveau de sécurité Toutefois, dans un réseau, il est assez aisé pour n'importe qui d'avoir accès à une partie du système et tout peut arriver aux données échangées.

Dans ce contexte, un protocole est vu comme un ensemble de bons comportements (honnêtes) mis en présence d'un environnement hostile. Il ne faut pas que cet environnement hostile n'induise de mauvais comportements dans le protocole. L'interférence est le mauvais comportement induit.

Considérant un formalisme basée sur les algèbres de processus, toutes les propriétés de flux d'information peuvent s'exprimer sous la forme algébrique suivante :

$$\text{Un système est } \Phi\text{-sécure } \Leftrightarrow \mathcal{C}_\Phi \Delta \mathcal{D}_\Phi$$

où Φ est une propriété à vérifier et Δ représente une relation comportementale pour l'algèbre utilisée. $\mathcal{C}_\Phi$ et $\mathcal{D}_\Phi$ sont des contextes pour la propriété Φ, c'est-à-dire des processus qui, prenant en paramètre le protocole synchronisé, modifie son comportement pour l'adapter au contexte.

Focardi et Gorrieri précisent davantage ce gabarit pour les propriétés de sécurité. Ils définissent $\alpha_\Phi(P)$ comme l'ensemble de tous les comportements du protocole P respectant la propriété de sécurité Φ. Informellement, on écrit que P garantit une propriété Φ si, peu importe l'environnement hostile considéré, tous les comportements de P sont dans $\alpha_\Phi(P)$. Algébriquement, cela s'exprime par la propriété GNDC (non-déductibilité généralisée sur la composition).

$$P \text{ est } GNDC_{\sqsubseteq}^{\Phi} \Leftrightarrow \forall E \text{ un processus hostile.} (P|E)\backslash C \sqsubseteq \alpha_\Phi(P)$$

où $\sqsubseteq$ est un préordre dépendant de la propriété et C est l'ensemble des canaux de communication. Cette approche s'apparente à celle du spi-calcul (section 2.2.3) et de FDR (section 2.2.2) où les propriétés étaient exprimées comme une équivalence entre une spécification et une implémentation du système. Ici, l'implémentation est le protocole dans son environnement $(P|E)$ et la spécification est le protocole tel qu'il devrait être exécuté, ie les comportements satisfaisant la propriété $(\alpha_\Phi(P))$.

Chaque propriété des protocoles à vérifier nécessite l'instanciation de la relation de préordre et le processus α. Pour l'authentification et la confidentialité, le préordre en sera un de traces. C'est-à-dire qu'on exige que toutes les séquences d'événements possibles dans $(P|E)\backslash C$ soient possibles dans α_Φ.

Exemple. Authentification d'Alice par Bob dans le protocole NSPK. On définit d'abord deux événements $commit_{Bob}(Alice)$ et $run_{Alice}(Bob)$ représentant les événements où "*Bob* a terminé avec succès une session avec *Alice* " et "*Alice* a réellement initié une session avec *Bob* ". Le processus $\alpha_{auth}(P)$ est donc un processus tel que l'événement $run_{Alice}(Bob)$ précède toujours l'événement $commit_{Bob}(Alice)$, ou, en d'autres termes que "si *Bob* termine une session en authentifiant avec succès l'agent *Alice*, alors *Alice* a débuté une session avec l'agent *Bob* ".

Soit la spécification du protocole de Needham-Schroeder à clé publique :

1. Alice $\rightarrow$ Bob : $\{N_a, Alice\}_{PK_{Bob}}$ $run_{Alice}(Bob)$
2. Bob $\rightarrow$ Alice : $\{N_a, N_b\}_{PK_{Alice}}$
3. Alice $\rightarrow$ Bob : $\{N_b\}_{PK_{Bob}}$ $commit_{Bob}(Alice)$

Les événements de la colonne de droite sont ceux marquant un point du protocole et s'exécutant par le principal concerné après l'envoi ou la réception des messages correspondant. Ceux de la colonne de gauche sont des canaux et, par conséquent,

cachés à l'observateur. Nous avons donc le processus du protocole correct suivant

$$\alpha_{auth}(P) = run_{Alice}(Bob).commit_{Bob}(Alice).0$$

Toutefois, nous avons la séquence

1. Alice $\rightarrow$ E : $\{N_a, Alice\}_{K_E}$
2. E(Alice) $\rightarrow$ Bob : $\{N_a, Alice\}_{K_{Bob}}$
3. Bob $\rightarrow$ E(Alice) : $\{N_a, N_b\}_{K_{Alice}}$
4. E $\rightarrow$ Alice : $\{N_a, N_b\}_{K_{Alice}}$
5. Alice $\rightarrow$ E : $\{N_b\}_{K_E}$
6. E(Alice) $\rightarrow$ Bob : $\{N_b\}_{K_{Bob}}$ $commit_{Bob}(Alice)$

Pour le E de cette exécution, $(P|E)\backslash C \not\sqsubseteq \alpha_{auth}(P)$, puisque $commit_{Bob}(Alice)$ ne fait pas partie des traces possibles de $\alpha_{auth}(P)$. Il y a donc une faille d'authentification dans ce protocole. $\triangleleft$

Exemple. Confidentialité de N_b dans le protocole NSPK. On définit l'événement $learnt_E(M)$ par lequel un processus ennemi E apprend la valeur de M. Le processus $\alpha_{conf}(P)$ est donc le processus dans lequel l'événement $learnt_E(M)$ ne se produit jamais ou "l'intrus n'apprend jamais la valeur M". Pour un protocole correct,

$$\alpha_{conf}(P) = 0$$

Toutefois, le même intrus que précédemment amène l'exécution suivante

1. Alice $\rightarrow$ E : $\{N_a, Alice\}_{K_E}$
2. E(Alice) $\rightarrow$ Bob : $\{N_a, Alice\}_{K_{Bob}}$
3. Bob $\rightarrow$ E(Alice) : $\{N_a, N_b\}_{K_{Alice}}$

4. E $\rightarrow$ Alice : $\{N_a, N_b\}_{K_{Alice}}$
5. Alice $\rightarrow$ E : $\{N_b\}_{K_E}$ $learnt_E(N_b)$
6. E(Alice) $\rightarrow$ Bob : $\{N_b\}_{K_{Bob}}$

Pour le E de cette exécution, $(P|E)\backslash C \not\sqsubseteq \alpha_{conf}(P)$, puisque $learnt_E(N_b)$ n'est pas une trace possible de $\alpha_{conf}(P)$. La confidentialité n'est donc pas assurée. $\lhd$

Exemple. Non-répudiation équitable du protocole de Zhou-Gollman. On peut aussi vérifier la satisfaction de l'équité de la non-répudiation avec la propriété $GNDC$. On ajoute deux événements $evidence_{Alice}$ et $evidence_{Bob}$ pour représenter respectivement la réception par *Alice* et par *Bob* d'un message que l'agent envoyeur ne peut répudier. Le processus $\alpha_{NR}(P)$ est donc celui où la réception de l'évidence d'origine d'*Alice* par *Bob* est toujours suivie de la réception par *Alice* de l'évidence de réception de *Bob*.

Prenons uniquement l'échange entre *Alice* et *Bob* dans le protocole de Zhou-Gollman optimiste (annexe I.6.2) :

1. Alice $\rightarrow$ Bob : $f_{EOO}, Bob, L, \{M\}_K, EOO$
2. Bob $\rightarrow$ Alice : $f_{EOR}, Alice, L, EOR$
3. Alice $\rightarrow$ Bob : $f_{EOO_K}, Bob, L, K, EOO_K$ $evidence_{Bob}$
4. Bob $\rightarrow$ Alice : $f_{EOR_K}, Alice, L, EOR_K$ $evidence_{Alice}$

Nous avons alors

$$\alpha_{NR}(P) = evidence_{Bob}.evidence_{Alice}.0$$

La situation suivante se pose lorsqu'on a un *Bob* malhonnête qui décide volontairement de couper la communication après la réception de sa preuve par *Alice* :

1. Alice $\rightarrow$ Bob : $f_{EOO}, Bob, L, \{M\}_K, EOO$
2. Bob $\rightarrow$ Alice : $f_{EOR}, Alice, L, EOR$
3. Alice $\rightarrow$ Bob : $f_{EOO_K}, Bob, L, K, EOO_K$ $evidence_{Bob}$

L'équité n'est clairement pas préservée ici. Or, si on considère le même préordre de traces que pour l'authentification et la confidentialité, nous avons $(P|E)\backslash C \sqsubseteq \alpha_{NR}(P)$.

La relation de préordre choisie dans ce cas-ci doit être sensible au blocage, nous la noterons $\sqsubseteq_{blocage}$. En effet, dans la deuxième exécution fautive de l'exemple précédent, le processus bloque après réception de l'évidence de *Bob* . Sans entrer dans les détails d'une telle relation, on a maintenant que $(P|E)\backslash C \not\sqsubseteq_{blocage} \alpha_{NR}(P)$. L'équité de la non-répudiation n'est donc pas satisfaite. $\lhd$

Focardi et Gorrieri ont implanté leur méthode de spécification et vérification dans un outil appelé *CoSeC*. Son utilisation a permis de découvrir des failles auparavant non-documentées, certaines sur des protocoles jugés corrects.

4.2 Interférence admissible

La non-interférence capture l'idée qu'une action de haut niveau ne devrait jamais être vue par un usager de bas niveau. Toutefois, cette notion est plutôt restrictive. Il peut arriver qu'il soit acceptable qu'une action de haut niveau puisse être vue. (Mullins, 2000) décrit cette notion d'*interférence admissible*. Un usager de haut niveau peut interférer avec un usager de bas niveau seulement si c'est à travers un système de déclassification.

L'ensemble des actions visibles Σ d'un ST est partitionné en 3 ensembles disjoints selon le niveau de sécurité requis : Hi, Lo, Dwn. Les actions de ce dernier ensemble permettent la communication entre le haut niveau et le bas niveau, normalement interdite. Elles annulent l'effet de toute action de haut niveau qui se produit avant son exécution et rendent ainsi admissible l'interférence que cette action $'hi'$ aurait pu causer.

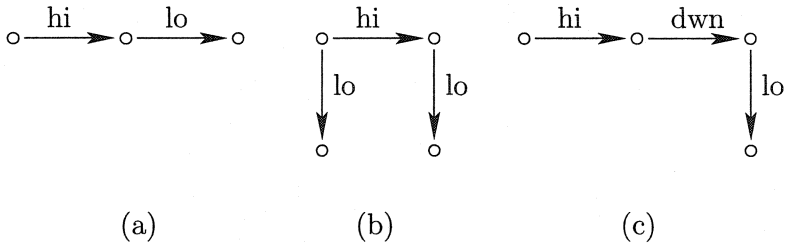


FIGURE 4.1 Exemples de processus satisfaisant a) interférence, b) non-interférence, c) interférence admissible

Exemple. Soit le processus de la figure 4.1a. On voit que l'action *'lo'* ne peut se produire que si l'action *'hi'* se produit d'abord. Il y a donc interférence puisqu'un usager de bas niveau qui observe l'action *'lo'* obtient de l'information sur ce qui se passe au haut niveau. Pour que ce processus satisfasse la non-interférence, il faut ajouter la possibilité d'exécuter l'action *'lo'* à tout moment du système (figure 4.1b). Par contre, si l'on veut permettre cette interférence, il suffit d'ajouter une action de déclassification *'dwn'* après l'action de haut niveau (figure 4.1c). Un usager de bas niveau peut encore déduire les actions de haut niveau, mais l'action *'dwn'* rend cette action tout à fait légale. ◁

Comme c'était le cas pour les propriétés de non-interférence, l'interférence admissible peut s'exprimer sous la forme algébrique des flux d'information

$$\text{Un système est } \Phi\text{-sécure } \Leftrightarrow \mathcal{C}_\Phi \Delta \mathcal{D}_\Phi$$

Mais d'abord, nous avons vu dans l'exemple précédent que le critère qui décide de la satisfaction ou non des propriétés de flux d'information (Δ) dépend de l'*observation* qu'un usager de bas niveau fait du système. Il serait donc logique que Δ soit défini en fonction de cette observation(Lafrance and Mullins, 2002).

Définition 4.2. Un critère d'observation est une fonction partielle $\mathcal{O} : Act* \rightarrow Act$ qui exprime le comportement d'un processus vu par un observateur. L'observation d'un processus selon un critère $\mathcal{O}$ est notée $P/\mathcal{O}$.

L'observation permet de définir des niveaux de sécurité et ce qui est observable de ces niveaux. Deux suites d'actions γ_1 et γ_2 mènent à la même observation α relativement à $\mathcal{O}$ si $\gamma_1, \gamma_2 \in \mathcal{O}^{-1}(\alpha)$. Soit un sous-ensemble $L \subseteq \Sigma$ d'actions observables, le critère d'observation $\mathcal{O}_L$ est défini comme suit :

$$\mathcal{O}_L^{-1}(\alpha) = \begin{cases} (Act\backslash L)^* \alpha (Act\backslash L)^* & \text{si } \alpha \in L \\ (Act\backslash L)^* & \text{si } \alpha = \tau \end{cases}$$

Avec cette notion, on peut définir une $\mathcal{O}$-relation(Lafrance and Mullins, 2002).

Définition 4.3 ($\mathcal{O}$-relation). Soit $\sqsubseteq$ un préordre, $\simeq$ une relation d'équivalence et $\mathcal{O}$ un critère d'observation. On dit qu'un processus P est un *$\mathcal{O}$-préordre* de Q (noté $P \sqsubseteq_{\mathcal{O}} Q$) si $P/\mathcal{O} \sqsubseteq Q/\mathcal{O}$. De même, P est *$\mathcal{O}$-équivalent* à Q (noté $P \simeq_{\mathcal{O}} Q$) si $P/\mathcal{O} \simeq Q/\mathcal{O}$.

Nous pouvons maintenant exprimer l'interférence admissible en termes d'équivalence observationnelle. Cette propriété s'appelle BNAI (*Bisimulation-based non-deterministic admissible interference*) et s'exprime formellement comme suit

$$\forall P' \text{ atteignable de } P.(P'\backslash Dwn)/\mathcal{O}_{Lo} \sqsubseteq_{\mathcal{O}_{Hi}} (P'\backslash Dwn)$$

Nous noterons $\mathcal{R}(P)$ l'ensemble des états atteignables de P. Le théorème suivant (appelé "unwinding" par les théoriciens de la sécurité) est une caractérisation algébrique de l'interférence admissible dont la démonstration complète est donnée dans (Lafrance and Mullins, 2002)

Théorème 4.1 (Théorème "unwinding" pour BNAI). *Un processus P satisfait l'interférence admissible non-déterministe basée sur la bisimulation (BNAI) si,*

$$\forall P' \in \mathcal{R}(P).P'\backslash Dwn \simeq_{\mathcal{O}_{Lo}} P'\backslash(Dwn \cup Hi)$$

La non-interférence est un cas de la BNAI où l'ensemble d'actions déclassifiantes est vide ($Dwn = \emptyset$). Dans ce cas, le membre de gauche de la propriété devient $P'\backslash Dwn = P'$, un sous-ensemble de P. On peut donc éliminer la condition "pour tout P' atteignable de P" puisque si la propriété est vraie pour P, elle le sera trivialement pour P'.

Corollaire 4.1 (Non-interférence (en termes de BNAI)). *Un processus P satisfait la non-interférence (exprimée en fonction de BNAI), si $P \simeq_{\mathcal{O}_{Lo}} P\backslash Hi$.*

Tout comme la GNDC de Focardi et Gorrieri, la BNAI permet d'exprimer plusieurs propriétés de sécurité de protocoles.

$$P_E \text{ est } BNAI^{\Phi} \Leftrightarrow \forall P'_E \in \mathcal{R}(P_E).P'_E\backslash Dwn_{\Phi} \simeq_{\mathcal{O}_{Lo_{\Phi}}} P'_E\backslash(Dwn_{\Phi} \cup Hi_{\Phi})$$

où P_E est le processus obtenu de la concurrence des agents honnêtes du protocole P et de l'ennemi générique décrit au chapitre précédent.

4.3 Spécification des propriétés de sécurité

(Lafrance and Mullins, 2002; Brlek et al., 2003) ont spécifié l'authentification, la confidentialité et l'anonymat des crypto-protocoles en utilisant la notion de BNAI. Pour chaque propriété, il suffit de bien identifier les catégories d'actions Lo, Dwn et

Hi, ainsi que le critère d'observation.

La spécification des propriétés utilise la définition suivante.

Définition 4.4. Soit m et m' des messages et $\overrightarrow{m}$ un vecteur de messages de taille variable, considéré comme un message. Et soit $\lambda \in \Lambda$ une fonction privée de SPASM. On dit que m' *contient* m (noté $m \prec m'$) si

- $m' = m$
- $m' = \lambda(\overrightarrow{m})$ et $m \prec \overrightarrow{m}$
- $m' = \overrightarrow{m}$ et $\exists m'' \in \overrightarrow{m}.m \prec m''$

et m' contient m en clair (noté $m \prec_{clair} m'$) si $m' = \overrightarrow{m}$ et $m \in \overrightarrow{m}$.

4.3.1 Authentification

Pour l'authentification, il faut montrer qu'un intrus ne peut se faire passer pour *Alice* auprès de *Bob* si telle n'est pas l'intention d'*Alice* et vice-versa. Nous avons modifié la propriété initiale de Mullins et Lafrance puisqu'elle ne tenait pas compte de l'*intention* d'*Alice* et découvrait de ce fait une série d'attaques qui n'en étaient pas en réalité(Lafrance and Mullins, 2002).

Exemple. Observons les 3 premières étapes du protocole de Woo& Lam (annexe I.2) :

1. Alice $\rightarrow$ Bob : *Alice*
2. Bob $\rightarrow$ Alice : N_b
3. Alice $\rightarrow$ Bob : $\{N_b\}_{ShK(Alice,S)}$

L'intrus peut interagir avec le protocole de la façon suivante :

1. Alice $\rightarrow$ E : *Alice*
1′. E $\rightarrow$ Bob : *Alice*
2′. Bob $\rightarrow$ E : N_b
2. E $\rightarrow$ Alice : N_b
3. Alice $\rightarrow$ E : $\{N_b\}_{ShK(Alice,S)}$
3′. E $\rightarrow$ Bob : $\{N_b\}_{ShK(Alice,S)}$
...

menant à l'authentification réussie d'*Alice* par *Bob* . Toutefois, rien dans cette séquence n'indique à qui *Alice* voulait envoyer le message au départ. Si elle a amorcé une communication avec l'intrus, alors l'authentification d'*Alice* par *Bob* constitue une faille au protocole. Par contre, si elle voulait réellement communiquer avec *Bob* et que l'intrus n'a agi que comme transmetteur des messages, alors cette séquence est correcte. Or, la propriété initiale de Mullins et Lafrance détectait cette séquence comme fautive à tout coup. ◁

Soit $\sigma \subseteq Act$, le sous-ensemble des actions authentifiant *Alice* auprès de *Bob* . Nous définissons les ensembles H, L et D d'actions de haut niveau, bas niveau et déclassifiantes respectivement pour l'authentification d'*Alice* par *Bob* :

H_{auth} = Les message envoyés sur le réseau par *Alice*
= $\bigcup_{x \in \mathcal{P} \setminus \{Bob\}, c \in \mathcal{C}, p \in dom(c)} \{c(Alice, p, x, y)\}$

D_{auth} = Les actions par lesquelles *Alice* montre son intention de communiquer avec *Bob*

$$= \bigcup_{c\in\mathcal{C},p\in dom(c)}\{c(Alice,p,Bob,x), c(Alice,p,\_,\_).Bob \prec p\}$$

L_{auth} = Les actions de *Bob* authentifiant *Alice* avec succès

$$= \{\sigma_{Bob}\}$$

En d'autres mots, l'intrus ne peut causer d'interférence menant à l'authentification d'*Alice* par *Bob* sans que ce ne soit l'intention initiale d'*Alice* . On peut maintenant exprimer directement l'authentification avec BNAI.

Corollaire 4.2 (Authentification de Alice**).** *(Lafrance and Mullins, 2002)*

$$P_E \text{ est } BNAI^{auth} \Leftrightarrow \forall P'_E \in \mathcal{R}(P_E).P'_E\backslash D_{auth} \simeq_{\mathcal{O}_{L_{auth}}} P'_E\backslash(D_{auth} \cup H_{auth})$$

Cette propriété est trivialement vraie si $D_{auth} = H_{auth} = \emptyset$, donc si *Alice* n'a envoyé aucun message sur le réseau. Or, si $L_{auth} \neq \emptyset$, donc si *Bob* croit avoir identifié *Alice* avec succès, il y attaque d'authentification puisqu'*Alice* n'a même pas participé au protocole. Il convient donc de vérifier cette condition préalablement à la vérification de la propriété.

4.3.2 Confidentialité et anonymat

Quant à la confidentialité, on en distingue deux extrêmes, la confidentialité forte et la confidentialité faible(Lafrance and Mullins, 2002).

La confidentialité faible est préservée si l'intrus n'apprend jamais le contenu d'un message confidentiel. La confidentialité forte est préservée si l'intrus ne peut discriminer le comportement du protocole normal de celui du protocole n'échangeant aucune information confidentielle. Il n'a pas à apprendre le contenu du message, mais seulement une information concernant la réussite ou l'échec de l'échange d'information confidentielle. Entre ces deux extrêmes, on peut admettre différents niveaux de confi-

dentialité, par exemple en vérifiant la confidentialité forte, mais en permettant de couler la parité du message.

Exemple. Soit le protocole suivant :

$$\begin{aligned} Alice(m) &= y := encrypt(Shared(Alice, Bob), m).'C(Bob, c1; y).C(Bob, c2; x).0 \\ Bob &= C(Alice, c1; x).m := decrypt(Shared(Alice, Bob), x).([m \text{ pair}] \\ &\quad 'C(Alice, c2; 0).0 + [m \text{ impair}]'C(Alice, c2; 1).0) \end{aligned}$$

Ce protocole satisfait la confidentialité faible puisque l'intrus ne peut apprendre le message m dédié à Bob . Par contre, la confidentialité forte n'est pas satisfaite puisque l'intrus peut apprendre la parité du message. $\triangleleft$

La différence entre les niveaux de confidentialité réside dans l'ensemble d'actions qui seront considérées comme déclassifiantes. Par définition, une action déclassifiante permet le coulage d'information entre un usager de haut niveau et un usager de bas niveau. Les différents niveaux de confidentialité dépend donc de l'information que l'on désire couler ou non.

Les actions de haut niveau sont les actions par lesquelles des principaux honnêtes envoient un message confidentiel compréhensible à l'intrus. Les actions de bas niveau sont les actions de l'intrus. Les actions de déclassification sont les communications des participants honnêtes de messages confidentiels inintelligible pour l'intrus. De plus, les participants honnêtes peuvent envoyer sur le réseau des messages ne contenant aucune donnée confidentielle. Dépendamment du niveau de confidentialité désiré, ces actions peuvent ou non être déclassifiantes. Soit $\mathcal{M}_c$ l'ensemble des messages confidentiels et $\mathcal{P}_c(m) \subseteq \mathcal{P}$ l'ensemble des participants au protocole partageant un message confidentiel $m \in \mathcal{M}_c$.

Notons d'abord qu'un message m peut se retrouver sous plusieurs formes :
– *message en clair* : m se retrouve sous forme non-encryptée. Il est donc révélé à

quiconque l'obtient.

- *message encrypté* : $m' = \{m\}_k$ est encrypté avec la clé k. Il est révélé seulement si le principal recevant le message m' connaît la clé k^{-1}.
- *message signé* : $m' = [m]_{ag}$ est signé par l'agent ag. L'intention de la signature n'est pas de cacher le message m à qui que ce soit, mais plutôt d'authentifier l'agent l'émettant. L'action de signer un message ne le déclassifie donc pas.
- *message hashé* : $m' = h(m)$ est un hash de m. m' ne révèle pas le message m.

Les ensembles d'actions de haut niveau, bas niveau et déclassifiantes pour la confidentialité faible s'expriment ainsi, avec $L \vdash k_e$ où L est l'ensemble des connaissances de l'intrus :

$H_X(m)$ = Les actions du participant X contenant m en clair comme paramètre

$= \bigcup_{m',c}\{c(X,m',*,*), c(*,m',X,*) | m \prec_{clair} m' \text{ ou } (m' = (encrypt_X(k_e,t) |\ sign_X(t)) \wedge m \prec t)\}$

$H(m)$ = Les actions révélant m en clair

$= \bigcup_{X \in \mathcal{P}_c(m)} H_X(m)$

H_{conf} = Les actions révélant un message confidentiel

$= \bigcup_{m \in \mathcal{M}_c} H(m)$

$D_X(m)$ = Les actions du participant X déclassifiant m ou ne contenant pas m en clair

$= \bigcup_{k,m',m'',c}\{c(X,m'',*,*), c(*,m'',X,*) | m \not\prec m'' \text{ ou } m'' = encrypt_X(k,t) | hast(t) \wedge m \prec t \wedge k \neq k_e\}$

$D(m)$ = Les actions déclassifiant m

$= \bigcup_{X \in \mathcal{P}_c(m)} D_X(m)$

D_{conf} = Les actions déclassifiant un message confidentiel

$= \bigcup_{m \in \mathcal{M}_c} D(m)$

$L(m)$ = Les actions révélant m observables par l'intrus ou les participants ne partageant pas le secret

$= \{f(m') | f \in Act_{E \cup (\mathcal{P} \setminus \mathcal{P}_c(m))}$ et $(m \prec_{clair} m'$ ou $L \vdash m)\}$

$L_{conf} = \bigcup_{m \in \mathcal{M}_c} L(m)$

Pour vérifier la confidentialité forte, les ensembles des niveaux d'actions sont sensiblement les mêmes sauf que les actions $\not{C} = \bigcup_{c,m''} \{c(X, m'', *, *), c(*, m'', X, *) | m \not\prec m''\} \in D_X(m)$ sont déplacées dans l'ensemble $H_X(m)$. De plus, les actions de bas niveaux sont maintenant toutes les actions observables par l'intrus et non seulement celles par lesquelles il apprend un message.

Pour une confidentialité intermédiaire entre la forte et la faible, on peut partitionner $\not{C}$ en deux sous-ensembles disjoints $\not{C}_1$ et $\not{C}_2$ tels que $\not{C}_1 \subseteq H_X(m)$ et $\not{C}_2 \subseteq D_X(m)$.

Corollaire 4.3 (Confidentialité des messages $\mathcal{M}_c$). *(Lafrance and Mullins, 2002)*

$$P_E \text{ est } BNAI^{conf} \Leftrightarrow \forall P'_E \in \mathcal{R}(P_E). P'_E \backslash D_{conf} \simeq_{\mathcal{O}_{L_{conf}}} P'_E \backslash (D_{conf} \cup H_{conf})$$

Plus récemment (Brlek et al., 2003) ont exprimé l'anonymat comme un cas de la confidentialité faible où la donnée à garder secrète est l'identificateur d'un des participants id_p. Toutefois, les messages révélant id_p en clair ne sont pas les seuls à révéler l'identité de p. Regardons pour chaque forme de message possible ce qui peut révéler l'identité :

- *message en clair* : id_p se retrouve en clair.
- *message encrypté* : $m' = \{m\}_k$ est encrypté avec la clé k. Si $k = Shared(X, p)$ est

une clé partagée entre X et p, la décryption ou l'encryption par X avec cette clé révèle qu'il connaît l'identité de p. De même si X encrypte ou décrypte un message avec la clé $k = Public(p)$, il connaît l'identité de p. Ces messages révèlent donc id_p même si $id_p \not\leq m$.

- *message signé* : $m' = [m]_{ag}$ est signé par l'agent ag. Si $ag = p$, une vérification de signature réussie sur ce message révèle l'identité de p même si $id_p \not\leq m$.
- *message hashé* : $m' = h(m)$ ne révèle l'identité de p d'aucune manière.

Soit $\mathcal{P}_{id} \subseteq \mathcal{P}$ l'ensemble des participants d'un protocole ayant accès à l'identité du participant anonyme. Les actions de haut niveau, bas niveau et déclassifiantes sont définies ainsi :

$$
\begin{aligned}
H_{anon} &= \text{Les actions révélant } id_p \text{ en clair} \\
&= H(id_p) \cup \bigcup_{X \notin \mathcal{P}_{id},m,c} \{c(p, m'', *, *) | m'' = encrypt_p(Shared(X,p), m), \\
&\quad encrypt_p(Secret(p), m),\ sign_p(m)\} \\
D_{anon} &= \text{Les actions déclassifiant } id_p \\
&= D(id_p) \cup \bigcup_{X \in \mathcal{P}_{id},m,c} \{c(p, m', *, *) | m' = encrypt_p(Shared(X,p), m)\} \\
L_{anon} &= \text{Les actions observables par l'intrus ou les participants n'ayant pas} \\
&\quad \text{droit de connaître l'identité de } p \\
&\quad L(id_p) \cup \bigcup_{X \notin \mathcal{P}_{id},m} \{encrypt_X(Shared(X,p), m), encrypt_X(Public(p), m), \\
&= decrypt_X(Shared(X,p), \{m\}_{Shared(X,p)}), \\
&\quad decrypt_X(Public(p), \{m\}_{Secret(p)}),\ checksign_X(p, [m]_p, m)\}
\end{aligned}
$$

Corollaire 4.4 (Anonymat de id_p). *(Brlek et al., 2003)*

$$P_E \text{ est } BNAI^{anon} \Leftrightarrow \forall P'_E \in \mathcal{R}(P_E).P'_E \backslash D_{anon} \simeq_{\mathcal{O}_{L_{anon}}} P'_E \backslash (D_{anon} \cup H_{anon})$$

4.3.3 Non-répudiation et équité

En nous basant sur le travail fait pour exprimer les propriétés précédentes, nous avons spécifié la non-répudiation et l'équité en termes d'interférence admissible.

La non-répudiation est satisfaite si un parti ne peut nier être à l'origine d'un message. Concrètement, cela signifie que la réception d'un message dont on peut garantir la provenance d'un des participants est toujours précédée de l'envoi de ce message par le participant en question.

La non-répudiation peut se diviser en 2 propriétés, soit la non-répudiation de l'origine (NRO) et la non-répudiation de la réception (NRR). La NRO assure que l'émetteur d'un message ne peut nier en être la source tandis que la NRR assure que le récepteur du message ne peut nier l'avoir reçu.

La notion d'interférence admissible n'est pas assez forte dans ce cas-ci puisqu'il n'est pas question d'*admettre* une interférence entre l'envoi d'un message non-répudiable et sa réception, mais plutôt de la *forcer*. Nous utiliserons donc la négation de la non-interférence pour s'assurer qu'une interférence se produit dans le système. Il n'y aura pas d'action déclassifiante, seulement des haut niveau et bas niveau.

Mais d'abord, analysons pour chaque forme de termes possible lesquels constituent des preuves d'origine que l'émetteur ne peut répudier :

- *message en clair* : aucun message en clair n'offre de garantie sur sa source. Ils peuvent avoir été rejoués ou créés de toute pièce par un intrus.
- *message encrypté* : $m' = \{m\}_k$ est encrypté avec la clé k. Il existe plusieurs cas de clés. N'importe quel participant peut encrypter avec une clé publique, donc si $k = Public(ag)$, il n'y a pas de garantie. Par contre, seul le participant concerné connaît sa clé secrète, donc si $k = Secret(ag)$ (ce qui revient à une signature pour

certains algorithmes), la provenance est garantie. De même, pour une clé partagée entre deux participants, en supposant qu'un participant peut détecter les messages qu'il a lui-même encryptés, la réception d'un message encrypté par ce type de clé assure sa provenance. Pour tout autre clé k, à moins que l'on ait d'abord vérifié la confidentialité de k, on ne peut rien garantir.

- *message signé* : $m' = [m]_{ag}$ est signé par l'agent ag. Seul ag peut signer un message, donc la réception d'un tel message garantit que ag est à la source de m ou, du moins, accepte sa valeur.
- *message hashé* : N'importe quel participant peut hasher un message qu'il possède.

Définition 4.5. Soit l'ensemble $\mathcal{NR}_X(m)$ des termes assurant la non-répudiation du message m par le participant X. $\mathcal{NR}_X(m) = \{t.(t = Sign_X(p)|\ t = Encrypt(Shared(X, ...), p)|\ t = Encrypt(Secret(X), p).m \prec p)\}$.

Pour la NRO, *Bob* veut être sûr que le message m reçu d'*Alice* ne peut être répudié.

H_{NRO} = Les actions par lesquelles *Alice* (ou un "trusted third party") envoie un message qu'elle ne peut répudier à *Bob*

$= \bigcup_{c \in \mathcal{C}, p \in dom(c)} \{c(Alice|TTP, p, *, *).m' \in \mathcal{NR}_{Alice}(m) \prec p\}$

L_{NRO} = Les actions de *Bob* prouvant la réception du message m

$= \bigcup_{\sigma \in Act_{Bob} \setminus (\mathcal{C} \cup \Lambda)} \{\sigma(p).m \prec p\}$

Corollaire 4.5 (Non-répudiation de l'origine de m par *Alice*).

$$P_E \text{ satisfait NRO} \Leftrightarrow P_E \not\approx_{\mathcal{O}_{L_{NRO}}} P_E \backslash H_{NRO}$$

La NRR, par contre, est plus faible comme propriété. Alors que la NRO requiert que le message envoyé lui-même ne puisse être répudié ($m' \in \mathcal{NR}(m)$), la NRR peut utiliser d'autres mécanismes qui prouveront la réception du message.

Exemple. Soit le protocole de non-répudiation suivant :

1. Alice $\rightarrow$ Bob : $\{n, [m, Bob]_{Alice}\}_{PK_{Bob}}$
2. Bob $\rightarrow$ Alice : n

Alice ne peut répudier avoir envoyé le message m puisque $[m, Bob]_{Alice} \in \mathcal{NR}(m)$. Par contre, *Bob* peut répudier l'envoi du nonce n, par exemple si une *Alice* malhonnête exhibe n comme preuve de réception de *Bob*, un juge n'en sera pas convaincu. Mais la réception par une *Alice* honnête de n, qui était encrypté avec la clé publique de *Bob* , est suffisante comme preuve de NRR, puisque seul *Bob* aurait pu décrypter ce message.

La NRR s'exprimera plutôt en terme d'interférence obligatoire entre la réception du message par *Bob* et le moment où *Alice* croit être certaine que *Bob* a reçu le message. Les actions seront donc partitionnées ainsi :

H_{NRR} = Les actions par lesquelles *Bob* (ou un "trusted third party") reçoit un message qu'*Alice* ne peut répudier à *Bob*

$= \bigcup_{c \in \mathcal{C}, p \in dom(c)} \{c(*, p, Bob|TTP, *).m' \in \mathcal{NR}_{Alice}(m) \prec p\}$

L_{NRR} = Les actions de *Alice* prouvant la réception par *Bob* de m

$= \bigcup_{\sigma \in Act_{Alice} \setminus (\mathcal{C} \cup \Lambda)} \{\sigma(p).m \prec p\}$

Corollaire 4.6 (Non-répudiation de la réception de m par *Bob*).

$$P_E \textit{ satisfait NRR} \Leftrightarrow P_E \not\approx_{\mathcal{O}_{L_{NRR}}} P_E \backslash H_{NRR}$$

Dans bien des cas, ce n'est pas la non-répudiation qui est problèmatique, mais plutôt l'équité de cette non-répudiation. Par exemple, si *Alice* envoie à *Bob* un message qu'elle ne peut nier avoir créé, elle voudrait recevoir une preuve irréfutable que *Bob* a bien reçu le message. L'équité est nécessaire puisque si *Alice* a affaire à un *Bob* malhonnête, celui-ci peut couper la communication après avoir reçu le message d'*Alice* et dire par la suite qu'il ne l'a pas reçu, *Alice* n'a aucun moyen de corriger la situation.

Cette propriété peut elle aussi être vérifiée par flux d'information. Un protocole est équitable pour *Alice* si lorsqu'elle envoie un message de non-répudiation à *Bob*, alors elle recevra une preuve de la part de *Bob* , même si celui-ci est malhonnête. Donc, il ne doit pas y avoir d'interférence de la part de *Bob* entre l'envoi du message par *Alice* et la réception de sa preuve. Il n'y a donc pas d'actions déclassifiantes ; on a affaire à de la non-interférence simple.

La vérification de l'équité pour *Alice* suppose les niveaux d'actions suivants :

H_{fair} = Les messages envoyés sur le réseau par *Bob* et par lesquels il se commet

$= \bigcup_{c\in\mathcal{C},p\in dom(c)}\{c(Bob,p,*,*).m' \in \mathcal{NR}_{Bob}(m) \prec p\}$

L_{fair1} = Les actions par lesquelles *Alice* envoie un message qu'elle ne peut répudier à *Bob*

$= \bigcup_{c\in\mathcal{C},p\in dom(c)}\{c(Alice,p,*,*).m' \in \mathcal{NR}_{Alice}(m) \prec p\}$

L_{fair2} = Les actions de *Alice* prouvant la réception du message m

$= \bigcup_{\sigma\in Act_{Alice}\setminus(\mathcal{C}\cup\Lambda)}\{\sigma(p).m \prec p\}$

$L_{fair} = L_{fair1} \cup L_{fair2}$

Corollaire 4.7 (Équité pour *Alice*).

$$P_E \textit{ satisfait l'équité} \Leftrightarrow P_E \simeq_{\mathcal{O}_{L_{fair}}} P_E\backslash(H_{fair})$$

Exemple. Prenons le protocole de non-répudiation optimiste de Zhou& Gollmann avec sous-protocole de recouvrement

1.	Alice $\rightarrow$ Bob	:	$f_{EOO}, [Bob, L, \{M\}_K]_{Alice}$	
2.	Bob $\rightarrow$ Alice	:	$f_{EOR}, [Alice, L]_{Bob}$	
3.	Alice $\rightarrow$ Bob	:	$f_{EOO_K}, [Bob, L, K]_{Alice}$	$evidence_{Bob}(Alice, K)$
4.	Bob $\rightarrow$ Alice	:	$f_{EOR_K}, [Alice, L, K]_{Bob}$	$evidence_{Alice}(Bob, K)$
	ou			
3'.	Alice $\rightarrow$ TTP	:	$f_{SUB}, [Bob, L, K]_{Alice}$	
4'.	Bob $\leftrightarrow$ TTP	:	$f_{CON}, [Alice, Bob, L, K]_{TTP}$	$evidence_{Bob}(Alice, K)$
5'.	Alice $\leftrightarrow$ TTP	:	$f_{CON}, [Alice, Bob, L, K]_{TTP}$	$evidence_{Alice}(Bob, K)$

Nous n'avons pas à considérer d'intrus E extérieur au protocole puisque le comportement malhonnête vient de Bob et non de l'extérieur. Nous devons vérifier la NRO de K par $Alice$ lorsqu'envoyé à Bob, la NRR de K par Bob et l'équité du protocole.

D'abord, vérifions la NRO de K de $Alice$ vers Bob. Les actions de haut niveau H_{NRO} sont l'envoi des messages 3 et $3'$ contenant les valeurs signées, tandis que les actions de bas niveau sont les deux actions $evidence_{Bob}(Alice, K)$. Ne considérant que ces actions, nous avons le processus suivant : $\tau * .3.(\tau * .evidence_{Bob}(Alice, K).\tau * +3'.(\tau * .evidence_{Bob}(Alice, K).\tau * |\tau*))$, ou, en niveaux d'actions : $Hi.(Lo + Hi.Lo)$. La propriété de NRO est donc satisfaite puisque les actions de bas niveau sont le résultat d'une interférence. Quant à la NRR de K par Bob, les actions de haut niveau sont la réception des messages 3 et $3'$ et les actions de bas niveau sont les deux actions de $Alice$, $evidence_{Alice}(Bob, K)$. Les traces donnent alors $\tau * .3.(\tau * .evidence_{Alice}(Bob, K).0 + 3'.(\tau * .evidence_{Alice}(Bob, K)|\tau*))$, ou, en niveaux d'actions $Hi.(Lo + Hi.Lo)$ qui respecte aussi l'interférence. NRR est satisfaite.

Maintenant, intéressons-nous à l'équité. L'envoi du message 4 est une action de haut niveau, tandis que les actions de bas niveau sont l'envoi de 3, $3'$, de même que les deux actions $evidence_{Alice}(Bob, K)$. Le processus, après observation des actions concernées devient : $\tau * .3.(\tau * .4.\tau * .evidence_{Alice}(Bob, K) + 3'.(\tau * |\tau * .evidence_{Alice}(Bob, K)))$

ou, en niveau d'actions : $Lo.(Hi.Lo + Lo.Lo)$. L'équité est donc satisfaite puisque peu importe que *Bob* ait envoyé son évidence de réception ou non, *Alice* reçoit cette évidence de la part du TTP. Par contre, si on élimine le sous-protocole de recouvrement, le processus en niveau d'actions devient simplement $Lo.Hi.Lo$. L'équité n'est alors plus respectée puisque $Lo.Hi.Lo \not\simeq_{\mathcal{O}_{Lo}} Lo$.

Cette formule ne fonctionne pas pour l'équité pour *Bob* . Les actions de haut niveau et de bas niveau sont respectivement $(3, 3')$ et $(4, evidence_{Bob}(Alice, K)$ donnant le processus $\tau*.3.(\tau*.evidence_{Bob}(Alice, K).\tau*.4.\tau*+3'.(\tau*.evidence_{Bob}(Alice, K)|\tau*))$ ou en niveau d'actions $Hi.(Lo.Lo + Hi.Lo)$, ce qui ne satisfait clairement pas la propriété. ◁

Cette dernière particularité montre la limite de l'expression de l'équité. Elle n'est valide que pour le participant qui se commet le premier. L'équité a d'ailleurs moins de sens pour l'autre participant puisque les actions par lesquelles il se commet succèdent à la réception de sa preuve. Cette façon d'exprimer l'équité oblige toutefois à pouvoir identifier clairement le participant à risque, soit *Alice* dans l'exemple.

4.4 Conclusion

Nous avons dans ce chapitre introduit la notion de flux d'information et décrit quelques propriétés ainsi exprimables. Plus spécifiquement, l'interférence admissible a servi de gabarit pour exprimer des propriétés de sécurité comme l'authentification, la confidentialité, l'anonymat, la non-répudiation et l'équité. D'une propriété à l'autre, seul le niveau de sécurité des actions change, la spécification demeure la même. Mais jusqu'à présent, nous ne nous sommes intéressés qu'aux membres de gauche et de droite de l'expression des propriétés, sans expliquer en quoi consiste la relation $\simeq_{\mathcal{O}_L}$. Le chapitre suivant précise la relation les liant, soit la bisimulation symbolique.

CHAPITRE 5

VÉRIFICATION

Nous allons maintenant définir la relation $\simeq_{\mathcal{O}}$ utilisée pour lier les membres de droite et de gauche des propriétés de sécurité d'un système SPASM : une bisimulation symbolique. Une bisimulation est une relation d'équivalence sur deux systèmes qui exige qu'en tout temps en cours d'exécution, les comportements possibles des deux systèmes soient identiques.

Nous avons vu comment une sémantique opérationnelle symbolique permettait d'exprimer sous forme de graphe de transitions fini un système dont le graphe serait autrement infini. Une équivalence sur des graphes symboliques doit être telle que deux graphes symboliques sont équivalents si et seulement si les graphes standards sous-jacents le sont.

La première démarche dans ce sens a été effectuée par (Hennessy and Lin, 1995). Ils ont développé une bisimulation pour des systèmes symboliques correspondant à une algèbre CCS avec passage de valeurs. Nous utiliserons une démarche semblable pour formuler une bisimulation symbolique associée à la sémantique opérationnelle de SPASM, représentant la sémantique non-symbolique SPPA décrite dans (Lafrance and Mullins, 2003b).

5.1 Bisimulation standard de SPPA

Une bisimulation entre deux systèmes $P1$ et $P2$ exige que chacun des systèmes simule l'autre à tout moment, donc que pour toute action exécutable par $P1$, $P2$ puisse faire

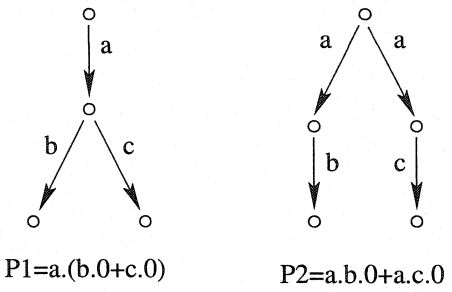


FIGURE 5.1 Exemple de processus non-bisimilaires

de même et vice versa.

Exemple. Les processus de la figure 5.1 ne sont pas bisimilaires, puisque après l'exécution de l'action a, le processus $P1$ peut encore choisir entre les actions b et c alors qu'il ne reste qu'une seule possibilité au processus $P2$ selon la branche choisie. Toutefois, $P1$ simule $P2$, puisque après l'action a, peu importe l'action exécutée par $P2$, $P1$ peut faire de même. $P2$ ne simule pas $P1$ puisque si $P2$ prend la branche de gauche en exécutant l'action a et que $P1$ décide d'exécuter l'action c, $P2$ se trouve bloqué.

L'exemple précédent illustre le concept de bisimulation pour deux processus CCS simples. Il en va de même pour des processus décrits par une algèbre avec passage de valeurs où les données échangées font aussi partie de l'action et doivent donc être considérées lors de la comparaison de deux actions.

La différence principale entre les systèmes de transitions symbolique et non-symbolique vient de la description des variables à chaque noeud. Alors que SPASM associe à chaque processus une valuation symbolique φ reliant chaque variable à un terme symbolique, SPPA leur associe une valuation non-symbolique $\rho : \mathcal{V} \rightarrow \mathcal{T}^{\setminus}$, qui relie les variables à un terme clos non-symbolique, où $\mathcal{T}^{\setminus}$ est le sous-ensemble des termes de $\mathcal{T}$ obtenu sans les constructeurs $Choice$ et Var.

Définition 5.1. Soit $\langle s, \varphi \rangle$ un noeud d'un système de transitions symbolique et s_ρ un noeud d'un système de transitions standard. Nous définissons la relation $\models$ comme suit :

$$\rho \models \varphi \Leftrightarrow \forall v \in dom(\rho).\rho(v) \in [\![v]\!]_{\overline{\varphi}}$$

Nous disons que s_ρ *correspond* à $\langle s, \varphi \rangle$ si et seulement si $\rho \models \varphi$.

La bisimulation est, par définition, une relation sur les termes clos de l'algèbre. Dans notre cas, un terme clos de l'algèbre est un noeud s_ρ, u_ϱ d'un graphe de transitions. Soit $\mathcal{S}$ l'ensemble des termes de l'algèbre. Nous définirons donc une relation paramétrée sur les valuations non-symboliques ρ, ϱ, $\mathcal{R}^{\rho,\varrho} \subseteq \mathcal{S} \times \mathcal{S}$. Nous utilisons les 3 types d'actions suivants : $c(x)$ pour les canaux externes, $\beta(x,y) ::= (y := \lambda(x))$ ou $(y := \lambda_{gen}(-))$ pour les fonctions privées et génératrices, $\delta(p1, x, p2, y)$ pour les marqueurs.

De plus, puisque la relation que nous devons définir $\simeq_{\mathcal{O}}$ est une relation observationnelle (voir chapitre 4), nous utiliserons les actions $s_\rho \stackrel{\alpha}{\Longrightarrow} s'_\rho$ pour signifier que α est l'action visible de la transition dont la trace réelle est $s_\rho \stackrel{\tau}{\longrightarrow}^* \stackrel{\alpha}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* s'_\rho$.

Définition 5.2. Soit ρ et ϱ deux valuations non-symboliques. La relation $\mathcal{R}^{\rho,\varrho} \subseteq \mathcal{S} \times \mathcal{S}$ est une (ρ, ϱ)-simulation si lorsque $(s, u) \in \mathcal{R}^{\rho,\varrho}$ alors

- $s_\rho \stackrel{c(x)}{\Longrightarrow} s'_\rho$ implique que $u_\varrho \stackrel{c(y)}{\Longrightarrow} u'_\varrho$ pour un u' tel que $\rho(x) = \varrho(y)$ et $(s', u') \in \mathcal{R}^{\rho,\varrho}$.
- $s_\rho \stackrel{\beta(x,z)}{\Longrightarrow} s'_{\rho[z \mapsto \lambda(x)]}$ implique que $u_\varrho \stackrel{\beta(y,w)}{\Longrightarrow} u'_{\varrho[w \mapsto \lambda(y)]}$ pour un u' tel que $\rho(x) = \varrho(y)$ et $(s', u') \in \mathcal{R}^{\rho[z \mapsto \lambda(x)], \varrho[w \mapsto \lambda(y)]}$.
- $s_\rho \stackrel{\delta(p1,x,p2,z)}{\Longrightarrow} s'_{\rho[z \mapsto x]}$ implique que $u_\varrho \stackrel{\delta(p1,y,p2,w)}{\Longrightarrow} u'_{\varrho[w \mapsto y]}$ pour un u' tel que $\rho(x) = \varrho(y)$ et $(s', u') \in \mathcal{R}^{\rho[z \mapsto x], \varrho[w \mapsto y]}$.

Une (ρ, ϱ)-simulation est une (ρ, ϱ)-bisimulation si $(\mathcal{R}^{\rho,\varrho})^{-1}$ est une (ϱ, ρ)-simulation.

Définition 5.3. Deux systèmes non symboliques P_1 et P_2, ayant pour états initiaux s_0 et u_0 respectivement, sont bisimilaires, noté $P_1 \stackrel{*}{\sim} P_2$, et $\mathcal{R} = \{\mathcal{R}^{\rho,\varrho} | \rho, \varrho \in [\mathcal{V} \rightarrow$

Soit φ et φ' des valuations symboliques et ρ une valuation non-symbolique. Alors

$$\begin{aligned}
\langle s, \varphi \rangle \xrightarrow{\alpha(x)} \langle s', \varphi' \rangle \quad &\Leftrightarrow \quad s_\rho \xrightarrow{\alpha(x)} s'_\rho \\
&\text{pour } \rho,\ \varphi' \text{ tels que } \rho \models \varphi'. \\
\langle s, \varphi \rangle \xrightarrow{\beta(x,y)} \langle s', \varphi' \rangle \quad &\Leftrightarrow \quad s_\rho \xrightarrow{\beta(x,y)} s'_{\rho[y \mapsto \lambda(x)]} \\
&\text{pour } \rho,\ \varphi' \text{ tels que } \rho[y \mapsto \lambda(x)] \models \varphi'. \\
\langle s, \varphi \rangle \xrightarrow{\delta(a1:x,a2:y)} \langle s', \varphi' \rangle \quad &\Leftrightarrow \quad s_\rho \xrightarrow{\delta(a1:x,a2:y)} s'_{\rho[y \mapsto x]} \\
&\rho,\ \varphi' \text{ tels que } \rho[y \mapsto x] \models \varphi'.
\end{aligned}$$

FIGURE 5.2 Équivalence des sémantiques opérationnelles SPPA et SPASM

$\mathcal{T}\backslash]\}$ est une *bisimulation* si, pour tout ρ, ϱ, $(s_0, u_0) \in \mathcal{R}^{\rho,\varrho}$ et $\mathcal{R}^{\rho,\varrho}$ est une (ρ, ϱ)-bisimulation.

5.2 Bisimulation symbolique de SPASM

Nous établissons maintenant cette définition dans un cadre symbolique. Mais d'abord, il faut trouver une relation permettant d'associer un système de transitions standard à un système symbolique.

Proposition 5.1. Les règles de la figure 5.2 permettent d'associer un système SPASM à un système SPPA en trouvant des transitions équivalentes dans les deux systèmes.

Preuve. Trivial, étant donnée la définition de $\rho \models \varphi$.

Exemple. Pour illustrer le passage d'une bisimulation standard à symbolique, considérons la figure 5.3. Chacun des deux graphes sous-tend 3 valuations non-symboliques ρ selon la valeur de la variable x. On peut facilement vérifier que $s_{0\rho} \overset{\star}{\sim} u_{0\rho}$ pour tout ρ. Une bisimulation symbolique $\overset{\star}{\approx}$ de ces deux graphes devrait donc donner le même résultat.

Toutefois, l'équivalence est loin d'être triviale. Par exemple, la transition $\langle s_0, \varphi_0 \rangle \xrightarrow{a(x)} \langle s_1, \varphi_0 \rangle$ ne peut être associée à la transition $\langle u_0, \varphi_0 \rangle \xrightarrow{a(x)} \langle u_1, \varphi_0 \rangle$ puisque pour cer-

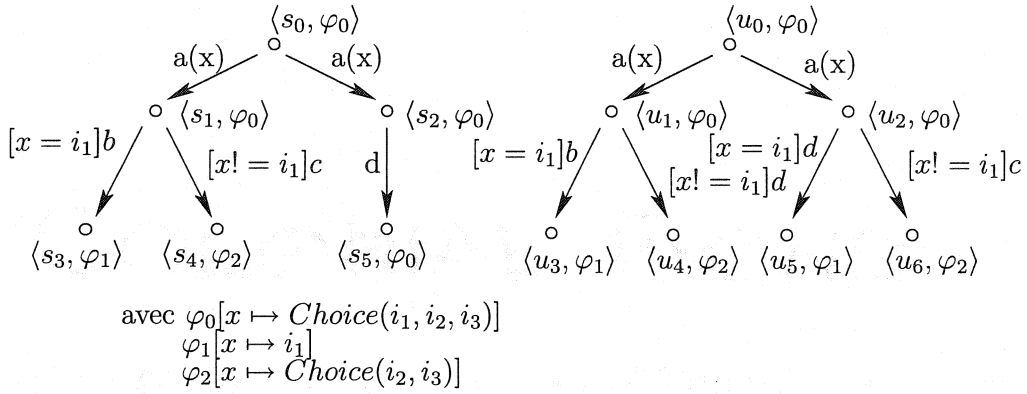


FIGURE 5.3 Graphes de transition SPASM

taines valeurs, s_1 et u_1 ne se comportent pas de la même façon. La situation est semblable avec la transition $\langle u_0, \varphi_0 \rangle \stackrel{a(x)}{\Longrightarrow} \langle u_2, \varphi_0 \rangle$. Nous ne devons pas pour autant conclure à la non-équivalence de ces deux graphes. Pour certaines valeurs, $s_1 \stackrel{\star}{\approx} u_1$ tandis que pour d'autres, $s_1 \stackrel{\star}{\approx} u_2$.

Nous paramétrerons donc l'équivalence symbolique sur les valuations φ. Nous aurons $s_1 \stackrel{\star^{\varphi}}{\approx} u_1$ lorsque $\varphi = \varphi_1$ et $s_1 \stackrel{\star^{\varphi}}{\approx} u_2$ lorsque $\varphi = \varphi_2$. Un raisonnement similaire donne $s_2 \stackrel{\star^{\varphi_1}}{\approx} u_2$ et $s_2 \stackrel{\star^{\varphi_2}}{\approx} u_1$. Toutefois, comment cela nous permet-il de conclure que $s_0 \stackrel{\star}{\approx} u_0$ pour une valuation φ_0 ?

φ_0 peut être divisée en deux valuations, φ_1 et φ_2, telles que $\varphi_1 \vee \varphi_2 = \varphi_0$. Pour chacune, on trouve, pour chaque transition depuis s_0, une transition partant de u_0 vers des états bisimilaires. La valuation φ_0 est donc couverte en entier, donc $s_0 \stackrel{\star}{\approx} u_0$

. $\triangleleft$

De façon générale, soit φ et ψ des valuations symboliques. Si nous partons de noeuds $\langle s, \varphi \rangle$ et $\langle u, \psi \rangle$ et que nous pouvons diviser φ en un ensemble $\Phi = \{\varphi'\}$ tel que $\varphi = \bigvee \Phi$ et que pour chaque $\varphi' \in \Phi$, nous pouvons trouver un noeud tel que $s_j \stackrel{\star^{\varphi'}}{\approx} u_k$, alors nous pouvons conclure que $s \stackrel{\star}{\approx} u$.

Ceci motive la définition d'une famille de relations $\mathcal{B}^{\varphi,\psi} \subseteq \mathcal{S} \times \mathcal{S}$, paramétrée sur les valuations symboliques.

Définition 5.4. Soit φ, φ_a, ψ et ψ_a des valuations symboliques. La relation $\mathcal{B}^{\varphi,\psi} \subseteq \mathcal{S}\times\mathcal{S}$ est une (φ,ψ)-simulation si lorsque $(s,u) \in \mathcal{B}^{\varphi,\psi}$ alors, lorsque $\langle s,\varphi\rangle \stackrel{\alpha}{\Longrightarrow} \langle s',\varphi_a\rangle$, il existe une collection de valuations symboliques Φ telle que $\varphi_a \subseteq \bigvee \Phi$ et pour chaque $\varphi' \in \Phi$, il existe une transition $\langle u,\psi\rangle \stackrel{\alpha'}{\Longrightarrow} \langle u',\psi_a\rangle$ telle que

- si α est de la forme $c(x)$, alors $\alpha' = c(y)$ et $[\![x]\!]_{\overline{\varphi_a}} \cap [\![y]\!]_{\overline{\psi_a}} \supseteq [\![x]\!]_{\overline{\varphi'}} = [\![y]\!]_{\overline{\psi'}}$ et $(s',u') \in \mathcal{B}^{\varphi',\psi'}$.
- si α est de la forme $\beta(x,z)$, alors $\alpha' = \beta(y,w)$ et $[\![x]\!]_{\overline{\varphi_a}} \cap [\![y]\!]_{\overline{\psi_a}} \supseteq [\![x]\!]_{\overline{\varphi'}} = [\![y]\!]_{\overline{\psi'}}$, $[\![z]\!]_{\overline{\varphi_a}} \cap [\![w]\!]_{\overline{\psi_a}} \supseteq [\![z]\!]_{\overline{\varphi'}} = [\![w]\!]_{\overline{\psi'}}$ et $(s',u') \in \mathcal{B}^{\varphi',\psi'}$.
- si α est de la forme $\delta(p1,x,p2,z)$, alors $\alpha' = \delta(p1,y,p2,w)$ et $[\![x]\!]_{\overline{\varphi_a}} \cap [\![y]\!]_{\overline{\psi_a}} \supseteq [\![x]\!]_{\overline{\varphi'}} = [\![y]\!]_{\overline{\psi'}}$, $[\![z]\!]_{\overline{\varphi_a}} \cap [\![w]\!]_{\overline{\psi_a}} \supseteq [\![z]\!]_{\overline{\varphi'}} = [\![w]\!]_{\overline{\psi'}}$ et $(s',u') \in \mathcal{B}^{\varphi',\psi'}$.

Une (φ,ψ)-simulation est une (φ,ψ)-bisimulation si $(\mathcal{B}^{\varphi,\psi})^{-1}$ est une (ψ,φ)-simulation.

Définition 5.5. Deux systèmes symboliques P_1 et P_2, dont les états initiaux sont respectivement s_0 et u_0, sont bisimilaires, noté $P_1 \stackrel{*}{\approx} P_2$, et $\mathcal{B} = \{\mathcal{B}^{\varphi,\psi} | \varphi,\psi \in [\mathcal{V} \rightarrow \mathcal{T}]\}$ est une *bisimulation symbolique* si, pour tout φ,ψ, $(s_0,u_0) \in \mathcal{B}^{\varphi,\psi}$ et $\mathcal{B}^{\varphi,\psi}$ est une (φ,ψ)-bisimulation.

Le reste de cette section montre la correspondance entre la bisimulation standard pour un système SPPA et la bisimulation symbolique qui en découle pour un système SPASM.

Proposition 5.2. Nous définissons un ensemble de relations indexées sur les valuations non-symboliques, $\mathcal{R}_\mathcal{B} = \{\mathcal{R}_\mathcal{B}^{\rho,\varrho}\}$, telle que $\mathcal{R}_\mathcal{B}^{\rho,\varrho} = \{(s,u) | \exists\varphi,\psi.\rho \models \varphi, \varrho \models \psi$ et $(s,u) \in \mathcal{B}^{\varphi,\psi}\}$. Si $\mathcal{B}$ est une bisimulation symbolique, alors $\mathcal{R}_\mathcal{B}$ est une bisimulation.

Preuve. Soit $(s,u) \in \mathcal{R}_\mathcal{B}^{\rho,\varrho}$, ie $(s,u) \in \mathcal{B}^{\varphi,\psi}$ pour un certain φ,ψ tels que $\rho \models \varphi$ et $\varrho \models \psi$. Nous devons montrer que les systèmes partant de s et u sont simulables et inversement.

- Soit $s_\rho \overset{c(x)}{\Longrightarrow} s'_\rho$. Par la proposition 5.1, il s'ensuit qu'il doit exister une transition $\langle s, \varphi\rangle \overset{c(x)}{\Longrightarrow} \langle s', \varphi_a\rangle$ telle que $\rho \models \varphi_a$. Il existe donc un ensemble de valuations symboliques Φ tel que $\varphi_a \subseteq \bigvee \Phi$ et pour chaque $\varphi' \in \Phi$, il existe une transition $\langle u, \psi\rangle \overset{c(y)}{\Longrightarrow} \langle u', \psi_a\rangle$ telle que $[\![x]\!]_{\overline{\varphi_a}} \cap [\![y]\!]_{\overline{\psi_a}} \supseteq [\![x]\!]_{\overline{\varphi'}} = [\![y]\!]_{\overline{\psi'}}$ et $(s', u') \in \mathcal{B}^{\varphi', \psi'}$. Puisque $\rho \models \varphi_a$ et $\varphi_a \subseteq \bigvee \Phi$, il doit y avoir un $\varphi' \in \Phi$ tel que $\rho \models \varphi'$. Puisque $\varrho \models \psi$, alors ϱ est tel que $\rho(x) = \varrho(y)$ et $\varrho \models \psi'$ correspondant, donc $\varrho \models \psi_a$. Encore par la proposition 5.1, nous savons qu'il existe une transition $u_\varrho \overset{c(y)}{\Longrightarrow} u'_\varrho$ pour le u' associé avec ce φ'. Puisque $\rho \models \varphi'$ et $\varrho \models \psi'$ et $(s', u') \in \mathcal{B}^{\varphi', \psi'}$, il s'ensuite que $(s', u') \in \mathcal{R}_{\mathcal{B}}^{\rho, \varrho}$.
- Le raisonnement est similaire pour les cas où $s_\rho \overset{\beta(x,z)}{\Longrightarrow} s'_\rho$ et $s_\rho \overset{\delta(p1,x,p2,z)}{\Longrightarrow} s'_\rho$. □

Proposition 5.3. Inversement, on construit un ensemble de relations indexées sur les valuations symboliques $\mathcal{B}_\mathcal{R}$, telle que $\mathcal{B}_\mathcal{R}^{\varphi,\psi} = \{(s,u) | \rho \models \varphi$ implique $(s,u) \in \mathcal{R}^{\rho,\varrho}$ pour un ϱ tel que $\varrho \models \psi\}$. Si $\mathcal{R}$ est une bisimulation, alors $\mathcal{B}_\mathcal{R}$ est une bisimulation symbolique.

Preuve. Soit $(s,u) \in \mathcal{B}_\mathcal{R}^{\varphi,\psi}$. Il faut montrer que les systèmes partant de s et u sont simulables et inversement

- Supposons $\langle s, \varphi\rangle \overset{c(x)}{\Longrightarrow} \langle s', \varphi_a\rangle$. Nous devons montrer qu'il existe un ensemble de valuations symboliques Φ tel que $\varphi_a \subseteq \bigvee \Phi$ et que $\forall \varphi' \in \Phi$, il existe une transition $\langle u, \psi\rangle \overset{c(y)}{\Longrightarrow} \langle u', \psi_a\rangle$ telle que $[\![x]\!]_{\overline{\varphi_a}} \cap [\![y]\!]_{\overline{\psi_a}} \supseteq [\![x]\!]_{\overline{\varphi'}} = [\![y]\!]_{\overline{\psi'}}$ et $(s', u') \in \mathcal{B}_\mathcal{R}^{\varphi', \psi'}$.
  Soit l'ensemble $U = \{u' | \langle u, \psi\rangle \overset{c(y)}{\Longrightarrow} \langle u', \psi_{au'}\rangle\}$. Pour chaque $u' \in U$, nous définissons des valuations $\varphi_{u'}$ et $\psi_{u'}$ telles que $[\![x]\!]_{\overline{\varphi_a}} \cap [\![y]\!]_{\overline{\psi_{au'}}} \supseteq [\![x]\!]_{\overline{\varphi_{u'}}} = [\![y]\!]_{\overline{\psi_{u'}}}$ et $\rho \models \varphi_{u'} \Leftrightarrow (s', u') \in \mathcal{R}^{\rho,\varrho}$ pour un ϱ tel que $\varrho \models \psi_{u'}$. Maintenant, soit un ensemble $\Phi' = \{\varphi_{u'} | u' \in U\}$.
  Nous vérifions d'abord que $\varphi_a \subseteq \bigvee \Phi'$. Il suffit de montrer que $\rho \models \varphi_a$ implique $\rho \models \varphi_{u'}$ pour un $\varphi_{u'} \in \Phi'$. Puisque $\rho \models \varphi_a$, $\rho \models \varphi$[1] donc $(s,u) \in \mathcal{R}^{\rho,\varrho}$ pour un ϱ et, par la proposition 5.1, il y a une transition $s_\rho \overset{c(x)}{\Longrightarrow} s'_\rho$. Il existe donc une transition

[1] La sémantique opérationnelle de SPASM est telle que pour chaque transition $\langle s, \varphi\rangle \overset{\alpha}{\Longrightarrow} \langle s', \varphi_a\rangle$, $\varphi_a \subseteq \varphi$, donc si un ρ est tel que $\rho \models \varphi_a$, alors $\rho \models \varphi$.

$u_\varrho \overset{c(y)}{\Longrightarrow} u'_\varrho$ telle que $\rho(x) = \varrho(y)$ et $(s', u') \in \mathcal{R}^{\rho,\varrho}$. Toujours par la proposition 5.1, on sait qu'il doit exister un ψ_a tel que $\varrho \models \psi_a$ et $\langle u, \psi\rangle \overset{c(y)}{\Longrightarrow} \langle u', \psi_a\rangle$, avec $u' \in U$. Puisque $(s', u') \in \mathcal{R}^{\rho,\varrho}$, il s'ensuit que $\rho \models \varphi_{u'}$ ($\varrho \models \psi_{u'}$). Nous avons donc que $\varphi_{u'} \in \Phi'$. Par conséquent, $\rho \models \varphi_a$ implique $\rho \models \varphi_{u'}$ pour un $\varphi_{u'} \in \Phi'$, donc $\varphi_a \subseteq \bigvee \Phi'$. Et puisque $\rho \models \varphi_{u'}$ et $(s', u') \in \mathcal{R}^{\rho,\varrho}$ pour un ϱ, il s'ensuit que $(s', u') \in \mathcal{B}_{\mathcal{R}}^{\varphi_{u'},\psi_{u'}}$.

Maintenant, pour chaque $\varphi_{u'} \in \Phi'$, il existe une transition $\langle u, \psi\rangle \overset{c(y)}{\Longrightarrow} \langle u', \psi_{au'}\rangle$ pour laquelle $[\![x]\!]_{\overline{\varphi_a}} \cap [\![y]\!]_{\overline{\psi_{au'}}} \supseteq [\![x]\!]_{\overline{\varphi_{u'}}} = [\![y]\!]_{\overline{\psi_{u'}}}$ et $\rho \models \varphi_{u'} \Leftrightarrow (s', u') \in \mathcal{R}^{\rho,\varrho}$ pour un ϱ tel que $\varrho \models \psi_{u'}$. Donc, $\rho \models \varphi_{u'}$ implique que $(s', u') \in \mathcal{R}^{\rho,\varrho}$ pour un ϱ tel que $\varrho \models \psi_{u'}$. Donc $(s', u') \in \mathcal{B}_{\mathcal{R}}^{\varphi_{u'},\psi_{u'}}$ pour chaque $\varphi_{u'} \in \Phi'$.

Les preuves précédentes montrent que Φ' est le Φ recherché au départ. $\mathcal{B}_{\mathcal{R}}$ est donc une bisimulation symbolique.

- Le raisonnement est similaire pour les cas où $\langle s, \varphi\rangle \overset{\beta(x,z)}{\Longrightarrow} \langle s', \varphi_a\rangle$ et $\langle s, \varphi\rangle \overset{\delta(p1,x,p2,z)}{\Longrightarrow} \langle s', \varphi_a\rangle$. □

Nous venons de prouver que pour tout système de transitions obtenu d'une algèbre de processus SPPA, il existe une correspondance symbolique en SPASM. Nous avons réduit la vérification d'équivalence de deux systèmes de transitions à celle de systèmes de transitions symboliques. Cette dernière équivalence est plus difficile à vérifier à cause de la nécessité de décomposer la valuation d'un système en plusieurs valuations.

La section suivante présente un algorithme permettant de vérifier automatiquement la bisimulation symbolique de deux systèmes SPASM.

5.3 L'algorithme

La façon la plus simple d'aborder la bisimulation est de la considérer comme une relation symétrique de simulation. Un algorithme naïf de bisimulation entre deux

systèmes p et q serait de calculer la simulation entre p et q, puis celle entre q et p. Rappelons que le contexte d'application de la bisimulation est la vérification de propriétés de sécurité exprimées en termes de non-interférence admissible. Or les deux systèmes à vérifier sont des sous-ensembles d'un processus initial et un sens de la bisimulation est trivial. Notre problème revient donc à vérifier une seule simulation.

L'algorithme utilise la définition suivante qui permettra de ne considérer que les valeurs des variables intéressantes et non celles de toutes les variables du système.

Définition 5.6. $\varphi^{|\alpha}$ est une *restriction du domaine* de la valuation φ sur l'action α telle que $dom(\varphi^{|\alpha}) = fv(\alpha)$.

Pour simplifier l'écriture, nous supposons que des changements de variables ont été effectués pour s'assurer de la concordance des valuations des deux systèmes. Par exemple, soit deux transitions à comparer $\langle s, \varphi \rangle \stackrel{c(x)}{\Longrightarrow} \langle s', \varphi' \rangle$ et $\langle u, \psi \rangle \stackrel{c(y)}{\Longrightarrow} \langle u', \psi' \rangle$, la deuxième transition devient $\langle u, \psi_\varphi \rangle \stackrel{c(x)}{\Longrightarrow} \langle u', \psi'_\varphi \rangle$ où $\psi_\varphi(x) = \psi(y)$ et $\psi'_\varphi(x) = \psi'(y)$.

La figure 5.4 décrit un algorithme qui, prenant en entrée deux noeuds $\langle s, \varphi \rangle$ et $\langle u, \psi \rangle$, détermine les valeurs pour lesquelles ces deux noeuds sont similaires. Pour ce faire, nous utilisons deux structures auxiliaires :

- $Sim : \mathcal{S} \times \mathcal{S} \rightarrow [\mathcal{V} \rightarrow \mathcal{T}]$. Contient la valuation pour laquelle s simule u.
- $Partial : \mathcal{S} \rightarrow [\mathcal{V} \rightarrow \mathcal{T}]$. Associe à chaque terme algébrique s du premier graphe une valuation pour laquelle une similarité a été trouvée.

Définition 5.7. G_{s_0} représente le *graphe de transitions*, ou système, dont l'état initial est s_0 et composé de noeuds $\langle s, \varphi \rangle$ et de transitions $(s, \alpha, s') \in (\mathcal{S} \times Act \times \mathcal{S})$.

L'algorithme fait une recherche en profondeur à partir des noeuds $\langle s, \varphi \rangle$ et $\langle u, \psi \rangle$. Pour chaque transition possible depuis s, la fonction *match* calcule la valuation pour

```
1   match(⟨s, φ⟩ ⟹^α ⟨s', φ'⟩, ⟨u, ψ⟩) =
2       if ∄⟨u, ψ⟩ ⟹^α ⟨u', ψ'⟩
3           return (φ\φ')
4       else
5           simul_on_α :=
6               for each ⟨u, ψ⟩ ⟹^α' ⟨u_j, ψ_j⟩
7                   next_φ := φ'|α ∧ ψ_j|α'
8                   φ_j := close(⟨s', next_φ⟩, ⟨u_j, next_φ⟩)
9               ∨_j φ_j
10          no_trans_for_α := φ|α \ φ'|α
11          return (no_trans_for_α ∨ simul_on_α)


12  close(⟨s, φ⟩, ⟨u, ψ⟩) =
13      if (s, u) ∈ dom(Sim)
14          return Sim(s, u)
15      else if {⟨s, φ⟩ ⟹^α ⟨s', φ'⟩} = ∅
16          Partial(s) := Partial(s) ∨ φ
17          Sim(s, u) := φ
18          return φ
19      else
20          Sim(s, u) := φ
21          for each t ∈ {⟨s, φ⟩ ⟹^α ⟨s', φ'⟩}
22              φ_t := match(t, ⟨u, ψ⟩)
23          φ_s := ⋀_t φ_t
24          Partial(s) := Partial(s) ∨ φ_s
25          Sim(s, u) := φ_s
26          return (φ_s)
```

FIGURE 5.4 Algorithme de bisimulation symbolique de SPASM

laquelle les deux systèmes G_s et G_u sont similaires sur une transition, tandis que la fonction *close* retourne la valuation pour laquelle ils sont similaires au total.

La fonction principale est la fonction *close*. Pour chaque paire d'états $(\langle s, \varphi\rangle, \langle u, \psi\rangle)$, elle vérifie qu'elle n'a pas déjà été visitée (ligne 13). Sinon, elle cherche la valuation pour laquelle G_u simule G_s. Si aucune transition n'est possible depuis s, alors forcément, il y a simulation pour toute valeur de φ (ligne 15).

Pour chaque transition $\langle s, \varphi\rangle \stackrel{\alpha}{\Longrightarrow} \langle s', \varphi'\rangle$ depuis s, la fonction *match* essaie de trouver les transitions correspondantes depuis u. S'il n'y en a pas, alors il n'y a pas de similarité pour la valuation φ' (ligne 2). Sinon, pour chaque transition depuis u, l'algorithme calcule la valuation pour laquelle les états cibles sont similaires (ligne 6). L'union de ces valuations constitue celle pour laquelle cette transition de s trouve une équivalence depuis u (ligne 9).

L'intersection des valuations obtenues pour toutes les transitions partant de s est celle pour laquelle il y a similarité entre les systèmes partant de $\langle s, \varphi\rangle$ et $\langle u, \psi\rangle$ (ligne 23).

On détermine si oui ou non il y a simulation complète des 2 graphes si, $Sim(s_0, u_0) = \varphi_0$ et, pour chaque noeud $\langle s, \varphi\rangle$ atteignable depuis s_0, $Partial(s) = \varphi$.

Les propositions suivantes énoncent des propriétés de l'algorithme de simulation.

Proposition 5.4 (Terminaison et complexité de l'algorithme). Si s_0 et u_0 sont les états initiaux de deux graphes symboliques finis G_{s_0} et G_{u_0}, alors la procédure $close(\langle s_0, \varphi_0\rangle, \langle u_0, \psi_0\rangle)$ termine dans un temps $\mathcal{O}(n^2)$ du nombre de noeuds.

Preuve. L'algorithme appelle récursivement la fonction *close* qui prend en paramètre deux noeuds $\langle s, \varphi\rangle$ et $\langle u, \psi\rangle$. On est intéressé à savoir s'il y a similitude entre les termes s et u, donc si le graphe G_s, simule le graphe G_u.

La condition d'arrêt la plus forte de la procédure *close* est que les termes s et u aient

déjà été vérifiés donc $(s, u) \in dom(Sim)$ (l'absence de transitions depuis s entraîne trivialement l'arrêt).

Soit $\mathcal{S}_1 = \{s|\langle s, \varphi\rangle \in G_{s_0}\}$ et $\mathcal{S}_2 = \{u|\langle u, \psi\rangle \in G_{u_0}\}$. Les graphes étant finis, les ensembles $\mathcal{S}_1$ et $\mathcal{S}_2$ sont aussi finis. Notons que le domaine de la fonction Sim est ici $\mathcal{S}_1 \times \mathcal{S}_2$, c'est donc un domaine fini. L'algorithme terminera donc forcément puisque chaque appel récursif à *close* augmente la taille du domaine de la fonction Sim.

Quant à la complexité, si G_{s_0} comprend n_1 noeuds et G_{u_0} en a n_2, au pire cas, chacun des noeuds de G_{s_0} devra être vérifié avec tous les noeuds de G_{u_0}. La complexité est donc de $n_1 * n_2$, ou $\mathcal{O}(n^2)$. □

La proposition suivante montre la correction de l'algorithme (cohérence et complétude) qui dit calculer la (bi)simulation symbolique de deux systèmes selon la définition 5.4.

Proposition 5.5 (Correction de l'algorithme). Si l'exécution de la fonction $close(\langle s_0, \varphi_0\rangle, \langle u_0, \psi_0\rangle)$ retourne φ_0 et $Sim(s_0, u_0) = \varphi_0$ et $\forall\langle s, \varphi\rangle \in G_{s_0}.Partial(s) = \varphi$, alors $(s_0, u_0) \in \mathcal{B}^{\varphi_0,\psi_0}$ et vice versa.

Preuve. La preuve de cette proposition se trouve en annexe II. □

5.4 Conclusion

Ceci conclut la partie théorique de notre étude. Nous avons vu comment spécifier un protocole sous forme d'algèbre de processus spécialisée, puis comment spécifier et vérifier les propriétés requises de ce protocole. Ces résultats sont une extension symbolique de l'approche développée dans (Lafrance and Mullins, 2003b), sans toutefois atteindre le niveau de symbolisme de (Lafrance and Mullins, 2003a), encore en développement. Les chapitres suivants étudient l'applicabilité d'une telle approche de vérification par le biais d'une implémentation et de quelques études de cas.

CHAPITRE 6

ASPiC

Le principal intérêt des méthodes de model-checking est de pouvoir vérifier automatiquement sur un modèle d'un système la satisfaction ou non de propriétés. Nous avons donc implanté un outil, *ASPiC*, utilisant la méthode par flux d'information décrite dans les chapitres précédents. *ASPiC* signifie *Analyse Symbolique automatisée de Protocoles Cryptographiques* et permet de vérifier les différentes propriétés mentionnées au chapitre 4. Les sections qui suivent présentent l'outil, mais d'abord, un peu d'histoire.

6.1 Historique du *BNAI Analyzer*

Comme nous l'avons vu, les différentes propriétés de sécurité s'expriment en fonction de la propriété BNAI. La satisfaction de cette propriété sur des systèmes simples est vérifiée automatiquement grâce à l'outil *Bnai analyzer* version 0.2 développée par (Hains and Bergeron, 2002).

Cette première version publique de l'outil[1] vérifie la propriété de BNAI sur des systèmes à états finis simples décrits avec la syntaxe de CCS. À partir de la syntaxe d'entrée, l'outil génère un système de transitions étiqueté déclassifiant, possiblement récursif. Le niveau de sécurité des actions est spécifié directement dans la syntaxe, en nommant les actions "h...", "l..." et "d..." pour des actions respectivement de haut niveau, bas niveau et déclassifiante ; ces niveaux sont donc statiques

[1] http ://www.univ-orleans.fr/SCIENCES/LIFO/Members/ghains/bnai.html

pour la vérification. L'algorithme de vérification de BNAI par bisimulation est ensuite appliqué sur le système de transitions obtenu.

Le *BNAI analyzer* version 0.2 se limite à l'étude de systèmes finis. Par contre, les systèmes concrets répondant à ce critère de finitude sont plutôt rares. La plupart des systèmes nécessitent le passage de données par valeur et dès que l'on a une valeur d'un domaine infini, par exemple un entier, ce système n'est plus vérifiable.

La vérification *symbolique* apporte une solution à ce problème, en plus de réduire de beaucoup la taille de certains systèmes finis.

(Paroux, 2003) a fait un pas de plus vers une implémentation symbolique de BNAI en implantant une sémantique CCS avec passage de valeurs. Toutefois, les variables doivent avoir un domaine borné et la modélisation de ces processus aplatit le système pour éliminer les variables. La vérification se fait donc sur un système de transitions régulier plutôt que symbolique.

L'avantage que procure cette nouvelle version du *BNAI analyzer* est surtout du point de vue de l'utilisateur de l'outil qui peut utiliser des variables plutôt qu'écrire explicitement le système de transitions pour chacune des valeurs concrètes possibles. Elle ne change en rien le problème de la taille du système engendré, surtout s'il faut le synchroniser avec un processus intrus généré automatiquement.

Au début du présent projet de recherche (janvier 2003), nous avons tenté d'implanter naïvement la théorie à partir de SPPA, avant son virage symbolique. Le résultat fut désastreux : la modélisation d'un protocole des plus simples, tel NSPK, prenait plusieurs minutes et générait une quantité d'états qui ne laissait rien présager de bon pour les protocoles plus complexes.

Nous proposons une autre extension de la version 0.2 du *BNAI analyzer*, nommée

ASPiC, qui va tirer avantage de la méthode symbolique. Ce nouvel outil sera dédié exclusivement à la vérification des crypto-protocoles. Toute la conception tiendra compte de ce contexte.

6.2 Conception

ASPiC a été programmé en langage CAML[2], un langage fonctionnel typé statiquement particulièrement bien adapté à l'implantation de concepts mathématiques.

Le choix de ce langage de programmation vient d'abord du fait que le *BNAI analyzer* original était lui-même écrit en CAML ; il est donc naturel que l'extension utilise le même langage. De plus, *ASPiC* n'est qu'un prototype pour lequel l'implémentation des bons algorithmes prime sur les performances. Or les structures de CAML permettent l'implantation assez directe des définitions proposées dans les chapitre précédents, sans "traduction" en structures secondaires ou définition d'un grand nombre de classes comme ça aurait été le cas en Java ou autre langage objet.

Le développement de l'outil s'est fait selon un mode spirale, c'est-à-dire le raffinement de la théorie par la pratique et de la pratique par la théorie : la théorie développée initialement a été implantée, les premiers tests ont montré certaines lacunes qui ont permis d'améliorer la théorie, qui a été réimplantée en conséquence, etc. La facilité de modification du programme est donc de mise. Encore une fois, les structures simples de CAML et son mécanisme très efficace de gestion d'exceptions ont été des avantages indéniables.

Quant aux algorithmes implantés, nous n'avons tenté aucune optimisation. Ils sont implantés dans leur forme la plus naïve (donc la plus simple) possible.

[2]Pour plus d'informations sur CAML, voir http ://caml.inria.fr

6.2.1 Division en modules

ASPiC est divisé en 3 modules distincts :

- Le parseur : Ce module lit un protocole spécifié selon la syntaxe décrite au chapitre 3. À cette étape les agents ne sont pas encore instanciés. Le seul ajout à cette syntaxe est la déclaration des canaux, qui sont typés et doivent donc être déclarés en début de fichier de la manière suivante :
  Canal := let chan $cname(type_1, ..., type_n)$ in
  où $type_i$ = msg | $nonce$ | $number$ | $enc(type_1, ..., type_n)$
  | key | any | $agent$ | $sign(type_1, ..., type_n)$
  | $hash(type_1, ..., type_n)$
- Le modéliseur : Ce module instancie les agents puis synchronise les principaux du protocole, avec un intrus s'il y a lieu tel que décrit par la sémantique du chapitre 3. Les processus sont traduits en système de transitions $\mathcal{STS} = \{s_0, \delta, \Phi\}$ où
  - s_0 est l'état initial du système,
  - $\delta \subseteq \mathcal{S} \times Act \times \mathcal{S}$ est l'ensemble des transitions où $\mathcal{S}$ est un ensemble fini d'états correspondant à des termes de l'algèbre et Act est l'ensemble des actions correspondant aux préfixes,
  - $\Phi : \mathcal{S} \rightarrow [\mathcal{V} \rightarrow \mathcal{T}]$ est une fonction qui associe à chaque état du système la valuation φ obtenue en appliquant la sémantique opérationnelle.

  Deux noeuds $\langle s_1, \varphi_1 \rangle$ et $\langle s_2, \varphi_2 \rangle$ sont dit équivalents si pour chaque acteur x du protocole, $s_{1x} = s_{2x}$, $dom(\varphi_1) = dom(\varphi_2)$ et pour chaque $y \in dom(\varphi_i).\varphi_1(y) = \varphi_2(y)$.

  Notons que les variables sont évaluées de façon paresseuse. C'est-à-dire, à moins que les règles de sémantique ne disent le contraire, les variables ne sont pas évaluées au moment de la modélisation, mais plus tard, lors de la vérification des propriétés de sécurité. Par exemple, soit $\varphi[x_{Alice} \mapsto v]$ où $v \in \mathcal{M}$. *Alice* envoie la variable x à *Bob* qui la reçoit dans y. La valuation résultante est $\varphi[x_{Alice} \mapsto v][y_{Bob} \mapsto x_{Alice}]$

et non $\varphi[x_{Alice} \mapsto v][y_{Bob} \mapsto v]$. Ceci garde le lien de causalité entre les variables et facilite les opérations sur le modèle (restriction et observation).

- Le vérificateur : Ce module implante l'algorithme de bisimulation décrit au chapitre 5. L'utilisateur spécifie la (ou les) propriété(s) à vérifier et dans chaque cas, les actions du système de transitions prennent le niveau approprié (haut, bas ou déclassifiant) tel que décrit au chapitre 4.

6.2.2 Spécification des propriétés

L'utilisateur d'*ASPiC* doit spécifier les propriétés à vérifier selon la syntaxe suivante :

- Authentification : *authentication of* Agent_1 *by* Agent_2 *on* α

  Spécifie la vérification de l'authentification d'un agent honnête $Agent_1$ par un agent honnête $Agent_2$. L'action α est l'action de $Agent_2$ qui marque une authentification réussie. Elle doit donc être présente dans la spécification du protocole sous la forme $\alpha(Agent_1)$. Le paramètre est important puisqu'il indique avec qui l'authentification s'est faite. La vérification de l'authentification se fait sur un sous-ensemble du système de transitions ne contenant que les états à partir desquels il existe un chemin tel qu'une action $\alpha(Agent_1)$ est atteignable. La taille du système à vérifier diminue de beaucoup sans éliminer de possibilités. En effet, c'est cette seule action qui est visible après partitionnement des actions. Les chemins laissés de côté n'aurait été que des séquences de τ invisibles.
- Confidentialité : *[strong|weak] confidentiality* type t[3] (;type t)* *for* Agent_1 (;Agent_2)*

  Spécifie que les termes t restent un secret partagé uniquement par les agents $Agent_i$. Nous spécifions aussi le niveau de confidentialité désiré, forte ou faible, la valeur

[3]type t := *msg* _msgname | *nonce* _noncename | *number* entier | *key* _keyname | *var* _varname principalname | *agent* principalname | *Public*(agentname|principalname) | *Secret*(agentname|principalname) | *Shared*(agentname|principalname,agentname|principalname)

par défaut étant faible. Cette propriété ne nécessite l'ajout d'aucune action particulière, les niveaux d'actions dépendant uniquement des paramètres et non de la nature de l'action elle-même. La vérification s'effectue sur l'ensemble du système de transitions puisque le succès du protocole n'est pas nécessaire pour qu'un bris de confidentialité se produise.

- Anonymat : *anonymity* principalname *for* Agent_1 (;Agent_2)*

  Spécifie l'anonymat de l'acteur *principalname* dont l'identité ne peut être connue que des agents $Agent_i$. La vérification est semblable à celle de la confidentialité.

- Non-répudiation : *non repudiation of* type t (;type t)* *from* Agent_1 *to* Agent_2 *[with* Agent_3*] [fair]*

  Spécifie que l'agent $Agent_1$ ne peut répudier l'envoi à l'agent $Agent_2$ des termes t (NRO)[4]. Cette vérification nécessite l'ajout au protocole d'une action de l'$Agent_2$ prouvant la réception des termes t de l'$Agent_1$. Cette action doit donc avoir en paramètres tous les termes t. Cette action constitue le succès du protocole, donc comme pour l'authentification, le système de transitions à vérifier est un sous-ensemble du système initial ne contenant que les états à partir desquels il existe un chemin tel qu'une action de non-répudiation est possible. L'option $fair$ spécifie que le protocole doit être équitable, donc $Agent_2$ doit envoyer à $Agent_1$ une preuve non-répudiable qu'il a reçu les termes t. Une action telle que décrite précédemment pour l'$Agent_2$ doit donc être présente pour l'$Agent_1$. Lorsque cette option est présente, on vérifie aussi la NRR. L'option *with* $Agent_3$ permet de spécifier qu'un des participants a le droit d'envoyer un message au nom de $Agent_1$, par exemple un tiers-parti fiable (TTP).

[4]Les termes t de la liste doivent être liés, c'est-à-dire envoyés dans un même message. Si deux termes peuvent être non-répudiés séparément, il faut le spécifier en deux temps.

6.2.3 Autres options

D'autres options sont aussi disponibles à l'utilisateur d'*ASPiC* :

- *no ennemy* : Modélise un protocole sans le synchroniser avec l'intrus. Cette option est utile pour vérifier qu'il n'y ait pas d'erreur dans la syntaxe du protocole ou simplement s'assurer de son bon fonctionnement dans un environnement fermé.
- *talk to honest* : Interdit aux acteurs du protocole d'initier volontairement une session avec l'intrus. L'intrus fait toutefois bien partie du protocole.
- *E knows* | *E doesn't know* type t (; type t)* : Permet respectivement d'ajouter des termes aux connaissances initiales par défaut de l'intrus ou d'en éliminer.
- *E cannot play* Agent_1(; Agent_2)* : Indique que l'intrus ne peut jouer le rôle des agents mentionnés. Cette option est différente de *trusted* en ce sens que l'intrus peut intercepter ou jouer des messages en tant que ces agents, mais n'est pas autorisé à servir en tant que ces agents. Par exemple, un intrus ne peut jouer le rôle d'un agent *Banque*. Si on lui permettait de jouer ce rôle, des clients pourraient avoir un compte auprès de lui et il pourrait autoriser des transactions. Ce serait une banque corrompue.

6.2.4 Abstractions et simplifications

Plusieurs abstractions et simplifications étaient de mise pour diminuer la taille du système de transitions résultant et, conséquemment, rendre relativement raisonnable le temps de vérification.

D'abord, nous avons diminué l'entrelâcement des actions en exécutant en priorité et comme un bloc les actions simples (qui ne sont pas des canaux) des différents acteurs. Cette simplification est justifiée par le fait que l'exécution des actions simples par un principal se fait indépendamment de ce qui se passe chez les autres acteurs et ces

actions ne sont visibles que par l'exécuteur. C'est donc dire qu'il ne peut non plus y avoir d'interférence entre ces actions puisque l'interférence implique soit un contact entre deux acteurs, ce qui n'est clairement pas le cas ici, soit un changement de niveau d'action, qui ne peut se produire à moins qu'un acteur ne veuille se tromper lui-même.

Exemple. Soit deux processus
$alice = ...\_y := encrypt(Public(Bob), \_x).\_hy := hash(\_y)....$
$bob = ...\_m := decrypt(Secret(Bob), \_y).\_sigm := sign(\_m)....$
Il existe 6 possibilités d'entrelâcement d'actions avec ces deux processus. Or, dans chaque cas, l'observation résultante pour chaque acteur demeure la même, soit
$\mathcal{O}_{alice} = \_y := encrypt(Public(Bob), \_x).\_hy := hash(\_y)$,
$\mathcal{O}_{bob} = \_m := decrypt(Secret(Bob), \_y).\_sigm := sign(\_m)$
et pour tout autre observateur x, $\mathcal{O}_x = \tau$. Les 6 séquences d'entrelâcement sont donc équivalentes. ◁

De plus, nous ne considérons par défaut qu'une seule session par participant honnête. Plus de sessions, pour un même rôle ou un rôle différent, doit être explicité au moment de l'instanciation des acteurs, par exemple en écrivant $(Alice|Bob, richard)$, qui signifie que l'acteur *richard* joue une session dans le rôle d'*Alice* et une session en tant que *Bob*. L'intrus peut de son côté participer à autant de sessions qu'il le désire.

6.2.5 Diagnostic d'erreur

Ce n'est pas tout d'implanter un outil capable de détecter des failles dans des protocoles, encore faut-il qu'on puisse en rendre compte. Or, lors de la vérification par bisimulation standard, il est très diffile de mettre le doigt sur la partie erronée du système lorsqu'un état a été détecté comme non-bisimilaire. On dispose uniquement des états des deux systèmes qui font défaut et l'action pour laquelle ils ne sont pas

similaires. L'attaque ayant causé cette erreur peut se trouver dans un autre partie du système, il n'y a pas moyen de le savoir.

C'est encore pire pour la bisimulation symbolique : on ne peut cibler une action causant le manque de bisimilarité puisqu'il se peut qu'elle ait son équivalent depuis un autre état. Nous ne disposons que des valeurs des variables (φ) pour lesquelles la bisimilarité fait défaut et ce, uniquement après le parcours complet des deux graphes. Nous ne tenterons pas ici de résoudre ce problème, nous allons seulement fournir assez d'informations à l'utilisateur pour qu'il puisse cerner le défaut.

Le diagnostic consiste en une représentation du système sur lequel l'erreur a été détectée, qu'on peut aussi visualiser sous forme graphique. La version graphique du système erroné est une aide précieuse à sa correction puisqu'on peut y voir aisément les niveaux des actions, ainsi que les états à partir desquels une bisimulation n'a pu être trouvée. On dispose aussi, pour chaque état, des valuations φ n'ayant pas trouvé de bisimilitude. À partir de ces informations et du graphe du système, un usager peut cibler l'erreur.

Exemple. La figure 6.1 montre un exemple de graphique de sortie de *ASPiC* pour la vérification de l'authentification du protocole NSPK. Les transitions foncées représentent les actions de haut niveau et la grise, une action de bas niveau. Les transitions pâles sont des τ. Il n'y a pas d'action déclassifiante. Les états représentés par un gros point noir sont ceux pour lesquels une bisimilarité complète n'a pu être trouvée.

L'étude plus approfondie des transitions et des valeurs pour cette transition permet de voir qu'une faille existe avec l'action $init(alice : \_msg1_{alice} \rightarrow e : k3_e$, et la valeur $\_msg1_{alice} = \{n_{alice}, alice\}_{PK(e)}$. Donc, dès le début du protocole, l'initiation d'une session entre *Alice* et l'intrus peut occasionner une attaque d'authentification. $\triangleleft$

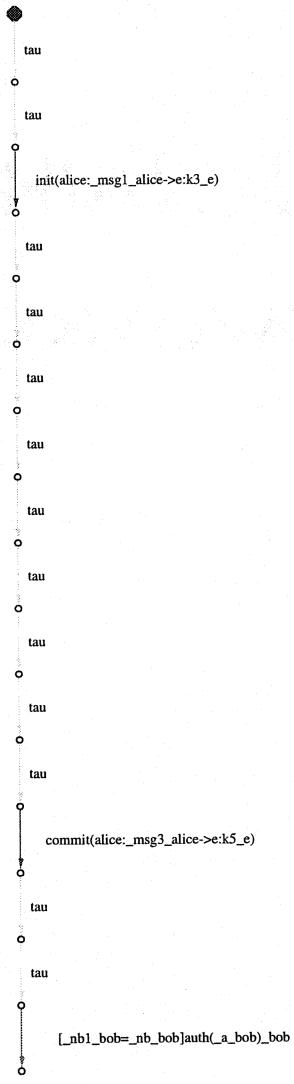


FIGURE 6.1 Exemple de graphe de sortie de *ASPiC*

6.3 Conclusion

L'outil *ASPiC*, une extension du *BNAI Analyzer*, développé en CAML, a été implanté pour mettre à l'épreuve la méthode de flux d'information symbolique basée sur BNAI. Cet outil est conçu pour vérifier symboliquement différentes propriétés de sécurité des crypto-protocoles dans un environnement hostile. Le chapitre suivant présente des études de cas exécutées avec *ASPiC*.

CHAPITRE 7

ÉTUDES DE CAS

Ce chapitre présente les différents tests effectués à l'aide d'*ASPiC*, d'abord sur des protocoles de sécurité simples, puis sur des protocoles de commerce électronique. Nous terminons ce chapitre par une comparaison des résultats obtenus avec *ASPiC* et ceux d'autres outils de model-checking.

7.1 Protocoles simples

Afin de tester individuellement chacune des propriétés, l'outil a été exécuté sur des protocoles défaillants simples, dont les résultats sont connus et attendus.

Les tableaux 7.1 et 7.2 présentent les résultats obtenus. La colonne "résultat" indique si la vérification a trouvé une faille ou non. Le nombre d'états total est le nombre d'états générés lors de la modélisation du processus. Le nombre d'états vérifiés est le nombre d'états contenus dans le sous-système sur lequel l'algorithme de vérification de la propriété a été effectué. Les temps mentionnés sont d'abord le temps pris pour la modélisation complète du protocole, puis le temps total incluant la vérification. Les tests ont été effectués sur un CPU Intel Pentium 4 2.8 MHz avec 512 Mo de mémoire vive.

[1]l'option "talk to honest" a été ajoutée à la modélisation. De plus, l'intrus ne peut envoyer sa propre identité à *Bob* .

[2]La modélisation a été ajoutée de l'option "talk to honest".

[3]avec l'option "talk to honest"

TABLEAU 7.1 Résultats obtenus en appliquant *ASPiC* sur des protocoles simples d'authentification et de confidentialité

Propriété / protocole	Résultat	nb états total	nb états à vérifier	temps(sec.) modèle/total
Authentification ($A \rightarrow B$: A authentifie B)				
Woo & Lam	non($Bob \rightarrow Alice$)	2693	39	2.06/2.28
Woo & Lam corrigé	oui($Bob \rightarrow Alice$)	2548	21	2/2.09
NSPK	non($Bob \rightarrow Alice$) oui($Alice \rightarrow Bob$)	531	32 17	0.18/0.23
NSPK corrigé	oui($Bob \rightarrow Alice$) oui($Alice \rightarrow Bob$)	522	17 17	0.2/0.22
Grenouille à Grande Bouche	non($Bob \rightarrow Alice$)	423	45	0.1/0.15
GGB corrigé	oui($Bob \rightarrow Alice$)	393	24	0.09/0.15
Confidentialité (f|$F, m; A, B$: message m est un secret (f :faible, F :fort) entre A et B)				
NSPK	non($f, N_a; Alice, Bob$) non($f, N_b; Alice, Bob$)	531	550 550	0.19/10.98
NSPK corrigé	non($f, N_a; Alice, Bob$) non($f, N_b; Alice, Bob$)	522	541 541	0.10/11.33
NSPK corrigé 2[1]	oui($f, N_a; Alice, Bob$) oui($f, N_b; Alice, Bob$)	142	151 151	0.06/1.24
Grenouille à Grande Bouche	non($f, m; Alice, Bob$) non($f, kab; Alice, Bob$)	423	385 385	0.11/5.52
GGB corrigé	non($f, m; Alice, Bob$) non($f, kab; Alice, Bob$)	393	355 355	0.08/5.83
GGB corrigé 2[2]	oui($f, m; Alice, Bob$) oui($f, kab; Alice, Bob$)	203	184 184	0.04/2.51
TMN[3]	non($f, kb; A, B, S$)	1972	1043	3.58/43
Exemple 4.6	oui($f, m; Alice, Bob$) non($F, m; Alice, Bob$)	46	48 48	0.01/0.31

TABLEAU 7.2 Résultats obtenus en appliquant *ASPiC* sur des protocoles de non-répudiation simples

Propriété / protocole	Résultat	nb états total	nb états à vérifier	temps(sec.) modèle/total
Non-répudiation ($m : A \rightarrow B \| fair - A$: message m non-répudié par A lorsqu'envoyé à B, ou protocole équitable pour A)				
Zhou & Gollmann avec TTP[4]	oui($k : Alice \rightarrow Bob$) oui($fair - Alice$) oui($k : Bob \rightarrow Alice$) non($fair - Bob$)	4375	43 43 43 43	6.59/8.85
Zhou & Gollmann optimiste	oui($k : Alice \rightarrow Bob$) non($fair - Alice$) oui($k : Bob \rightarrow Alice$)	86570	2944 713 1048	3450/4100
Zhou & Gollmann sans TTP[5]	oui($k : Alice \rightarrow Bob$) non($fair - Alice$) oui($k : Bob \rightarrow Alice$)	565	22 21 21	0.33/0.48

Nous voyons que les résultats obtenus sont conformes à ceux attendus pour l'authentification et la non-répudiation. Par contre, pour la confidentialité (et par conséquent, l'anonymat, qui n'a pas été testé ici, mais qui utilise sensiblement les mêmes fonctions) les résultats sont faux, à moins de passer par des acrobaties de modélisation.

De tous les tests faits pour la non-répudiation, seul l'équité pour *Alice* du protocole optimiste donne un résultat différent de celui attendu. La cause en est une abstraction au niveau de la modélisation. Le principal TTP n'est pas explicitement décrit comme serveur, donc si l'intrus a déjà amorcé une session, même erronée, avec le TTP, ce dernier n'est plus disponible pour *Alice* . On a affaire plutôt à du déni de service qu'à une attaque à l'équité. En ajoutant la propriété de *serveur* au TTP, on règlerait cette attaque. Par contre, on augmenterait de beaucoup la taille du modèle à vérifier, et, conséquemment, le temps de modélisation.

[4]Le protocole a été modifié de celui en annexe en ce que les messages envoyés sont des encryption avec clé privé de l'envoyeur plutôt que des signatures.

[5]C'est le protocole optimiste sans la partie recouvrement.

De plus, les tests effectués avec des protocoles de non-répudiation montrent que l'action par laquelle le participant récepteur marque sa conviction d'avoir reçu un message non-répudiable ne doit pas être gardée. Si elle l'est, alors il y aura bisimulation pour certaines valeurs (celles ne passant pas le test) et une erreur sera détectée alors que ce n'est pas le cas. En évitant ce genre d'action gardée, on élimine cette source d'erreur.

Les attaques détectées pour la propriété de confidentialité, sont, quant à elles, plutôt dues à l'expression de cette propriété. En effet, les attaques trouvées sur les protocoles corrigés se produisent dans les cas où *Alice* décide de communiquer avec l'intrus. Il est donc normal qu'il y ait atteinte à la confidentialité puisque telle est l'intention d'*Alice*. Il faudrait idéalement tenir compte de cette intention et éliminer, soit directement du système initial à vérifier, soit en les changeant en τ, les actions résultant de cette intention. Nous avons réussi à obtenir le résultat désiré en empêchant *Alice* de communiquer avec l'intrus, mais par le fait même, nous limitons les comportements possibles de l'intrus. Mais ceci est bien beau lorsque les attaques sur un protocole sont déjà documentées, mais pour un nouveau protocole, ce type de limitation pourrait empêcher la détection d'attaques réelles. Il vaudrait mieux modifier la propriété pour prendre en compte ces "attaques" qui n'en sont pas.

7.2 Protocoles de commerce électronique

Maintenant que les propriétés ont été prouvées correctes pour des exemples de protocoles simples, nous pouvons appliquer l'outil à des protocoles de commerce électronique plus complexes.

1.	C → M	:	Initiation	
			$h(C, CompteClient), Cert_C$	$(cid, Cert_C)$
2.	M → C	:	Facture	
			$M, \{h(common)\}_{SK_M}$	$(M, sigm)$
3.	C → M	:	Paiement	
			$\{\{prix, h(common), CompteClient\}_{PK_B}\}_{SK_C}$	$(sigSlip)$
4.	M → B	:	Requête d'autorisation	
			$M, sigSlip, sigm, Cert_C$	
5.	B → M	:	Réponse d'autorisation	
			$\{Y/N, C, M, h(common)\}_{SK_B}$	$(sigb)$
6.	M → C	:	Confirmation	
			$sigb$	

FIGURE 7.1 Abstraction du protocole 3kp

7.2.1 Vérification du protocole 3KP

7.2.1.1 Abstraction de 3KP

À notre connaissance, très peu de travaux de model-checking s'intéressent au protocole iKP, malgré son utilisation commerciale. Nous tenterons de le vérifier. Toutefois la version complète du protocole (annexe I.12) est trop complexe pour être vérifiable par l'outil. Nous avons donc abstrait le protocole pour en réduire le nombre de données échangées, sans perdre de fonctionnalités. La figure 7.1 présente cette version abstraite pour 3kp où $common = (prix, M, cid, C)$ et $Cert_C$ est le certificat de la clé publique du client, signé par une autorité compétente. On suppose que les deux partis connaissent a priori la clé publique de la banque. Le client connaît aussi celle du marchand duquel il achète.

Les étapes du protocoles sont les suivantes :

1. Le client initie la transaction. Ce dernier se crée un identificateur à partir de son nom et son numéro de compte et l'envoie au marchand, de même qu'un

certificat pour sa clé publique.

2. On suppose que les deux partis se sont préalablement entendus sur le contenu et le prix de l'échange. Le marchand envoie la facture signée au client.
3. Le client vérifie la signature du message et la correspondance du $h(common)$. Il crée ensuite son paiement en encyptant pour la banque, puis signant les informations sur la transaction ainsi que son numéro de compte. Il envoie son paiement au marchand.
4. Le marchand vérifie la signature du client puis envoie à la banque le paiement du client ainsi que la facture.
5. La banque vérifie d'abord les signatures du marchand et du client, puis décrypte le paiement du client. Elle s'assure de la concordance des $h(common)$ entre les deux participants et avec l'information qu'elle a. Advenant un échec à cette vérification, elle envoie au marchand une réponse négative signée. Dans le cas contraire, elle effectue le paiement et envoie le résultat de l'opération signé.
6. Le marchand vérifie la signature de la banque et transfère au client la réponse reçue.

7.2.1.2 Modèle SPASM de 3KP

Nous modélisons ensuite le protocole 3KP décrit à la section précédente avec la syntaxe de l'algèbre SPASM. La figure 7.2 représente le processus *client* du protocole dont les connaissances initiales sont un numéro de compte et le prix sur lequel il s'est entendu avec le marchand. La figure 7.3 montre le processus *marchand* connaissant initialement le prix de la transaction et la figure 7.4 présente le processus *banque*, un agent fiable qui connaît implicitement le marchand et le client et peut ainsi, si toutes les conditions de la transaction sont remplies, non-déterministiquement choisir d'accepter ou refuser la transaction.

```
let Klient (number _can,number _price) = C(Autority,return;_enccert).
        _id,_cert :=decrypt(Secret(Klient),_enccert).
        [_id=Klient]_cid :=hash(Klient,_can).
        'C(Marchand,initiate;_cid,_cert).C(Marchand,invoice;_idm,_sigm).
        [_idm=Marchand]_hcommonm,_hvm :=decrypt(Public(Marchand),_sigm).
        _hcommon :=hash(_price,_idm,_cid,_hvm).[_hcommon=_hcommonm]
        _encslip :=encrypt(Public(Bank),_price,_hcommon,_can,Klient).
        _sigslip :=encrypt(Secret(Klient),_encslip).
        'C(Marchand,payment;_sigslip).C(Marchand,confirm;_v,_siga).
        _yn,_idc,_idmb,_hcommona :=decrypt(Public(Bank),_siga).
        [_cid=_idc][_idmb=Marchand][_hcommona=_hcommon]_hvc :=hash(_v).
        [_hvc=_hvm]done(Bank,_yn,_idc).auth(Marchand).Zero
in
```

FIGURE 7.2 Description du client en SPASM

```
let Marchand (number _price) = C(Klient,initiate;_cid,_certc).
        _pkc :=decrypt(Public(Autority),_certc)._v :=newnonce().
        _hv :=hash(_v)._hcommon :=hash(_price,Marchand,_cid,_hv).
        _sigm :=encrypt(Secret(Marchand),_hcommon,_hv).
        'C(Klient,invoice;Marchand,_sigm).C(Klient,payment;_sigslip).
        _encslip :=decrypt(_pkc,_sigslip).proof(_pkc,_encslip).
        'C(Bank,authrequest;Marchand,_sigslip,_sigm,_certc).
        C(Bank,authresponse;_siga).
        _yn,_idc,_idm,_hcommona :=decrypt(Public(Bank),_siga).[_idm=Marchand]
        [_idc=_cid][_hcommona=_hcommon]receipt(Bank,Marchand_yn).
        'C(Klient,confirm;_v,_siga).Zero
in
```

FIGURE 7.3 Description du marchand en SPASM

```
let Trusted Bank = C(Marchand,authrequest;_idm,_sigslip,_sigm,_certc).
        _hcommonm,_hv :=decrypt(Public(_idm),_sigm).
        _pkc :=decrypt(Public(Autority),_certc)._encslip :=decrypt(_pkc,_sigslip).
        _price,_hcommonc,_can,_idc :=decrypt(Secret(Bank),_encslip).
        [_hcommonm=_hcommonc]_cida :=hash(_idc,_can).
        _hcommon :=hash(_price,_idm,_cida,_hv).
           (_siga :=encrypt(Secret(Bank),0,_cida,_idm,_hcommon).
           'C(Marchand,authresponse;_siga).Zero
        + [_hcommon=_hcommonm]autorise(_idc,_price,_can).
           autorise(_idm,_hcommonm).authc(_idc).authm(_idm).
           _siga :=encrypt(Secret(Bank),1,_cida,_idm,_hcommonm).
           'C(Marchand,authresponse;_siga).Zero )
in
```

FIGURE 7.4 Description de la banque en SPASM

R1. non repudiation of number 0; var _cida bank from Bank to Klient;
number 1; var _cida bank from Bank to Klient,
R2. non repudiation of number 0; agent marc from Bank to Marchand;
number 1; agent marc from Bank to Marchand,
R3. non repudiation of var _price kli; var _can kli from Klient to Bank,
R4. non repudiation of var _hcommon marc from Marchand to Bank,
R5. non repudiation of var _encslip kli from Klient to Marchand,
R6. authentication of Marchand by Klient on auth,
R7. authentication of Klient by Bank on authc,
R8. authentication of Marchand by Bank on authm,

FIGURE 7.5 Spécification des requis de 3kp pour *ASPiC*

7.2.1.3 Vérification des propriétés

Les auteurs des protocoles iKP affirment que la version 3KP satisfait les requis suivants, de façon irréfutable :

- B1. Preuve de l'autorisation de la transaction du client à la banque.
- B2. Preuve de l'autorisation de la transaction du marchand à la banque.
- M1. Preuve de l'autorisation de la transaction de la banque au marchand.
- M2. Preuve de l'autorisation de la transaction du client au marchand.
- C1. Impossibilité d'un paiement non-autorisé par le client.
- C2. Preuve de l'autorisation de la transaction de la banque au client.
- C3. Authentification du marchand par le client.
- C4. Réception d'un reçu du marchand au client.

De plus, certaines versions prévoient l'anonymat du client pour le marchand.

Les propriétés de la figure 7.5 seront vérifiées par *ASPiC* afin de s'assurer la satisfaction des requis du protocole. La vérité des propriétés $R1$ et $R2$ assure la satisfaction des requis $C2$ et $M1$ respectivement. La propriété $R3$ correspond au requis $B1$ puisque par l'association dans un même message du numéro du compte du client et du prix, le client signifie son intention de débiter son compte du montant demandé. De même, $R4$ correspond au requis $B2$ puisque le marchand se commet à la transaction dont les données ont servi à calculer le $hash(common)$ et $R5$ correspond à $M2$ puisque le client envoie au marchand le paiement que ce dernier redirigera vers la banque. La propriété $R6$ correspond directement au requis $C3$ tandis que $R7$ et $R8$ sont des versions plus faibles de $B1$ et $B2$.

Le requis $C1$ peut être vérifié indirectement par une combinaison de $R3$ et $R7$ puisque si la banque peut authentifier le client et que ce dernier ne peut répudier le message qu'il a envoyé, alors le paiement est autorisé par le client. Si $R3$ n'est pas satisfait,

mais que la banque authentifie quand même le client, on a une satisfaction faible (non irréfutable) de $C1$. Si $R7$ n'est pas vérifié, alors $C1$ n'est pas satisfait.

Seul $C4$ ne peut être vérifié par *ASPiC*. Il faudrait vérifier l'équité pour le client : s'il envoie un paiement, alors éventuellement, il recevra une confirmation du marchand. Toutefois, on voit que si une panne réseau empêche le lien entre le marchand et le client après que celui-ci ait envoyé son paiement, alors le client ne recevra jamais son reçu. On suppose donc un mécanisme de recouvrement du protocole qui permettrait au client de vérifier directement avec la banque si la transaction a bien eu lieu et, au besoin, de contacter le marchand pour finaliser.

7.2.1.4 Résultats

L'application de l'outil sur le protocole 3kp donne un système de transitions de 13329 états en 21 secondes.

Les résultats obtenus sont que toutes les propriétés sont satisfaites sauf $R3$, la non-répudiation des variables *prix* et *can* par le client. La raison en est que, bien que le client signe le message encrypté pour la banque contenant ces valeurs, rien ne prouve que c'est bien lui qui a encrypté ces valeurs. Un intrus aurait pu, soit dans le cadre de ce protocole ou par un autre protocole complètement différent, forcer le client à signer un message et le rejouer dans une session de 3KP, soit pour pouvoir accuser un client d'avoir effectué une transaction sur le compte de l'intrus ou pour faire faire au client une transaction sur son compte qu'il n'aurait pas voulu faire.

Toutefois cette attaque est peu plausible si on suppose que la banque a un moyen d'associer le certificat de clé publique d'un client, son identité et son numéro de compte. La transaction serait donc refusée si le numéro de compte ne correspondait pas au certificat du client. Et, vue la confidentialité du numéro de compte client, un

TABLEAU 7.3 Résultats obtenus en appliquant *ASPiC* sur des protocoles de commerce électronique

Propriété	Résultat	nb états à vérifier
IBS [6]		
Non-répudiation(prix :$M \rightarrow C$)	oui	37
Non-répudiation(service :$M \rightarrow C$)	oui	34
Équité(service :M)	non	26
Non-répudiation(service :$C \rightarrow M$)	oui	26
SET (Lu et Smolka) [7]		
Confidentialité(f,CAN :C, B)	oui	8131
Confidentialité(f,MAN :M, B)	oui	8131
SET (Lu et Smolka) [8]		
Authentification($B \rightarrow C$)	oui	1336
Authentification($B \rightarrow M$)	oui	809
Authentification($M \rightarrow C$)	non	92
Non-repudiation($tid, (0\|1) : B \rightarrow M$)	oui	763
Non-repudiation($tid, (0\|1) : B \rightarrow C$)	oui	12267
Netbill		
Authentification($B \rightarrow C$)	oui	1365

intrus n'aurait pas pu créer un message contenant ce numéro.

7.2.2 Résultats des protocoles testés

Nous avons aussi testé d'autres protocoles de commerce électronique avec *ASPiC*. Ces protocoles sont décrits dans l'annexe I. Le tableau 7.3 présente les résultats.

Pour le protocole IBS, on voit que chaque étape ne peut être répudiée, tel que prescrit dans les spécifications du protocole. Toutefois, l'équité n'est jamais satisfaite puisqu'à tout moment, un participant peut se retirer et dire ne jamais avoir reçu un message, l'autre participant n'a aucun moyen de prouver le contraire. Il faudrait donc prévoir

[6] Seulement les phases d'entente sur le prix et d'obtention du service
[7] avec l'option "talk to honest"
[8] en forçant la communication du client avec l'intrus

un tiers-parti pour régler ces conflits et ainsi préserver l'équité, surtout lors de la phase d'obtention du service.

Dans le protocole abstrait de SET par Lu et Smolka, nous retrouvons l'attaque "man-in-the-middle" documentée dans (Panti et al., 2002). Un intrus peut conclure une transaction avec un marchand en faisant débiter le compte d'un autre client. Cet autre client voulait originalement conclure une transaction avec l'intrus comme marchand. L'ajout du nom du marchand dans le message encrypté pour la banque par le client règle ce problème.

7.3 Limites et comparaison

Le tableau 7.4 présente les résultats de notre approche, obtenus à l'aide d'*ASPiC*, comparés à ceux obtenus par le model-checker *daTac* de l'outil *AVISS*(Chevalier and Vigneron, 2002)(section 2.3.2), ainsi que par Lowe avec le model-checker FDR(Lowe et al., 1999)(section 2.2.2). La plupart des protocoles analysés sont décrits dans la librairie de protocoles d'authentification (Clark and Jacob, 1997).

Plusieurs attaques ne sont pas détectées par notre outil, mais elles ne sont pas toutes dues à une faiblesse des propriétés de flux d'information utilisées.

Afin d'obtenir des temps de vérification et des tailles de modèles viables, nous avons volontairement limité les capacités de l'outil.

Par défaut, on considère une seule session par acteur qui ne peut jouer qu'un rôle. Dans ces conditions, les attaques de répétition et de fraîcheur ne sont pas détectées puisqu'il n'y a pas d'autre session possible que celle en cours et les valeurs sont ainsi toujours fraîches.

Toutefois, au moment de la modélisation, on peut explicitier un plus grand nombre de

TABLEAU 7.4 Comparaison avec d'autres approches (A : Attaque détectée, OK : protocole correct , - : non analysé)

Protocole	Notre résultat	AVISS	Lowe	Clarke& Jacob
prot d'auth unilatéral ISO 1 passe clé symétrique	OK	A	A	OK
prot d'auth mutuelle ISO 2 passe clé symétrique	OK	A	A	OK
Carlsen's Secret Key Initiator	OK	-	OK	OK
Encrypted Key Exchange	A	A	A	A
Grenouille à Grande Bouche	A	-	A	A
Needham-Schroeder avec clé conventionnelle	OK	A	OK	A
Needham-Schroeder avec clé publique (NSPK)	A	A	A	A
Needham-Schroeder avec signature	OK	-	A	OK
Otway-Rees	OK	A	A	A
TMN	A	A	A	-
Woo and Lam Π_f	OK	-	OK	OK
Woo and Lam Π_1	OK	A	A	A
Woo and Lam Π_2	A	A	A	A
Woo and Lam Π_3	A	A	A	A
Woo and Lam Π	A	A	A	A

sessions ou de rôles par acteur, comme décrit au chapitre précédent. Ces manoeuvres augmentent de beaucoup la taille du système modélisé et nécessitent une intuition de la part du concepteur quant à la configuration qui pourrait occasionner une attaque (par exemple, "2 acteurs *Alice* et 1 *Bob*" ou "un acteur jouant *Alice* et *Bob* et un autre jouant *Bob*", etc.)

Les messages transmis sur le réseau sont typés de sorte qu'un acteur ne peut envoyer un message de type différent de celui attendu. Quelques failles répertoriées impliquent par exemple qu'une concaténation de messages de longueur $n/3$ remplace un message de longueur n et les étapes ultérieures avec ce message-triplet causent une attaque. Cette attaque ne sera pas détectée par *ASPiC*.

La plupart des attaques connues et non détectées sont des attaques de types. Notre outil n'a pas encore la capacité de détecter ces attaques. Il est fort probable que l'affaiblissement du typage lors de la modélisation permettrait de retrouver la plupart de ces attaques.

Par contre, plusieurs attaques, entre autres sur les protocoles ISO, sont des attaques de reprise ou de multiplicité. Dans ces attaques, un acteur se fait authentifier plus de fois qu'il ne le désire. La non-détection de ces attaques est due à une faiblesse dans la spécification des propriétés qui ne tiennent pas compte des sessions multiples. Il faudrait donc reviser l'expression des propriétés pour corriger cette faille.

7.4 Conclusion

L'outil *ASPiC*a été exécuté sur plusieurs exemples de crypto-protocoles et a trouvé avec succès plusieurs failles. Toutefois, certaines failles de types ou de multiplicité n'ont pu être détectées dû à certaines faiblesses du modèle SPASM. Mais somme toute, l'outil développé dans le cadre du présent projet ne sert que de base et d'étude de faisabilité pour le développement d'un outil plus complet de vérification de crypto-protocoles implantant la méthode de flux d'information développée au CRAC. Nous avons tout de même montré la souplesse de la méthode qui permet de valider plusieurs propriétés différentes et ce, avec une unique spécification du protocole par l'usager.

CHAPITRE 8

CONCLUSION

La vérification de protocoles de sécurité est un problème pour les concepteurs de protocoles. Les méthodes formelles apportent une solution à ce problème, mais elles comportent aussi leur lot de difficultés : les méthodes de "theorem-proving" demandent une expertise de la part de l'utilisateur, et les preuves sont longues et ardues mais permettent de trouver les failles ou d'en prouver l'absence ; tandis que les méthodes de "model-checking", quoique plus simples à utiliser, se limitent à l'analyse de systèmes finis afin d'empêcher l'explosion de l'espace-état au moment de la modélisation. Des méthodes de "model-checking" symboliques permettent la vérification de systèmes infinis en manipulant des contraintes faisant en sorte que chaque état du modèle représente un ensemble infini d'états non-symboliques. Nous avons fait un état de l'art des différentes méthodes de vérification formelles au chapitre 2.

Le "model-checking" avec propriétés spécifiées par flux d'information appliqué aux crypto-protocoles a d'abord été étudié par Focardi et Gorrieri (Focardi and Gorrieri, 2001; Focardi, 2001). Ils expriment les propriétés en termes de non-interférence, c'est-à-dire qu'ils vérifient qu'aucun intrus ne peut interférer avec la bonne exécution du protocole. Mullins et Lafrance (Lafrance and Mullins, 2002) ont raffiné cette théorie par la notion d'interférence admissible, qui admet que l'intrus puisse interagir avec le protocole, tant que ça ne change pas son aboutissement. La vérification de l'interférence admissible revient à décider de l'équivalence (bisimulation) de deux sous-ensembles d'un modèle, obtenu à l'aide d'une algèbre de processus, le SPPA symbolique correspondant au SPPA(Lafrance and Mullins, 2003a).

L'interférence admissible donne un gabarit pour la spécification de la plupart des

propriétés de sécurité. Les différences sont minimes entre les propriétés. On distingue 3 niveaux d'actions : haut niveau, bas niveau et déclassifiantes. Seul le partitionnement en niveaux d'action varie d'une propriété à l'autre. Il se fait d'ailleurs directement à partir de la description de chacun. La méthode par interférence admissible souffrait toutefois de l'absence d'une implémentation pour la mettre à l'épreuve. C'est là que se situe notre contribution.

Nous avons d'abord modifié l'algèbre de processus SPPA symbolique puisque la complexité de cette dernière et surtout de la relation d'équivalence associée la rendait difficilement candidate à une implémentation. Nous avons donc développé *SPASM (Security Protocol Algebra for Symbolic Manipulations)*, une algèbre qui permet la synchronisation d'agents honnêtes avec un intrus déductif paresseux, correspondant au modèle de Dolev-Yao (chapitre 3). Elle réduit la taille d'un système de transitions régulier SPPA en regroupant en une seule transition toutes celles ne différant que par les valeurs de leurs variables. Elle utilise de plus des valeurs symboliques plutôt que des valeurs exactes pour représenter les données. Nous avons aussi développé un algorithme de bisimulation symbolique pour vérifier l'équivalence entre deux systèmes de transitions *SPASM*, que nous avons prouvé équivalente à une bisimulation entre deux systèmes SPPA (chapitre 5).

Au chapitre 4, nous avons détaillé l'expression des propriétés de sécurité en termes d'interférence admissible et le partitionnement des actions du modèle en différents niveaux de sécurité selon leur nature et leurs paramètres. Les propriétés d'authentification, confidentialité et anonymat avait déjà été abordées précédemment (Brlek et al., 2003; Lafrance and Mullins, 2002). Nous avons ajouté la non-répudiation (d'origine et de réception) ainsi que l'équité à la liste des propriétés expressibles.

Pour mettre cette approche à l'épreuve, l'outil *ASPiC (Analyse Symbolique de Protocoles Cryptographiques)* a été implanté (chapitre 6). *ASPiC*a été appliqué à plusieurs

crypto-protocoles (chapitre 7) et a trouvé avec succès plusieurs failles documentées dans la littérature, entres autres sur les protocoles de Needham-Schroeder avec clé publique, TMN et une version abstraite du protocole SET. Toutefois, les abstractions et les limitations que nous avons dû faire pour rendre l'outil viable limitent de beaucoup son potentiel de détection d'erreurs.

8.1 Améliorations d'*ASPiC*

Plusieurs améliorations restent donc à être apportées à l'outil, que ce soit au niveau algorithmique ou fonctionnel. Le développement d'*ASPiC* n'en est qu'au stade embryonnaire et plusieurs améliorations permettraient de repousser les limites de l'outil.

L'amélioration des algorithmes, surtout celui de la modélisation qui est très naïf, devrait être une priorité. Ce sont surtout les opérations sur les valuations φ qui seraient à améliorer, car elles peuvent devenir dans un pire cas exponentielles. Certains φ ne diffèrent que par les valeurs de quelques variables alors que les autres sont semblables. Pourtant, les calculs sont toujours exécutés sur toutes les variables. Il y a beaucoup de redondance à ce niveau. Une solution pourrait être de partitionner les φ pour chaque agent du système au lieu d'avoir un φ global pour chaque état ; on diminuerait ainsi le nombre de valuations à conserver. Par exemple, 3 agents ayant chacun 3 configurations possibles de variables sous-tendent 27 valuations globales différentes, alors que partitionnées, ces valuations seraient au nombre de 9. Nous pourrions sauver beaucoup au niveau de la complexité en développant de meilleurs algorithmes.

Du côté fonctionnel, il faudrait éliminer le typage fort actuel. Ceci permettrait de détecter les attaques de type. De même, il faudrait reviser les propriétés afin qu'elles tiennent compte des valeurs fraîches et ainsi détecter des attaques de répétition et de

fraîcheur.

Mais les possibilités d'améliorations précédentes, bien que souhaitables, ne sont pas cruciales à la convivialité de l'outil. Par contre, la difficulté d'analyser les erreurs retournées est un sérieux impair à son application. Une fois une faille détectée, un utilisateur ne dispose que d'un graphe représentant le système attaqué et les valeurs pour lesquelles aucune bisimulation n'a été trouvée. C'est nettement insuffisant puisque la visualisation n'est efficace que pour un nombre restreint d'états. À partir de quelques centaines d'états, les graphes commencent à être très difficiles à interpréter. C'est pourquoi l'une des prochaines étapes serait de développer une procédure de diagnostic d'erreurs, pour rendre explicites, sans avoir à manipuler les résultats, les attaques trouvées.

De plus, il serait souhaitable de permettre à l'utilisateur de spécifier ses propres propriétés, ou les niveaux d'actions désirés à l'aide d'une interface graphique conviviale. Un module supplémentaire pourrait aussi permettre à l'usager de spécifier un protocole sans connaître la syntaxe (loin d'être intuitive) de SPASM, en entrant une spécification *Alice & Bob* par exemple.

8.2 Perspectives

Mais somme toute, *ASPiC* dans son état actuel, bien qu'optimal ni du point de vue des performances ni du point de vue de la détection d'erreurs, donne espoir pour des développements futurs de la vérification des propriétés exprimées par interférence admissible. En effet, la raison pour laquelle plusieurs erreurs n'ont pas été détectées est plutôt due aux limites de la modélisation du protocole et non à celles de la spécification des propriétés. Les modèles sur lesquels les attaques étaient détectables ont tous retournés une erreur.

Des travaux futurs sont prévus sur *ASPiC*. Le développement d'un nouveau module permettra de générer la syntaxe d'entrée en SPASM directement à partir d'une spécification de haut niveau *Alice&Bob*. On vise ainsi à intégrer *ASPiC* dans le projet *AVISPA* (voir section 2.3.2). Cette intégration donnera à l'outil plus de visibilité, en plus de permettre l'accès à une librairie de protocoles exprimés dans un langage de haut niveau.

Aussi, les performances d'*ASPiC* suggèrent qu'une algèbre de processus alliée d'une bisimulation, même symboliques, n'est peut-être pas la meilleure stratégie à jumeler avec l'interférence admissible : la modélisation est ce qui coûte actuellement le plus cher à l'outil, malgré les limites imposées aux comportements du protocole. Une approche de modélisation basée sur les règles de réécriture, par exemple, pourrait être envisagée. Les algèbres de processus partent d'un terme initial et construisent un système de transitions complet à partir d'une sémantique opérationnelle ; on crée de ce fait plusieurs branches inutiles du système. Une méthode à la volée pourrait être envisageable, mais ne ferait qu'accélérer le temps de réponse advenant une erreur ; dans le cas contraire, il faudrait quand même se rendre jusqu'au bout de la modélisation pour voir si oui ou non une branche mène à une action observable (de bas niveau). Par contre, avec un système de réécriture, on pourrait partir d'une action observable acceptée par le système, et, par recherche arrière, on examinerait le flux d'information à la recherche d'une attaque. La recherche serait ainsi ciblée pour la vérification d'une propriété et éviterait beaucoup d'opérations inutiles. Il conviendrait d'étudier plus à fond cette piste pour déterminer quels seraient les impacts sur les problèmes soulevés.

RÉFÉRENCES

Abadi, M. and Gordon, A. (1997). A calculus for cryptographic protocols : The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press.

Abadi, M. and Tuttle, M. (1991). A semantics for a logic of authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM Press.

Alur, R. et al. (1998a). MOCHA : Modularity in model checking. In *Proceedings of the Tenth International Conference on Computer Aided Verification*, volume 1427 of *LNCS*, pages 521–525. Springer-Verlag.

Alur, R., Henzinger, T., and Kupferman, O. (1998b). Alternating-time temporal logic. In *Proceedings of the International Symposium on Compositionality : The Significant Difference (COMPOS'97)*, volume 1536 of *LNCS*, pages 23–60. Springer-Verlag.

Anderson, R. (1999). The formal verification of a payment system. In *Industrial-Strength Formal Methods in Practice*, Formal Approaches to Computin and Information Technology, pages 43–52. Springer-Verlag.

Armando, A., Basin, D., Bouallagui, M., Chevalier, Y., Compagna, L., Mödersheim, S., Rusinowitch, M., Turuani, M., Vigano, L., and Vigneron, L. (2002). The AVISS Security Protocol Analysis Tool. In Brinksma, E. and Guldstrand Larsen, K., editors, *14th International Conference on Computer Aided Verification, CAV'2002*, volume 2404 of *LNCS*, Copenhagen (Denmark), pages 349–353. Springer. Tool presentation.

Asokan, N. et al. (1997). The state of the art in electronic payment systems. *Computer*, **30**(9), 28–35.

Bella, G., Massacci, F., and Paulson, L. (2002a). The verification of an industrial payment protocol : The set purchase phase. In Atluri, V., editor, *9th ACM Conference on Computer and Communications Security*, pages 12–20. ACM Press.

Bella, G., Massacci, F., and Paulson, L. (2002b). Verifying the SET registration protocols. Technical Report UCAM-CL-TR-531, Computer Laboratory, University of Cambridge.

Benerecetti, M. et al. (2002). Verification of payment protocols via multi-agent model checking. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, volume 2348 of *LNCS*, Toronto, Ontario, pages 311–327. Springer-Verlag.

Benerecetti, M. and Giunchiglia, F. (2000). Model checking security protocols using a logic of belief. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 519–534.

Bolignano, D. (1996). Formal verification of cryptographic protocols. In *Proceedings of the Third ACM Conference on Computer and Communication Security*, pages 106–118. ACM Press.

Bolignano, D. (1997). Towards the formal verification of electronic commerce protocols. In *PCSFW : Proceedings of The 10th Computer Security Foundations Workshop*, Rockport, Mass, pages 133–147. IEEE Computer Society Press.

Bouallagui, M., Chevalier, Y., Rusinowitch, M., Turuani, M., and Vigneron, L. (2002). Analyse Automatique de Protocoles de Sécurité avec CASRUL. In *Actes de SAR'2002, Sécurité et Architecture Réseaux.*

Bozga, L., Lakhnech, Y., and Perin, M. (2003). Pattern-based abstraction for verifying secrecy in protocols. In *Proceedings of TACAS'03*, volume 2619 of *LNCS*, pages 299–314. Springer-Verlag.

Brlek, S., Hamadou, S., and Mullins, J. (2003). Aset : Un protocole anonyme et sécuritaire pour les transactions électroniques. Submitted for publication.

Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems*, **8**(1), 18–36.

Chevalier, Y. and Vigneron, L. (2002). Automated Unbounded Verification of Security Protocols. In Brinksma, E. and Guldstrand Larsen, K., editors, *14th International Conference on Computer Aided Verification, CAV'2002*, volume 2404 of *LNCS*, Copenhagen (Denmark), pages 324–337. Springer.

Clark, J. and Jacob, J. (1997). A survey of authentication protocol literature. Technical Report 1.0.

Cleaveland, R. and Smolka., S. (1999). Process algebra. In Webster, J., editor, *Encyclopedia of Electrical Engineering*. John Wiley & Sons.

de Alfaro, L., Henzinger, T., and Majumdar, R. (2001). Symbolic algorithms for infinite-state games. In *Proceedings of the 12th International Conference on Concurrency Theory (CONCUR)*, volume 2154 of *LNCS*, pages 536–550. Springer-Vale.

Delzanno, G. and Ganty, P. (2003). Symbolic methods for automatically proving secrecy and authentication in inifinite-state models of cryptographic protocols. In *Proceedings of WISP, Workshop on Issues in Security and Petri Nets*.

Dolev, D. and Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, **29**(2), 198–208.

FDR (2000). *Failure Divergence Refinement - FDR2 User Manual*. Formal Systems Europe Ltd, version 2.67 edition.

Focardi, R. (2001). Classification of security properties (part II : Network security). slides. Second Internation School on Foundations Of Security Analysis and Design (FOSAD'01).

Focardi, R. and Gorrieri, R. (2001). Classification of security properties (part i : Information flow). In *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*, pages 331–396. Springer-Vale.

Genet, T. and Klay, F. (2000). Rewriting for cryptographic protocol verification. In *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Genet, T. and Tong, V. V. T. (2001). Reachability analysis of term rewriting systems with timbuk. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Glossary. Netcraft glossary. URL = http ://www.netcraft.com/ssl/glossary.html.

Gong, L., Needham, R., and Yahalom, R. (1990). Reasoning About Belief in Cryptographic Protocols. In Cooper, D. and Lunt, T., editors, *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society.

Guillem-Lessard, S. (2002). Cryptographie : tutoriel. URL = http ://www.uqtr.ca/ delisle/Crypto/. Projet de fin d'études, UQTR.

Guttman, J. (2001). Security goals : Packet trajectories and strand spaces. In Focardi, R. and Gorrieri, R., editors, *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*, pages 187–261. Springer-Verlag.

Hains, G. and Bergeron, M. (2002). The BNAI analyzer : A tool for verifying admissible information flow in protocols. In *International Conference on Parallel and Distributed Computer Systems*, Cambridge MA, USA. IASTED.

Heintze, N. et al. (1996). Model checking electronic commerce protocols. In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, Oakland, Ca.

Hennessy, M. and Lin, H. (1995). Symbolic bisimulation. In *Theoretical Computer Science*, volume 138, pages 353–389.

Hoare, C. (1985). *Communicating Sequential Processes*. Prentice-Hall.

Kailar, R. (1996). Accountability in electronic commerce protocols. *IEEE Transactions on Software Engineering*, **22**(5).

Kessler, V. and Neumann, H. (1998). A sound logic for analysing electronic commerce protocols. In Quisquater, J.-J., Deswarte, Meadows, C., and Gollmann, D., editors, *Computer Security ESORICS 98*, volume 1485 of *LNCS*, pages 345–360. Springer-Vale.

Kremer, S. and Raskin, J.-F. (2001). A game-based verification of non-repudiation and fair exchange protocols. In *Proceedings of the International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 551–566. Springer-Vale.

Lafrance, S. and Mullins, J. (2002). Bisimulation-based non-deterministic admissible interference with applications to the analysis of cryptographic protocols. In Harland, J., editor, *Computing : The Australasian Theory Symposium*, volume 61 of *Electronic Notes in Theoretical Computer Science*, pages 1–24. Elsevier.

Lafrance, S. and Mullins, J. (2003a). Symbolic approach to the analysis of security protocols. In Cervesato, I., editor, *Proceedings of Fondations of Computer Security*.

Lafrance, S. and Mullins, J. (2003b). Using admissible interference to detect denial of service vulnerabilities. In Morris, J. M., Aziz, B., and Oehl, F., editors, *Sixth International Workshop in Formal Methods*. Electronic Workshops in Computing (eWiC) by British Computer Society (BCS).

Lin, H. (1998). On-the-fly instantiation of value-passing processes. In *FORTE/PSTV'98*, Paris, pages 215–230. Kluwer Academic Publishers.

Lowe, G. (1996). Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag.

Lowe, G., Donovan, B., and Norris, P. (1999). Analyzing a Library of Security Protocols Using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*.

Lowe, G. and Roscoe, B. (1997). Using csp to detect errors in the tmn protocol. *IEEE Transactions on Software Engineering*, **23**(10), 659–669.

Lu, S. and Smolka, S. (1999). Model checking the secure electronic transaction (SET) protocol. In *Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 358–365. IEEE Computer Society Press.

Meadows, C. (1996a). Analyzing the needham-schroeder public key protocol : A comparison of two approaches. In *Proceedings of ESORICS*, volume 1146 of *LNCS*. Springer Verlag.

Meadows, C. (1996b). The NRL protocol analyzer : An overview. *Journal of Logic Programming*, **26**(2), 113–131.

Meadows, C. (1999). Analysis of the internet key exchange protocol using the nrl protocol analyzer. In *Proceedings of SSP-99*, pages 216–231. IEEE Comp. Society Press.

Meadows, C. and Syverson, P. (1998). A formal specification of requirements for payment transactions in the set protocol. In *DRAFT for Preproceedings of Financial Cryptography 98*, Anguilla, BWI.

Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall.

Mullins, J. (2000). Non-deterministic admissible interference. *Journal of Universal Computer Science*.

Needham, R. and Schroeder, M. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, **21**(12).

Panti, M., Spalazzi, L., and Tacconi, S. (2002). Verification of security properties in electronic payment protocols. In *ACM SIGPLAN and IFIP WG 1.7 Workshop on Issues in the Theory of Security*, Portland, Oregon. ACM Press.

Paroux, G. (2003). Conception et implantation d'une nouvelle version de l'analyseur BNAI. stage de maîtrise réalisé au LIFO (Université d'Orléans) sous la direction de G. Hains.

Parrow, J. (2001). An introduction to the pi-calculus. In Bergstra, Ponse, S., editor, *Handbook of Process Algebra*, pages 479–543. Elsevier.

Paulson, L. (1998). The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, **6**, 85–128.

Schneider, S. (1998). Formal analysis of a non-repudiation protocol. In *PCSFW : Proceedings of The 11th Computer Security Foundations Workshop*, pages 54–65. IEEE Computer Society Press.

Schneider, S. and Sidiropoulos, A. (1996). Csp and anonymity. In *Proceedings : Computer Security - ESORICS 96*, volume 1146 of *LNCS*, pages 198–218. Springer-Vale.

Schnoebelen, P. et al. (1999). *Vérification de logiciels : techniques et outils du model-checking*. Vuibert, Paris, J.-P. Moreux edition.

Song, D. X., Berezin, S., and Perrig, A. (2001). Athena : A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, **9**(1/2), 47–74.

Syverson, P. and van Oorschot, P. (1994). On unifying some cryptographic protocol logics. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 14–28.

van Oorschot, P. (1993). Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 233–243. ACM Press.

Vienneau, R. (1996). *Software Engineering*, chapter A Review of Formal Methods. IEEE Computer Society Press.

Zhou, J. and Gollmann, D. (1997). Evidence and non-repudiation. *Journal of Network and Computer Application*, **20**(3), 267–281.

Zhou, J. and Gollmann, D. (1998). Towards verification of non-repudiation protocols. In *Proceedings of 1998 International Refinement Workshop and Formal Methods, Pacific*, Canberra, Autralia, pages 370–380. Springer.

Index

équité, 4, 78
équivalence, 27
    observationnelle, 67

algèbres de processus, 25
    CCS, 25
    CSP, 27
    SPASM, 47
    SPPA, 45
anonymat, 3, 74
atomicité, 4
attaques, 5
authentification, 3, 69

bisimulation, 82
    standard, 84
    symbolique, 87
BNAI, 68

canaux
    externes, 50
    publics, 50
chiffrement, 2
confidentialité, 3, 71

disponibilité, 4

fiabilité, 4
fonctions
    génératrices, 49
    privées, 49

hachage, 2

intégrité, 4
interférence
    admissible, 65
    non-interférence, 60

logiques
    alternantes, 37
    temporelles, 33

marqueurs, 54

noeud
    d'un système de transitions, 84
non-répudiation, 3, 76
non-traçabilité, 3

observation, 66

processus, 50
    intrus, 58
protocole, 2

termes
    clos, 47
    de l'algèbre, 51
    paramètres, 47

valuation
    non-symbolique, 83
    symbolique, 52
        opérations sur, 53

ANNEXE I

CRYPTO-PROTOCOLES

Les sections qui suivent décrivent les protocoles référés dans le texte. Les premiers sont des protocoles de sécurité qui ont servi pour étudier la vérification de propriétés particulières, par exemple, des protocoles d'authentification ou de non-répudiation. Suivront ensuite les protocoles de commerce électronique qui ont fait l'object d'études de cas, que ce soit dans le cadre du présent mémoire ou d'autres travaux de recherche.

I.1 Protocole de Needham-Schoeder avec clé publique(NSPK)

Ce protocole d'authentification à double sens a été utilisé pendant plusieurs année avant que des failles y soient découvertes par Lowe(Lowe, 1996). Sa simplicité et la nature de ses failles font de lui une référence pour la vérification de protocoles.

1. Alice $\rightarrow$ Bob : $\{N_a, Alice\}_{PK_{Bob}}$
2. Bob $\rightarrow$ Alice : $\{N_a, N_b\}_{PK_{Alice}}$
3. Alice $\rightarrow$ Bob : $\{N_b\}_{PK_{Bob}}$

où N_a et N_b sont des nonces de sessions choisis respectivement par *Alice* et *Bob*. La réception du deuxième message par *Alice* et la présence du nonce N_a lui permettent d'authentifier *Bob*. Quant à *Bob*, il authentifie *Alice* lors de la réception de N_b dans le troisième message.

I.2 Protocole de Woo et Lam (version π)

Ce protocole est un protocole d'authentification à sens unique (*Bob* authentifie *Alice*) et utilise un serveur.

1. Alice $\rightarrow$ Bob : $Alice$
2. Bob $\rightarrow$ Alice : N_b
3. Alice $\rightarrow$ Bob : $\{N_b\}_{ShK(Alice,S)}$
4. Bob $\rightarrow$ S : $\{Alice, \{N_b\}_{ShK(Alice,S)}\}_{ShK(Bob,S)}$
5. S $\rightarrow$ Bob : $\{N_b\}_{ShK(Bob,S)}$

où N_b est un nonce généré par *Bob* . Lorsque *Bob* reçoit le cinquième message du serveur et recouvre N_b, il est assuré de parler avec *Alice*.

I.3 Protocole de la grenouille à grande bouche (GGB)

Ce protocole a pour but l'échange d'une clé de session entre deux acteurs *Alice* et *Bob* par l'intermédiaire d'un serveur. Le protocole doit aussi permettre à *Bob* d'authentifier *Alice*.

1. Alice $\rightarrow$ S : $Alice, Bob, \{kab\}_{K_{AliceS}}$
2. S $\rightarrow$ Bob : $\{Alice, kab\}_{K_{BobS}}$
3. Alice $\rightarrow$ Bob : $\{m\}_{kab}$

où kab est une clé de session et m un message qu'*Alice* veut envoyer à *Bob*. *Alice* crée une clé de session qu'elle envoie à *Bob* par l'intermédiaire du serveur S. La communication entre *Alice* et *Bob* peut continuer avec la nouvelle clé créée. La réception du deuxième message par *Bob* lui permet d'authentifier sa correspondante.

I.4 Protocole TMN

Ce protocole d'échange de clé permet d'établir une clé de session entre A et B par l'intermédiaire d'un serveur. L'opération $\otimes$ représente le OU-exclusif.

1. A $\rightarrow$ S : $A, S, B, \{k_a\}_{PK_S}$
2. S $\rightarrow$ B : S, B, A
3. B $\rightarrow$ S : $B, S, A, \{k_b\}_{PK_S}$
4. S $\rightarrow$ A : $S, A, B, k_a \otimes k_b$

où k_a et k_b sont des clés choisies respectivement par A et B pour la session courante. Chacun des deux participants choisit une clé. Suite à la réception du dernier message, A peut obtenir la clé k_b par un OU-exclusif puisqu'elle connaît sa propre clé. k_b sert ensuite de clé de session.

I.5 Protocole de Otway-Rees

Ce protocole est un protocole d'échange de clé entre deux participants A et B et utilise un serveur qui génère une nouvelle clé.

1. A $\rightarrow$ B : $N_a, A, B, \{N_a, A, B\}_{ShK(A,S)}$
2. B $\rightarrow$ S : $N_a, A, B, \{N_a, A, B\}_{ShK(A,S)}, N_b, \{N_a, A, B\}_{ShK(B,S)}$
3. S $\rightarrow$ B : $N_a, \{N_a, Kab\}_{ShK(A,S)}, \{N_b, Kab\}_{ShK(B,S)}$
4. B $\rightarrow$ A : $N_a, \{N_a, Kab\}_{ShK(A,S)}$

où N_a et N'_b sont des nonces générés respectivement par A et B et Kab est la clé générée par le serveur qui sera utilisée pour communiquer entre A et B. A amorce la session auprès de B et attend que celui-ci communique avec le serveur pour obtenir la clé de session Kab qui servira pour des communications futures.

I.6 Protocoles de Zhou et Gollmann

Ces deux protocoles sont des protocoles de non-répudiation équitables, le premier utilise à tout coup un tiers-parti pour transférer le message final tandis que le second, la version optimiste du premier, n'utilise le tiers-parti qu'en cas de recouvrement, si le protocole échoue. La non-répudiation est prévue par la signature des messages correspondant.

I.6.1 Protocole de non-répudiation avec tiers-parti

1. Alice $\rightarrow$ Bob : $f_{EOO}, Bob, L, \{M\}_K, EOO$
2. Bob $\rightarrow$ Alice : $f_{EOR}, Alice, L, EOR$
3. Alice $\rightarrow$ TTP : $f_{SUB}, Bob, L, K, sub_K$
4. Bob $\leftrightarrow$ TTP : $f_{CON}, Alice, Bob, L, K, con_K$
5. Alice $\leftrightarrow$ TTP : $f_{CON}, Alice, Bob, L, K, con_K$

où $\leftrightarrow$ représente un *ftp-get* de l'agent au lieu d'un envoi de message. On empêche ainsi que le message se perde indéfiniment sur le réseau. Les acteurs peuvent aller le chercher quand ils peuvent. Les messages 4 et 5 sont ceux par lesquels chacun des participants reçoit la preuve d'origine et de réception respectivement. Les données échangées sont

- L est une étiquette représentant la session du protocole
- M est le message à échanger entre *Alice* et *Bob*
- K est la clé choisie par *Alice* pour encrypter le message, elle devra être transmise à *Bob* pour qu'il puisse le lire.
- f_xxx sont des flags identifiant l'intention du message
- $EOO = [f_{EOO}, Bob, L, \{M\}_K]_{SK_{Alice}}$ est l'évidence de l'origine de $\{M\}_K$.
- $EOR = [f_{EOR}, Alice, L, \{M\}_K]_{SK_{Bob}}$ est l'évidence de la réception de $\{M\}_K$.

- $sub_K = [f_{SUB}, Bob, L, K]_{SK_{Alice}}$ est l'évidence de la submission de K par *Alice* .
- $con_K = [f_{CON}, Alice, Bob, L, K]_{SK_{TTP}}$ est l'évidence de la confirmation de la réception de K par le tiers-parti.

I.6.2 Protocole de non-répudiation optimiste

1. Alice $\rightarrow$ Bob : $f_{EOO}, Bob, L, \{M\}_K, EOO$
2. Bob $\rightarrow$ Alice : $f_{EOR}, Alice, L, EOR$
3. Alice $\rightarrow$ Bob : $f_{EOO_K}, Bob, L, K, EOO_K$
4. Bob $\rightarrow$ Alice : $f_{EOR_K}, Alice, L, EOR_K$

Si le protocole se termine avant l'envoi du 3e message, il n'y a pas de problème. Par contre, si *Alice* ne peut obtenir le 4e message, soit à cause d'un *Bob* malhonnête ou pour une panne sur le réseau, elle amorce le protocole de recouvrement.

3'. Alice $\rightarrow$ TTP : $f_{SUB}, Bob, L, K, sub_K$
4'. Bob $\leftrightarrow$ TTP : $f_{CON}, Alice, Bob, L, K, con_K$
5'. Alice $\leftrightarrow$ TTP : $f_{CON}, Alice, Bob, L, K, con_K$

où

- $EOO_K = [f_{EOO_K}, Bob, L, K]_{SK_{Alice}}$ est l'évidence de l'origine de K.
- $EOR_K = [f_{EOR_K}, Alice, L, K]_{SK_{Bob}}$ est l'évidence de la réception de K.

Bob reçoit une preuve de l'origine du message lorsqu'il reçoit les message 3 ou 4'. De même, *Alice* a sa preuve de réception par les messages 4 ou 5'.

I.7 SET

SET (Secure Electronic Transactions) est un protocole de commerce électronique développé conjointement par VISA et MasterCard pour sécuriser les transactions par

carte de crédit en ligne. Ce protocole est très compliqué et comprend plusieurs phases, entre autres la phase d'enregistrement, faite une fois par le client et lui permettant par la suite d'utiliser le service, et la phase de paiement. Le protocole complet est décrit en plusieurs centaines de pages. Nous ne le décrirons pas au complet ici.

I.7.1 Phase d'enregistrement de SET

Ce protocole se fait entre un client C et une autorité de certification CA sous la tutelle d'une autorité maître RCA. Elle permet au client C de s'inscrire auprès du service SET, afin de pouvoir par la suite effectuer des transactions en ligne.

1. C $\rightarrow$ CA : $C, N1_C$
2. CA $\rightarrow$ C : $[C, N1_C]_{SSK_{CA}}, Cert_{RCA}(PEK_{CA}), Cert_{RCA}(PSK_{CA})$
3. C $\rightarrow$ CA : $\{C, N2_C, h(CAN)\}_{k1_c}, \{k1_c, CAN, h(C, N2_C\}_{PEK_{CA}}$
4. CA $\rightarrow$ C : $[C, N2_C, N_{CA}]_{SSK_{CA}}, Cert_{RCA}(PEK_{CA}), Cert_{RCA}(PSK_{CA})$
5. C $\rightarrow$ CA : $\{m, [h(m, CAN, CardSecret)]_{SSK_C}\}_{k3_c}$,
   $\{k3_c, CAN, CardSecret\}_{PEK_{CA}}$
6. CA $\rightarrow$ C : $\{[C]_{SSK_{CA}}, N3_C, CA, N_{CA}, Cert_{CA}(PSK_C), Cert_{RCA}(PSK_{CA}\}_{k2C}$

où $m = C, N3_C, k2_C, PSK_C$, Nx sont des nonces, $Cert_x(\_)$ représente un certificat signé par x pour la valeur $\_$, CAN est le numéro de carte de crédit du client (Customer Account Number), $CardSecret$ est un mot de passe pour ce compte, $k1_c, k2_c, k3_c$ sont des clés choisies par le client. PEK et SEK est la paire de clés asymétriques pour l'encryption de données tandis que PSK et SSK est la paire de clés utilisées pour la signature.

I.7.2 Version de SET de Lu et Smolka

Ce protocole est une version simplifiée de la phase de paiement de SET en 6 messages qui comprend l'essentiel de la version complète. C'est plutôt cette version qui est utilisée pour le model-checking.

1. C $\rightarrow$ M : Tid_{req}
2. M $\rightarrow$ C : $[Tid]_{SK_M}$
3. C $\rightarrow$ M : $[Tid]_{SK_C}, \{Tid, \$purch\}_{PK_M}, [\{Tid, \$pay, CAN\}_{PK_B}]_{SK_C}$
4. M $\rightarrow$ B : $[\{Tid, \$pay, CAN\}_{PK_B}]_{SK_C}, [Tid]_{SK_M}, \{Tid, \$auth, MAN\}_{PK_B}$
5. B $\rightarrow$ M : $[Tid, res]_{SK_B}$
6. M $\rightarrow$ C : $[Tid, res]_{SK_B}$

où Tid est un identificateur de la transaction généré par le marchand, Tid_{req} est la requête de l'identificateur par le client, $\$purch$ est le montant de l'achat, $\$pay$ est le montant que le client désire payer, $\$auth$ est le montant demandé par le marchand auprès de la banque, MAN est le numéro de compte du marchand (Merchant Account Number) et res est le résultat de la transaction.

I.7.3 C-SET

C-SET ou Chip-Set est une variante de SET développée par le Groupement des Cartes Bancaires de France (GIE CB) et dont l'architecture est basée sur l'utilisation de cartes à puces. Les spécifications exactes du protocole ne sont pas disponibles.

I.8 Internet Billing Server Protocol

Ce protocole, développé par l'Université de Carnegie Mellon offre des services de facturation pour des fournisseurs de service par Internet. Il permet de s'assurer qu'un

client est chargé uniquement pour un service rendu dont le prix a été entendu auparavant. Ce protocole se fait en 3 phases :

La phase d'entente sur le prix :

1. C $\rightarrow$ M : $[RequetePrix]_{SK_C}$
2. M $\rightarrow$ C : $[Prix]_{SK_M}$

La phase d'obtention du service :

3. C $\rightarrow$ M : $[[Prix]_{SK_M}, Prix]_{SK_C}$
4. M $\rightarrow$ C : $[Service]_{SK_M}$
5. C $\rightarrow$ M : [Confirmation service rendu]$_{SK_C}$

La phase de facturation :

6. C $\rightarrow$ M : [Requete facture]$_{SK_C}$
7. M $\rightarrow$ B : $[\{Facture\}_{PK_B}]_{SK_M}$
8. B $\rightarrow$ M : $[\{Facture\}_{PK_M}]_{SK_B}, [\{Facture\}_{PK_C}]_{SK_B}$
9. M $\rightarrow$ C : $[\{Facture\}_{PK_C}]_{SK_B}$

Ce protocole prévoit la non-répudiation à chacune des étapes par la signature du participant concerné. Le client peut prouver que le marchand s'est entendu sur le prix et vice versa ; le client peut prouver qu'il a reçu tel service du marchand, tout comme le marchand peut prouver qu'il a rendu tel service au client. De même, marchand et client peuvent prouver qu'un montant X a été transféré d'un à l'autre.

I.9 Universal Electronic Payment System (UEPS)

UEPS est un protocole hors-ligne conçu pour être utilisé dans les régions du monde où la fiabilité des moyens de communication laisse à désirer. Il a d'ailleurs été implanté en Afrique du Sud au début des années 1990. Il utilise des cartes qui constituent un

porte-monnaie électronique. L'argent est transféré de la banque à la carte du client par l'intermédiaire d'un guichet fixe. Ensuite, le client produit un chèque électronique qu'il transfère de sa carte vers la carte du marchand. Le marchand encaisse les chèques par l'intermédiaire du même guichet que le client. Le protocole pour l'échange d'un chèque entre le client et le marchand est décrit ici. Les autres échanges se font de façon similaire.

1. C $\rightarrow$ M : $\{C, N_C\}_K$
2. M $\rightarrow$ C : $\{M, N_M\}_{\{C,N_C\}_K}$
3. C $\rightarrow$ M : $\{X\}_{\{M,N_M\}_{\{C,N_C\}_K}}$

où N_C, N_M sont des nonces, X est le chèque électronique et K est une clé dont la nature n'est pas expliquée.

I.10 NetBill

Ce protocole a été développé à l'Université de Carnegie-Mellon pour les transactions de biens électroniques de faible coût. Ce protocole permet l'authentification du client et la non-répudiation de la transaction. Une variante assure aussi l'anonymat du client.

1. C $\rightarrow$ M : *Requete bien*
2. M $\rightarrow$ C : $\{Bien\}_K$
3. C $\rightarrow$ M : $[Ordre\ de\ paiement\ électronique]_{SK_C}$
4. M $\rightarrow$ B : $Endorse([Ordre\ de\ paiement\ électronique]_{SK_C}), [K]_{SK_M}$
5. B $\rightarrow$ M : $[recu, [K]_{SK_M}]_{SK_B}$
6. M $\rightarrow$ C : $[recu, [K]_{SK_M}]_{SK_B}$

Si le client ne reçoit pas le reçu signé, il peut contacter directement la banque

7. C $\rightarrow$ B : *Demande transaction*
8. B $\rightarrow$ C : $[recu, [K]_{SK_M}]_{SK_B}$

La signature de l'ordre de paiement électronique du client permet de l'authentifier auprès de la banque et du marchand, de même, elle garantit la non-répudiation de la transaction par le client.

I.11 DigiCash

Ce protocole de type argent comptant se déroule hors-ligne. Il assure l'anonymat du client auprès du marchand, car jamais son identité n'est requise, et auprès de la banque, puisque la procédure d'aveuglement de la pièce la rend méconnaissable. Le protocole présenté ici est une version simplifiée pour fins de model-cheking.

1. C $\rightarrow$ B : *Requete retrait*
2. B $\rightarrow$ C : *piece*
3. C $\rightarrow$ M : *requete bien, piece aveugle*
4. M $\rightarrow$ C : *defi pour la piece aveugle*
5. C $\rightarrow$ M : *reponse au defi*
6. M $\rightarrow$ C : *bien*
6. M $\rightarrow$ B : *piece aveugle, paire defi/reponse*
6. B $\rightarrow$ M : *certificat de depot*

La partie importante de la dépense de la pièce est le défi et la réponse au défi. Le client doit faire attention de ne pas répondre à deux défis différents pour une même pièce car ça constituerait une fraude. De plus, l'algorithme de défi/réponse est tel que deux paires défi/réponse sont suffisantes pour que la banque retrace le client.

I.12 iKP

Ce protocole permet le paiement en ligne par carte de crédit et peut être étendu pour accepter les chèques électroniques. 3 versions du protocole existent selon le niveau de sécurité désiré, c'est-à-dire le nombre d'acteurs ayant une paire de clés privée/publique d'encryption. Le protocole où seule la banque possède une paire de clés est 1KP. Lorsque le marchand aussi possède des clés, on a affaire à 2KP, et 3KP ajoute le client à la liste d'acteurs avec clés. Ce protocole assure la non-répudiation des messages provenant des acteurs ayant une clé, de même que leur authentification. Nous décrivons ici le protocole 3KP.

1.	C $\rightarrow$ M	:	$Salt_C, CID, Cert_C$	
2.	M $\rightarrow$ C	:	$Clear, [h(common), h(v)]_{SK_M}$	$(Clear, sig_m)$
3.	C $\rightarrow$ M	:	$\{Slip\}_{PK_B}, [\{Slip\}_{PK_B}, h(common)]_{SK_C}$	$(encSlip, sig_c)$
4.	M $\rightarrow$ B	:	$Clear, h(desc, salt_c), encSlip, sig_m, sig_c$	
5.	B $\rightarrow$ M	:	$Y/N, [Y/N, h(common)]_{SK_B}$	$(Y/N, sig_b)$
6.	M $\rightarrow$ C	:	$Y/N, v, sig_b$	

où $Salt_C$ est un nombre aléatoire utilisé pour saler le hash de *desc* (description de la marchandise), $CID = h(r_c, can)$ où r_c un nombre aléatoire généré par le client et *can* son numéro de compte, identifie le client. $Cert_C$ est un certificat permettant d'authentifier le client ; il contient entre autres sa clé publique. TID est un nombre généré par le marchand pour identifier la transaction, $common = (price, ID_M, TID_M, date, n_M, CID, h(desc, Salt_C), h(v))$, $clear = (ID_M, TID_M, date, n_M, h(v), h(common))$, ID_M est l'identificateur du marchand, v est un nombre aléatoire généré par le marchand comme preuve supplémentaire qu'il a accepté la transaction. $Slip = (price, h(common), r_c, can)$ est la preuve de paiement du client.

I.13 Millicent

Ce protocole permet les transactions à faible coût de façon sécuritaire. Des courtiers servent d'intermédiaire entre les clients et les marchands. Le client achète des comptes chez le courtier qui servent de monnaie d'échange pour acheter auprès du courtier des comptes avec les marchands désirés. Lors d'une transaction avec un marchand, la valeur du compte marchand est déduite du montant de la transaction. Le client peut monnayer ses comptes marchands ou courtiers quand il veut. La transaction elle-même ne requiert pas l'intermédiaire du courtier, les informations étant vérifiées localement chez le marchand. Et les coûts d'ouverture de comptes pour le client sont faibles puisque tous les marchands auxquels il a affaire sont centralisés dans son unique compte courtier.

1. C $\rightarrow$ B : *Requete de compte courtier*
2. B $\rightarrow$ C : *Compte courtier*, $secret_{CC}$
3. C $\rightarrow$ B : *Requete d'un compte marchand,compte courtier*, $secret_{CC}$
4. B $\rightarrow$ C : *Compte marchand*, $secret_{CM}$, *change du compte courtier*
5. C $\rightarrow$ M : *Requete service,compte marchand,* $secret_{CM}$
6. M $\rightarrow$ C : *Service,change du compte marchand*

où $secret_{CC}, secret_{CM}$ représente des secrets associés au compte courtier et marchand respectivement. Les étapes 5 et 6 peuvent s'exécuter tant qu'il reste assez d'argent dans le compte marchand.

ANNEXE II

PREUVES DE PROPOSITIONS

II.1 Preuve de la correction de l'algorithme de bisimulation symbolique

Soit $\varphi, \varphi', \varphi_i, ..., \psi, \psi', \psi_j, ...$ des valuations symboliques.

Preuve de la proposition 5.5. Nous connaissons la post-condition à l'exécution de la fonction $close(\langle s_0, \varphi_0\rangle, \langle u_0, \psi_0\rangle)$, soit que la fonction retourne φ_0 et $Sim(s_0, u_0) = \varphi_0$ et $\forall\langle s, \varphi\rangle \in G_{s_0}.Partial(s) = \varphi$. Nous allons déterminer quelles préconditions peuvent mener à une telle post-condition et montrer qu'elles impliquent que $(s_0, u_0) \in \mathcal{B}^{\varphi_0,\psi_0}$.

D'abord, définissons une propriété $\mathcal{P}(s, u, \phi')$ si $close(\langle s, \phi\rangle, \langle u, \psi\rangle) = \phi'$.

En partant de la post-condition, nous avons remonté l'exécution de l'algorithme et obtenu la preuve d'algorithme suivante.

$\{Sim(s_i, u_j) \in dom(Sim) \longrightarrow \mathcal{P}(s_i, u_j, Sim(s_i, u_j))$
ou $\nexists\langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle \longrightarrow \mathcal{P}(s_i, u_j, \varphi_i)$
ou $\exists\{t|\langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle\} \longrightarrow \varphi'' = \bigwedge_t match(t, \langle u_j, \psi_j\rangle)$ et $\mathcal{P}(s_i, u_j, \varphi'')$
et pour chaque $\langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle$ alors
$(\nexists\langle u_j, \psi_j\rangle \stackrel{\alpha}{\Longrightarrow} \langle u'_j, \psi'_j\rangle \longrightarrow \varphi_i \backslash \varphi'_i \supseteq \varphi''$
ou $\{\exists\{\langle u_j, \psi_j\rangle \stackrel{\alpha'}{\Longrightarrow} \langle u_k, \psi_k\rangle\}$ tel que pour chaque k.
$\exists\varphi'_k = close(\langle s'_i, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle, \langle u_k, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle)$

et $(\forall v \in fv(\alpha).[\![v]\!]_{\overline{\varphi'_k}} \subseteq [\![v]\!]_{\overline{\varphi'_i}} \cap [\![v]\!]_{\overline{\psi_k}})$ et $\mathcal{P}(s'_i, u_k, \varphi'_k)$
et $\vee_k \varphi'_k \supseteq \varphi'')\}$

13 if $(s, u) \in dom(Sim)$
$\{\mathcal{P}(s_i, u_j, Sim(s_i, u_j))\}$

14 return $Sim(s_i, u_j)$

15 else if $\{\langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle\} = \emptyset$
$\{\mathcal{P}(s_i, u_j, \varphi_i)\}$

16 $Partial(s_i) := Partial(s_i) \vee \varphi_i$

17 $Sim(s_i, u_j) := \varphi_i$
$\{\mathcal{P}(s_i, u_j, Sim(s_i, u_j))\}$

18 return φ_i

19 else
$\{\exists\{t | \langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle\}$ et $\varphi'' = \bigwedge_t match(t, \langle u_j, \psi_j\rangle$ et $\mathcal{P}(s_i, u_j, \varphi'')$
et pour chaque $\langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle$ alors
$(\nexists \langle u_j, \psi_j\rangle \stackrel{\alpha}{\Longrightarrow} \langle u'_j, \psi'_j\rangle \longrightarrow \varphi_i \backslash \varphi'_i \supseteq \varphi''$
ou $\{\exists\{\langle u_j, \psi_j\rangle \stackrel{\alpha'}{\Longrightarrow} \langle u_k, \psi_k\rangle\}$ tel que pour chaque k.
$\exists \varphi'_k = close(\langle s'_i, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle, \langle u_k, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle)$
et $(\forall v \in fv(\alpha).[\![v]\!]_{\overline{\varphi'_k}} \subseteq [\![v]\!]_{\overline{\varphi'_i}} \cap [\![v]\!]_{\overline{\psi_k}})$ et $\mathcal{P}(s'_i, u_k, \varphi'_k)$
et $\vee_k \varphi'_k \supseteq \varphi'')\}$

20 $Sim(s_i, u_j) := \varphi_i$

21 for each $t \in \{\langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle\}$
{voir la remarque}
$\{\nexists \langle u_j, \psi_j\rangle \stackrel{\alpha}{\Longrightarrow} \langle u'_j, \psi'_j\rangle \longrightarrow \varphi_i \backslash \varphi'_i \supseteq \varphi''$
ou $\{\exists\{\langle u_j, \psi_j\rangle \stackrel{\alpha'}{\Longrightarrow} \langle u_k, \psi_k\rangle\}$ tel que pour chaque k.

$\exists\varphi'_k = close(\langle s'_i, \varphi'^{|\alpha}_i \wedge \psi^{|\alpha'}_k\rangle, \langle u_k, \varphi'^{|\alpha}_i \wedge \psi^{|\alpha'}_k\rangle)$
et $(\forall v \in fv(\alpha).[\![v]\!]_{\overline{\varphi'_k}} \subseteq [\![v]\!]_{\overline{\varphi'_i}} \cap [\![v]\!]_{\overline{\psi_k}})$ et $\mathcal{P}(s'_i, u_k, \varphi'_k)$
et $\vee_k \varphi'_k \supseteq \varphi''\}$

22 $\varphi_t := match(t, \langle u_j, \psi_j\rangle)$

$\{\varphi_t \supseteq \varphi''\}$
$\{\forall t \in \{\langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle\}.\varphi_t \supseteq \varphi''\}$

$\{\mathcal{P}(s_i, u_j, \bigwedge_t \varphi_t\}$
23 $\varphi_s := \bigwedge_t \varphi_t$
$\{\mathcal{P}(s_i, u_j, \varphi_s\}$
24 $Partial(s_i) := Partial(s_i) \vee \varphi_s$
25 $Sim(s_i, u_j) := \varphi_s$
$\{\mathcal{P}(s_i, u_j, Sim(s_i, u_j))\}$

Remarque. Pour obtenir cette pré-condition, il faut étudier ce qui se passe lors de la séquence $match(\langle s_i, \varphi_i\rangle \stackrel{\alpha}{\Longrightarrow} \langle s'_i, \varphi'_i\rangle, \langle u_j, \psi_j\rangle$ dont nous avons la post-condition $\{\varphi_t \supseteq \varphi''\}$.

Voici quelques rappels pour analyser cette séquence : Selon la définition 5.6, $dom(\varphi^{|\alpha}) = fv(\alpha)$. Donc, si
- α est de la forme $c(x)$, alors $dom(\varphi^{|c(x)}) = \{x\}$.
- α est de la forme $\beta(x, z)$, alors $dom(\varphi^{|\beta(x,z)}) = \{x, z\}$.
- α est de la forme $\delta(p1, x, p2, z)$, alors $dom(\varphi^{|\delta(p1,x,p2,z)}) = \{x, z\}$.

De plus, par la définition de $\wedge$, nous obtenons

$$\forall v \in dom(\varphi^{|\alpha}).[\![v]\!]_{\varphi_{res}} = \gamma^{-1}([\![v]\!]_{\overline{\varphi}} \cap [\![v]\!]_{\overline{\psi}})$$

$$\Longrightarrow \forall v \in dom(\varphi^{|\alpha}).[\![v]\!]_{\overline{\varphi_{res}}} = [\![v]\!]_{\overline{\varphi}} \cap [\![v]\!]_{\overline{\psi}}$$

$\{\nexists\langle u_j,\psi_j\rangle \stackrel{\alpha}{\Longrightarrow} \langle u'_j,\psi'_j\rangle \longrightarrow \varphi_i\backslash\varphi'_i \supseteq \varphi''$

ou $\{\exists\{\langle u_j,\psi_j\rangle \stackrel{\alpha'}{\Longrightarrow} \langle u_k,\psi_k\rangle\}$ tel que pour chaque k.

$\exists\varphi'_k = close(\langle s'_i, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle, \langle u_k, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle)$

et $(\forall v \in fv(\alpha).[\![v]\!]_{\overline{\varphi'_k}} \subseteq [\![v]\!]_{\overline{\varphi'_i}} \cap [\![v]\!]_{\overline{\psi_k}})$ et $\mathcal{P}(s'_i, u_k, \varphi'_k)$

et $\vee_k\varphi'_k \supseteq \varphi''\}$

2 if $\nexists\langle u_j,\psi_j\rangle \stackrel{\alpha}{\Longrightarrow} \langle u'_j,\psi'_j\rangle$

3 return $(\varphi_i\backslash\varphi'_i)$

$\{\varphi_i\backslash\varphi'_i \supseteq \varphi''$ et $\varphi'' \subseteq \varphi_i\}$

4 else

$\{\exists\{\langle u_j,\psi_j\rangle \stackrel{\alpha'}{\Longrightarrow} \langle u_k,\psi_k\rangle\}$ tel que pour chaque k.

$\exists\varphi'_k = close(\langle s'_i, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle, \langle u_k, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle)$ et

$(\forall v \in fv(\alpha).[\![v]\!]_{\overline{\varphi'_k}} \subseteq [\![v]\!]_{\overline{\varphi'_i}} \cap [\![v]\!]_{\overline{\psi_k}})$ et $\mathcal{P}(s'_i, u_k, \varphi'_k)$

et $\vee_k\varphi'_k \supseteq \varphi''\}$

5 $simul\_on\_\alpha :=$

6 for each $\langle u_j,\psi_j\rangle \stackrel{\alpha'}{\Longrightarrow} \langle u_k,\psi_k\rangle$

$\{\forall v \in fv(\alpha).[\![v]\!]_{\overline{close(\langle s'_i, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle, \langle u_k, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle)}} \subseteq [\![v]\!]_{\overline{\varphi'_i}} \cap [\![v]\!]_{\overline{\psi_k}}$

et $close(\langle s'_i, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle, \langle u_k, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle) \subseteq (\varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}) \subseteq \varphi'_i$

et $\mathcal{P}(s'_i, u_k, close(\langle s'_i, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle, \langle u_k, \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}\rangle))\}$

7 $next\_\varphi := \varphi_i'^{|\alpha} \wedge \psi_k^{|\alpha'}$

$\{\forall v \in fv(\alpha).[\![v]\!]_{\overline{close(\langle s'_i, next\_\varphi\rangle, \langle u_k, next\_\varphi\rangle)}} \subseteq [\![v]\!]_{\overline{\varphi'_i}} \cap [\![v]\!]_{\overline{\psi_k}}$ et

$close(\langle s'_i, next\_\varphi\rangle, \langle u_k, next\_\varphi\rangle) \subseteq next\_\varphi \subseteq \varphi'_i$ et

$\mathcal{P}(s'_i, u_k, close(\langle s'_i, next\_\varphi\rangle, \langle u_k, next\_\varphi\rangle))\}$

8 $\varphi_k := close(\langle s'_i, next\_\varphi\rangle, \langle u_k, next\_\varphi\rangle)$

$\{\forall v \in fv(\alpha).[\![v]\!]_{\overline{\varphi_k}} \subseteq [\![v]\!]_{\overline{\varphi'_i}} \cap [\![v]\!]_{\overline{\psi_k}}$ et $\varphi_k \subseteq next\_\varphi \subseteq \varphi'_i$ et $\mathcal{P}(s'_i, u_k, \varphi_k)\}$

9 $\vee_k\varphi_k$

$\{\vee_k\varphi_k \supseteq \varphi'' \subseteq \varphi'_i\}$

$\{simul\_on\_\alpha \supseteq \varphi'' \subseteq \varphi_i'\}$

$\{\varphi_i^{|\alpha} \backslash \varphi_i'^{|\alpha} \vee simul\_on\_\alpha \supseteq \varphi'' \subseteq \varphi_i\}$

10 $no\_trans\_for\_\alpha := \varphi_i^{|\alpha} \backslash \varphi_i'^{|\alpha}$

$\{no\_trans\_for\_\alpha \vee simul\_on\_\alpha \supseteq \varphi''$ et $\varphi'' \subseteq \varphi_i\}$

11 return $no\_trans\_for\_\alpha \vee simul\_on\_\alpha$

Nous avons donc la précondition de l'appel de la fonction *match* pour obtenir les post-conditions désirées.

Nous voyons d'après la pré-condition à la fonction *close* que, si on traduit la propriété $\mathcal{P}(s_i, u_j, \varphi'')$ par $(s_i, u_j) \in \mathcal{B}^{\varphi'', \varphi''}$, nous obtenons les conditions de la définition 5.4. Donc, l'algorithme calcule bien la bisimulation symbolique de deux noeuds.

De plus, remarquons qu'en remontant depuis la post-condition avec $Sim(s_0, u_0) = \varphi_0$, nous obtenons bien $(s_0, u_0) \in \mathcal{B}^{\varphi_0, \varphi_0}$. (Puisque les changements de variables appropriés ont été effectués pour assurer une correspondance entre les deux systèmes et que les deux graphes étudiés sont tels que $\varphi_0 \supseteq \psi_0$, il est tout à fait correct de dire $(s_0, u_0) \in \mathcal{B}^{\varphi_0, \varphi_0}$ au lieu de $(s_0, u_0) \in \mathcal{B}^{\varphi_0, \psi_0}$). □