Vérification des protocoles cryptographiques : Comparaison des modèles symboliques avec une application des résultats — Etude des protocoles récursifs

Heinrich Hördegen

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Vérification des protocoles cryptographiques : Comparaison des modèles symboliques avec une application des résultats — Étude des protocoles récursifs

THÈSE

présentée et soutenue publiquement le 29 Novembre 2007

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Heinrich Hördegen

Composition du jury

Président :	Gaétan Hains
Rapporteurs :	Gaétan Hains Ralf Treinen
Examinateurs :	Isabelle Chrisment Véronique Cortier François Laroussinie Michaël Rusinowitch

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Table des matières

Table des figures

Chapitre 1

Introduction

La sécurité de l'information a des buts multiples : les informations doivent être protégées contre l'accès, l'utilisation, la publication, la destruction ou la modification non autorisé. La protection doit garantir la confidentialité, l'intégrité ou l'accessibilité des informations, selon les exigences de la situation particulière. Dans ce contexte très général, les informations peuvent avoir la forme des documents imprimés, électroniques ou autres.

L'histoire de la sécurité de l'information commence avec l'existence des documents écrits. Depuis l'antiquité, les hommes étaient concernés par le problème de l'authentification des documents, leur protection contre des altérations ou la conservation des secrets. Des moyens tels que les sceaux ou le chiffrement de César ont été développés pour atteindre ce but. Pendant la deuxième guerre mondiale, la sécurité de l'information est devenue une tâche primordiale. De multiples professions dans ce domaine se sont développées depuis alors, comme, par exemple, les cryptographes ou les cryptanalystes. Avec l'arrivée de la téléphonie, du traitement électronique des informations et de l'internet, la sécurité des information est aujourd'hui une préoccupation de tout le monde qui trouve des applications dans le secteur civil, ainsi que militaire. La formation de nombreuses organisations qui sont chargées de réglementer et surveiller le domaine de la sécurité des information est témoin de ce développement. Des organisations qui s'occupent de la sécurité de l'information sont par exemple ISO, NIST, ISOC et ISF.

Cette thèse se place dans le cadre de la sécurité de l'information. Plus précisément, elle traite de l'échange des informations à travers des média non sécurisés. Pour bien structurer un tel échange, des protocoles de communication sont utilisés. Dans le contexte de la sécurité de l'information, nous les appelons les protocoles cryptographiques.

1.1 L'intérêt des protocoles cryptographiques

Un protocole consiste en deux ou plusieurs programmes qui communiquent à travers des canaux non sécurisés tels que la Toile ou les réseaux de téléphonie. La description d'un protocole est en général courte. Pour en donner un exemple, on considère le protocole suivant :

$$\begin{aligned} A \to B : &\quad \langle A, N_A \rangle \\ B \to A : &\quad N_B \end{aligned}$$

Cette description, certainement informelle, doit être lue de la manière suivante : le protocole consiste en deux programmes A et B, dits *rôles*. Dans un premier temps, le rôle A qui est censé

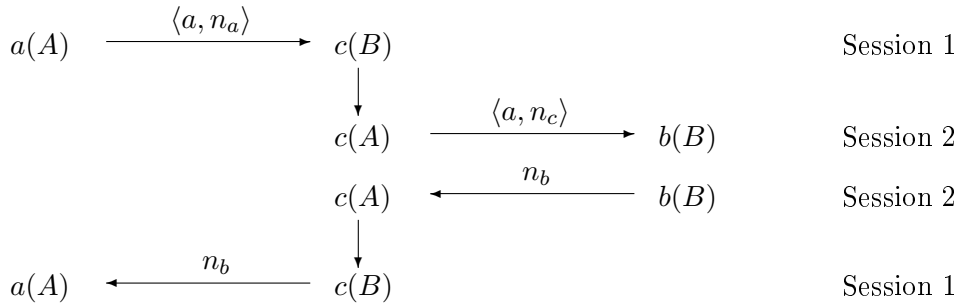


FIG. 1.1 – Un exemple introductif.

---

démarrer l'exécution du protocole envoie un message $\langle A, N_A \rangle$ au rôle B. Le message est une paire contenant l'identité du rôle A, ainsi qu'un nombre N_A. La notation N_A indique usuellement qu'il s'agit d'un nombre aléatoire généré par le rôle A. Un tel nombre est aussi appelé *nonce*. Lorsque le rôle B reçoit le message $\langle A, N_A \rangle$, il répond en envoyant un deuxième nonce N_B à l'identité indiquée par la première donnée du message $\langle A, N_A \rangle$, c'est-à-dire à A. L'exécution du protocole se termine sur ce dernier envoi.

Il est important de faire la différence entre les rôles d'un protocole et les identités qui les exécutent. Ces identités sont les *agents* ou *participants*. Un protocole peut être exécuté plusieurs fois entre différents participants. Une seule exécution d'un protocole est appelée une *session*.

Pour bien illustrer la différence entre les rôles et les agents, on considère le scénario de la figure 1.1 où deux session intercalées sont exécutées simultanément. Les rôles du protocole que les agents jouent respectivement sont indiqués entre parenthèses. Un agent a, jouant rôle A, envoie un message $\langle a, n_a \rangle$ à un agent c, jouant rôle B du protocole. Au lieu de répondre immédiatement, l'agent c ouvre une deuxième session en tant que rôle A. Il émet le message $\langle a, n_c \rangle$, destiné à l'agent b, jouant rôle B. Comme n_c est un nombre aléatoire, l'agent b ne peut pas se rendre compte que le nonce n'a pas été généré par a. Cette provenance lui est pourtant indiquée par l'identité d'agent a contenue dans le message $\langle a, n_c \rangle$. L'agent b répond alors en envoyant le nonce n_b à l'agent c. Ici, la deuxième session se termine. Ensuite, l'agent c transmet le nonce n_b à l'agent a. L'agent a ne peut pas savoir non plus que le nonce qu'il reçoit n'a pas été généré par l'agent c. Cela finit la première session.

Quelles leçon peut-on tirer de l'exemple ? On peut d'abord se demander quel agent pense avoir communiqué avec quel autre agent. Lorsque l'agent a a fini sa session, rien ne lui permet de savoir à quel agent en tant que répondeur il a eu affaire. En effet, lorsqu'il a émis le message $\langle a, n_a \rangle$, n'importe quel autre agent d lui aurait pu répondre en envoyant un nonce n_d. On conclut que l'agent a ne peut pas distinguer ses interlocuteurs dans ce protocole. Quant à l'agent c, il a un motif de penser que le message $\langle a, n_a \rangle$ provient de l'agent a. Cependant, il n'en peut pas être sûr, comme la suite de l'exemple montre. L'agent c, au lieu de répondre à l'agent a, envoie le message $\langle a, n_c \rangle$ à l'agent b. Celui-ci a une raison pour penser que le message vient de l'agent a. Il n'en est rien. En effet, il vient de l'agent c. Pour la même raison, l'agent c ne peut pas être sûr d'avoir reçu un message de l'agent a. Il est clair que ce protocole n'est guère apte à assurer l'authenticité des messages, respectivement des identités des agents. Dans notre cas concret, cela veut par exemple dire que le protocole ne fournit pas de moyen à l'agent b d'être convaincu que le message $\langle a, n_c \rangle$ provient de l'agent a. Il est encore plus évidant que ce protocole n'est pas capable d'assurer le secret des données transmises comme, par exemple, du nonce n_b destiné à l'agent a. En effet, l'agent c le connaît aussi.

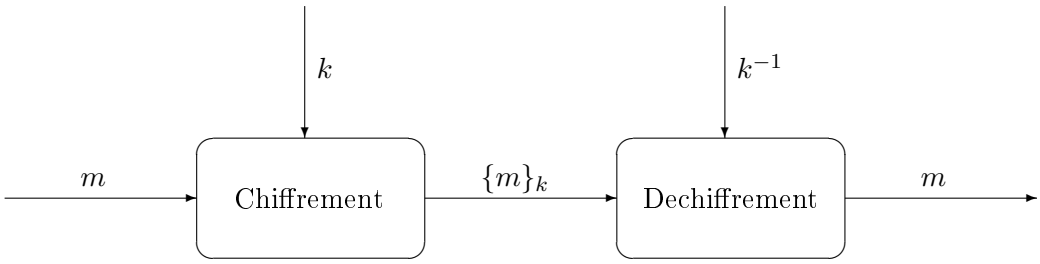


FIG. 1.2 – Le schéma général de chiffrement.

D'où l'intérêt des *protocoles cryptographiques*. Il n'est en effet pas acceptable que, lors d'une transaction bancaire, d'une communication avec un téléphone portable ou du transfert de données d'un site web sécurisé, des données confidentielles soient découvertes et lisibles pour des tierces personnes. Dans un environnement qui permet à pratiquement tout le monde de lire les messages échangés, ou encore à des individus malintentionnés de les retenir, rejouer ou modifier, on a alors intérêt à coder les message de manière qu'ils ne soient déchiffrable que par des personnes autorisées. Pour ce faire, on utilise des opérations dites *primitives cryptographiques*. L'emploi de bonnes primitives cryptographiques n'implique cependant pas automatiquement qu'un protocole possède les propriétés qu'on désire, à savoir que la façon même dont les messages sont échangés y joue un rôle important.

Ce chapitre est structuré comme suit : les sections 1.2 et 1.3 présentent les notions cryptographiques dont tout le reste de cette thèse se servira. La section 1.4 introduit les notions d'intrus, de propriété de sécurité et d'attaque. La section 1.5.2 présente le cadre de notre travail. Dans la section 1.5, nous présentons le modèle avec lequel nous travaillerons. Enfin, la section 1.6 présente le plan pour le reste de cette thèse.

1.2 Les modes de chiffrement

Le chiffrement est une opération qui vise à obtenir à partir d'un message m un autre message m' de façon à ne plus reconnaître le message d'origine. Pour ce faire, on utilise une clef. Celle-ci consiste généralement en un grand nombre. Soit k une clef de chiffrement. On dénote le message obtenu par chiffrement $m' = \{m\}_k$. Une personne qui connaît le message m' et la clef inverse de k, souvent notée k^{-1}, peut retrouver le message d'origine m. Cette opération est appelé déchiffrement. La figure 1.2 résume ce schéma. Il existe plusieurs modes de chiffrement qu'on expliquera dans la suite.

Le chiffrement symétrique

Ce mode de chiffrement est le plus ancien. Soit $m' = \{m\}_k$ le chiffrement de m par la clef k. S'il est possible, en connaissant la clef de chiffrement k et l'algorithme de chiffrement, de déduire la clef de déchiffrement k^{-1} dans un temps de calcul pas trop important, alors on dit que le chiffrement est symétrique. Pour qu'il s'agisse d'un chiffrement symétrique, on définit normalement que le temps de calcul de la clef de déchiffrement ne doit pas excéder la croissance d'un polynôme dans la longueur de la représentation binaire de la clef de chiffrement [MvOV96]. Nous disons alors aussi que la clef inverse k^{-1} est *calculatoirement facile* à trouver.

Exemple 1 (Chiffrement de César)**.** Étant donné un texte en français m, le chiffrement de César correspond à décaler chaque lettre d'un certain nombre n de lettres dans l'alphabet. Si on choisi par exemple $n = 3$, on remplacera les "a" dans m par "d", les "b" par "e", et ainsi de suite. Si on identifie chaque lettre à sa position x dans l'alphabet, la fonction de chiffrement est alors donnée par $E_n(x) = (x + n) \bmod 26$ où 26 est le nombre de lettres de l'alphabet. Il est clair qu'en connaissant la clef $k = n$, la clef inverse $k^{-1} = -n$ est facile à trouver. La fonction de déchiffrement est donc donnée par $D_n(x) = (x - n) \bmod 26$.

Dans la plupart de schémas de chiffrement symétrique, la clef de chiffrement k est identique à la clef de déchiffrement k^{-1}, c'est-à-dire $k = k^{-1}$. Un problème majeur de ce mode de chiffrement est le fait que deux participants qui veulent communiquer doivent se mettre d'accord sur une clef. Cela nécessite l'échange des données à travers un canal sécurisé. Des exemples d'algorithmes de chiffrement symétrique sont DES [DES77] et AES [AES01]

Le chiffrement asymétrique

Le chiffrement asymétrique est un autre schéma plus récent. Un premier algorithme a été proposé par Diffie et Hellman [DH76]. D'autres travaux sur ce mode de chiffrement comprennent l'algorithme de chiffrement RSA, développé par Rivest, Shamir et Adleman [RSA78a] ou encore le schéma de chiffrement développé par ElGamal [ElG85].

L'idée principale est la suivante [MvOV96] : un participant choisit une paire de clefs (k, k^{-1}), de façon à ce que, pour un message chiffré donné $\{m\}_k$, il soit calculatoirement difficile de retrouver m, même si on connaît la clef k et l'algorithme de chiffrement. Dans ce contexte, on définit normalement que le calcul est *difficile*, si le temps de calcul excède la croissance d'un polynôme dans la longueur de la représentation binaire de la clef de chiffrement. D'autre part, il doit être facile de retrouver m, si on connaît la clef de déchiffrement k^{-1}. Si le schéma de chiffrement satisfait ces conditions, le participant peut publiquement faire accessible la clef k. Pour cette raison, ce mode de chiffrement est aussi nommé *chiffrement à clef publique*. Le participant garde en secret seulement la clef de déchiffrement k^{-1}. Si un autre participant veut lui envoyer un message, il doit récupérer la clef publique k et chiffrer le message avec celle-ci, puis l'envoyer. La clef inverse k^{-1} n'étant connue que par le participant qui a généré la clef k, c'est lui seul qui est capable de retrouve le message d'origine.

Le chiffrement à clef publique a l'avantage de ne pas nécessiter un canal sécurisé pour échanger une clef. Un inconvénient est le fait qu'une clef publique doit être liée à un certificat qui prouve son origine. En effet, si cela n'est pas le cas, un agent malhonnête c peut demander à un agent a de lui envoyer une information m qui n'est pas à son intention. Pour ce faire, il donnera sa clef publique k_c à a en le faisant croire que c'est la clef publique d'un agent b qui, lui, a droit de connaître m. Si l'agent a n'a pas le moyen de s'assurer de la provenance de la clef k_c, il peut soit refuser d'envoyer l'information secrète, soit envoyer le message $\{m\}_{k_c}$. Les deux solutions ne sont alors pas satisfaisantes : dans le premier cas, l'agent a risque de ne pas concéder à l'agent b son droit, et, dans le deuxième cas, l'agent c peut retrouver m à l'aide de sa clef privée k_c^{-1}. L'agent a doit donc avoir un moyen de s'assurer de la provenance de la clef k_c.

Le chiffrement probabiliste

Les modes de chiffrement symétrique et asymétrique que nous venons de présenter souffrent d'un défaut majeur. Considérons que l'ensemble de messages possibles est petit, par exemple $\{oui, non\}$. Dans ce cas, une personne qui intercepte le message chiffré $m' = \{m\}_k$ et qui désire connaître le contenu m peut calculer $m'' = \{oui\}_k$, ou, dans le cas où la clef k n'est pas publique,

essayer de faire calculer m'' par un agent qui connaît la clef de chiffrement. Elle sait alors que, si $m' = m''$, alors $m = oui$, sinon $m = non$. Ce genre de faille est connu sous le nom d'*attaque à texte clair choisi*. Une telle attaque est bien sûr applicable à des ensembles de messages plus grands, tant qu'ils ne soient pas trop vastes. Un autre désavantage de ces schémas de chiffrement est le fait qu'un intrus peut parvenir, par une comparaison simple, à savoir, si un message quelconque a déjà été envoyé précédemment.

Pour prévenir ces défauts, la notion de chiffrement probabiliste a été développée par Goldwasser et Micali [GM84]. La solution consiste principalement en l'extension du domaine des messages. À cette fin, on ajoute, lors du chiffrement, du bruit à un message. Ainsi on n'émettra par exemple plus le message $\{oui\}_k$, mais le message $\{oui, n\}_k$ où n est un nombre aléatoire. Si on choisit l'ensemble dont le nombre n fait partie très vaste, une personne malintentionnée ne parviendra plus à chiffrer toutes les combinaisons possibles de paires de *oui* et d'un nombre aléatoire. Les attaques à texte clair choisi sont alors rendues inutiles. D'autre part, quelqu'un qui connaît la clef inverse est capable d'extraire pendant le déchiffrement l'information d'origine du message déchiffré, s'il est capable de distinguer entre un nombre aléatoire et la réponse qu'il attend. Des algorithmes bien connus pour le chiffrement probabiliste se trouvent dans [GM84,BG85].

1.3 Les autres primitives cryptographiques

La paire

Une fonction dont on se sert souvent est la fonction de paire. Elle est utilisée pour construire à partir de deux données m_1 et m_2 la paire $\langle m_1, m_2 \rangle$. Lorsqu'on parle d'une paire chiffrée, on écrira dans la suite plutôt $\{x, y\}_k$ au lieu de $\{\langle x, y \rangle\}_k$. Si nécessaire, on donnera des précisions.

Les fonctions de hachage

Dans ce paragraphe, nous entendons par la longueur d'une valeur le nombre de bits nécessaire pour représenter cette valeur. Nous donnons une définition informelle d'une fonction de hachage : une fonction de hachage *hash* prend comme entrée une valeur de longueur non bornée et renvoie une valeur de longueur n fixée. De plus, si on choisit une valeur m au hasard, la probabilité d'avoir $m' = hash(m)$ pour une valeur particulière m' de longueur n est 2^{-n} dans le cas idéal. La notion de fonction de hachage est fondamentale en cryptographie. Pour être utilisée dans ce contexte, on exige souvent deux propriétés supplémentaires [MvOV96] : premièrement, il doit être calculatoirement difficile de provoquer des collisions, c'est-à-dire qu'on suppose qu'un temps de calcul considérable est nécessaire pour trouver une valeur m' pour une valeur donnée m où $m' \neq m$, telle que $hash(m') = hash(m)$ et, deuxièmement, il doit être difficile de trouver une valeur m', pour une valeur donnée m de longueur n, telle que $hash(m') = m$.

Les fonctions de hachage connaissent de nombreuses applications en cryptographie, parmi lesquelles on peut citer les signatures digitales et les tests d'intégrité des données. Dans le premier cas, seulement le haché des données à signer est réellement signé. Cela suffit, car le fait qu'il est difficile de trouver une valeur m' pour une valeur donnée m de longueur n, tel qu'on ait $m = hash(m')$, évite qu'une signature puisse être attachée à un autre message en tant que preuve de provenance. Et le fait qu'il n'est pratiquement pas faisable pour un agent de trouver deux messages m et m', tels qu'on ait $hash(m) = hash(m')$, empêche qu'il signe m et affirme ultérieurement qu'il a signé m'. Par conséquent, il ne peut pas nier qu'il ait signé le message. Quant à l'intégrité des données, elle peut être assurée de la manière suivante : un agent calcule le haché des données à transmettre. Ensuite, il envoie les données avec son haché. L'agent receveur,

lui aussi, calcule le hachage. Si la valeur calculée par lui-même est égale à la valeur de hachage transmise, il y a une forte chance que les données n'aient pas subi d'altération pendant la transmission. L'émetteur peut aussi se servir d'une clef secrète qui a été établie entre les participants du protocole. Ainsi, il calcule $hash(\langle m, k\rangle)$. De cette façon, les données transmises sont signées. La valeur de hachage est alors appelé MAC (Message Authentication Code) dans la littérature, voir par exemple le protocole présenté dans [PCTS00].

La signature

Une signature digitale est une pièce de donnée attachée à une information m qui permet aux participants d'un protocole de déterminer la provenance de m. Soit k_a, k_a^{-1} deux clefs choisies par un agent a où la clef k_a est destiné à être publiée, tandis la clef k_a^{-1} reste le secret de a. Pour réaliser un schéma de signature, deux algorithmes sont nécessaires [MvOV96, PS00].

L'algorithme de signature S prend comme entrée la donnée à signer et la clef privée k_a^{-1} et calcule une signature $s = S(m, k_a^{-1})$. L'agent peut alors envoyer la paire $\langle m, s\rangle$.

Le deuxième algorithme V sert à vérifier la signature et est utilisé par les personnes recevant un message signé. Il prend comme entrée le message m, la signature s et la clef publique k_a et renvoie comme résultat soit *vrai*, soit *faux*. Il a les propriétés suivants : il renvoie *vrai*, si et seulement si s est la signature de m signé avec la clef k_a^{-1}. En outre, il est calculatoirement difficile pour n'importe quel agent sauf a de trouver une signature s pour n'importe quel message m', tel que $V(m', s, k_a) = vrai$.

Il existent plusieurs possibilités pour réaliser ce schéma [RSA78b, EG85]. Pour les messages courts, on peut utiliser un algorithme de chiffrement asymétrique avec la clef privée comme clef de chiffrement. Un autre participant peut alors déchiffrer le message avec la clef publique et ainsi, à la fois, être sûr d'avoir identifié l'émetteur et récupérer le contenu du message.

Pour les messages plus longs, au moins deux solutions sont possibles : soit, on hache le message m avec une fonction de hachage *hash* et chiffre le haché à l'aide de la clef privée. Un agent a enverra alors la paire $\langle m, \{hash(m)\}_{k_a^{-1}}\rangle$. Soit, on établit une clef secrète entre les participants et le message est signé à l'aide d'un MAC. On envoie alors le pair $\langle m, hash(\langle m, k\rangle)\rangle$.

Notons que, dans la variante qui chiffre tout le message au moyen d'une clef privée, un message ne peut pas être dissocié de sa signature, tandis que, dans les deux dernières versions, la signature peut être envoyée, sans qu'un participant puisse en déduire le message.

1.4 Les notions importantes pour la vérification

1.4.1 L'intrus

L'*intrus* ou *adversaire* est un agent malhonnête qui cherche à déjouer les propriétés que le protocole est censé préserver. On en distingue habituellement deux types : l'*adversaire passif* et l'*adversaire actif*.

Le premier type ne peut qu'écouter le trafic dans le réseau. Son seul moyen d'agir est d'essayer de déduire des information des messages qu'il a vus circuler. C'est un adversaire à capacités faibles. Un adversaire actif peut, par contre, bloquer et envoyer des messages, les analyser et modifier ou encore construire des messages tout à fait nouveaux. Il s'agit évidemment d'un modèle d'adversaire plus fort. C'est le modèle est utilisé par Dolev et Yao dans l'article [DY81]. Étant donnée sa capacité supérieure, c'est le modèle principalement considéré aujourd'hui. Aussi tous les outils de vérification automatique l'utilisent-ils.

1.4.2 Les propriétés à vérifier

Étant donné que les domaines d'application des protocoles cryptographiques sont divers, les tâches que ceux-ci sont censés assurer sont nombreuses. Le plus souvent sont considérées la capacité d'un protocole de *garder une donnée secrète*, ainsi que de permettre à un participant de *vérifier l'authenticité d'une donnée*. Dans le jargon technique, nous disons aussi qu'un protocole satisfait la *propriété du secret*, ainsi que la *propriété de l'authentification*. Au moins deux variantes importantes existent de cette dernière propriété, appelées *authentification faible* et *authentification forte*. Nous concentrons nos explications sur les propriétés mentionnées, mais, comme nous l'avons dit, il y a une multitude d'autres propriétés qu'un protocole, selon la situation, doit satisfaire.

Le secret

Un bon nombre de protocoles cryptographiques ont été proposés dans le but de permettre la distribution d'une donnée m dans un groupe de deux ou plusieurs participants, sans que un agent qui n'appartient pas à ce groupe puisse l'apprendre. Un protocole qui peut assurer cette tâche important satisfait la *propriété du secret de* m. Les applications de ces protocoles se trouvent par exemple dans la distribution des clefs de chiffrement symétrique. Une autre application est la distribution d'un secret pour permettre à un agent de s'authentifier, lorsqu'il est demandé par un autre agent qui connaît le secret. La propriété du secret est une des propriétés clefs dans le domaine de la sécurité d'informations.

L'authentification

Après avoir échangé des données, les participants d'un protocole veulent souvent savoir, si les données reçues sont authentiques. Pour se convaincre, ils lancent un processus pendant lequel l'origine des données est vérifiée. Ce processus est communément connu sous le nom d'*authentification*. Nous disons que l'authentification d'un message m d'un participant a auprès d'un participant b a réussi, si le participant b est convaincu que le message m provient du participant a. Pour que cela soit le cas, le protocole doit assurer que l'agent b obtient des preuves de la provenance du message m. Un protocole qui assure cette tâche satisfait la *propriété de l'authentification*. Au moins deux versions de cette propriété existent.

La *propriété de l'authentification faible* exige qu'un agent puisse vérifier l'authenticité d'un message, mais il ne peut pas être sûr que ce message a été émis lors de la session courante. L'intrus peut par exemple le stocker et rejouer lors d'une nouvelle session. Un participant d'un protocole qui satisfait la propriété de l'authentification faible peut donc être sur que son interlocuteur a participé dans une session du protocole, mais il ne sais pas dans laquelle.

Un protocole qui satisfait la *propriété de l'authentification forte* donne plus de garanties. Un participant qui exécute un tel protocole peut être sûr que le message en question à été émis dans la session courante par celui qu'il croit être son interlocuteur. Si un intrus lui envoie un message émis lors d'une autre session, il peut le détecter, même s'il a été émis par l'interlocuteur dont il l'attend.

Des différentes notions d'authentification se trouvent par exemple dans [DVOW92, Low97c] et [CMP05].

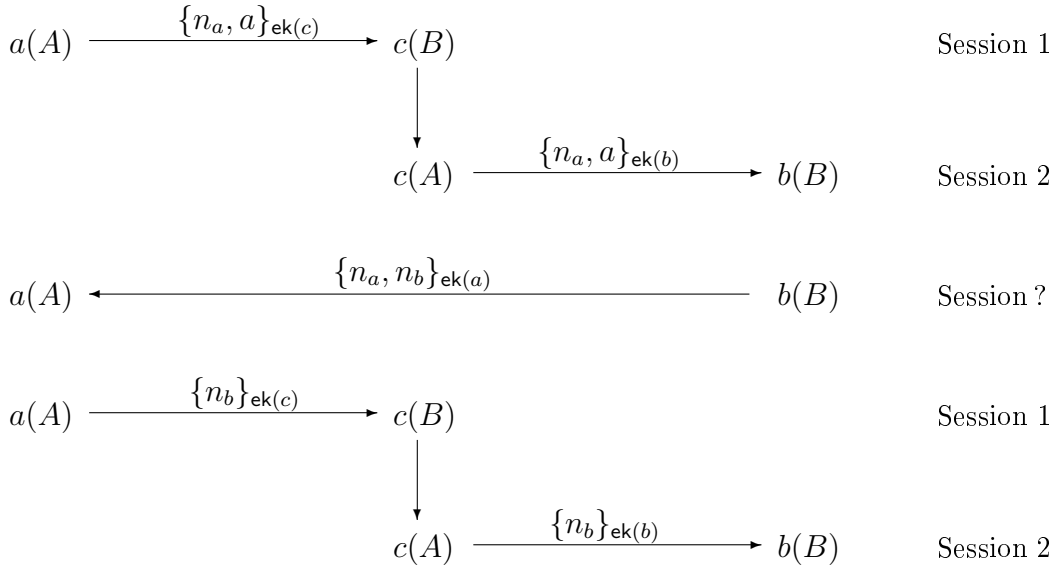


FIG. 1.3 – L'attaque contre le protocole de Needham-Schroeder, due à G. Lowe. Dans la session marquée par (?), l'agent b pense répondre dans la session 2, mais l'agent a pense recevoir ce message dans la session 1.

1.4.3 Les attaques

L'homme au milieu

Le premier type d'attaque présenté ici est connu sous le nom d'*attaque de l'homme au milieu* (man-in-the-middle-attack). Considérons un exemple classique qui se sert d'un algorithme de chiffrement asymétrique. Il s'agit du protocole de Needham-Schroeder [NS78] :

$$\begin{aligned} A \to B : &\quad \{N_A, A\}_{\mathsf{ek}(B)} \\ B \to A : &\quad \{N_A, N_B\}_{\mathsf{ek}(A)} \\ A \to B : &\quad \{N_B\}_{\mathsf{ek}(B)} \end{aligned}$$

Pour démarrer le protocole, le rôle A envoie son nom et un nonce N_A à B, chiffré avec la clef publique $\mathsf{ek}(B)$ du rôle B. Nous dénotons la clef inverse qui n'est connue que par B, par $\mathsf{dk}(B)$. À l'aide de cette clef, B peut déchiffrer le message et envoyer en suite la paire $\langle N_A, N_B \rangle$ à A, chiffrée cette fois-ci avec la clef publique de A. Dans la dernier étape, le rôle A, après avoir, en utilisant sa clef privée $\mathsf{dk}(A)$, déchiffré de son côté le message reçu, répond avec l'envoi du message $\{N_B\}_{\mathsf{ek}(B)}$.

Ce protocole a été proposé pour assurer l'authentification. En effet, à la deuxième étape, lorsque A reçoit le message $\{N_A, N_B\}_{\mathsf{ek}(A)}$, il peut en déduire, reconnaissant son nonce N_a, que ce message vient de B. En répondant par le message $\{N_B\}_{\mathsf{ek}(B)}$, les deux participants pensent alors qu'ils sont les seuls à connaître le nonce N_B. Ainsi, ils pourraient l'utiliser dans un autre protocole pour s'authentifier l'un à l'autre.

Cependant, comme l'attaque de [Low95] montre, ce protocole ne satisfait pas la propriété de l'authentification. En effet, une attaque existe avec seulement deux sessions intercalées. Nous la décrivons dans la figure 1.3. Dans cette attaque, l'agent a, jouant le rôle A, commence une com-

munication avec l'agent c, y jouant le rôle B. Au lieu de répondre immédiatement, c commence une deuxième session en tant que rôle A avec l'agent b qui joue ici le rôle du répondeur B. Il lui envoie la paire $\langle n_a, a\rangle$ qu'il a reçue de a, mais chiffrée avec la clef publique de b. Celui-ci pense que le message a été envoyé par a. Il répond alors en envoyant $\{n_a, n_b\}_{\mathsf{ek}(a)}$. L'agent a ne peut pas détecter que le nonce n_b ne vient pas de c. Il envoie comme confirmation le message $\{n_b\}_{\mathsf{ek}(c)}$ à c, puis c transmet le nonce n_b à b, chiffré avec la clef publique de b. Celui-ci croit maintenant avoir communiqué avec a, ce qui est évidemment faux. Pour remédier à cette faille, la correction suivante a été proposée par G. Lowe [Low96] :

$$\begin{aligned} A \to B: &\quad \{N_A, A\}_{\mathsf{ek}(B)} \\ B \to A: &\quad \{N_A, N_B, B\}_{\mathsf{ek}(A)} \\ A \to B: &\quad \{N_B\}_{\mathsf{ek}(B)} \end{aligned}$$

Dans la deuxième étape, l'agent qui joue le rôle B ajoute son identité, ce qui empêchera un homme au milieu de faire passer ce message incontesté auprès d'un rôle A qui n'a pas exécuté le protocole avec cet agent.

Attaques basées sur des propriétés algébriques

D'autres attaques reposent sur le fait que les primitives cryptographiques peuvent posséder des propriétés algébriques. Cela offre, souvent en combinaison avec une confusion de type, des possibilités d'attaque. Considérons une version légèrement modifiée du protocole de Needham-Schroeder. La modification ne concerne que le premier message qui est remplacé par le message $\{A, N_A\}_{\mathsf{ek}(B)}$. L'ordre des données N_A et A est donc inversé. L'attaque que nous présentons a pour la première fois été décelée par [MS01]. Elle repose sur hypothèse que la fonction de la paire est associative, c'est-à-dire $\langle\langle x, y\rangle, z\rangle = \langle x, \langle y, z\rangle\rangle$. C'est le cas, si les messages sont concaténés. Le schéma d'attaque se trouve dans la figure 1.4. L'intrus i, jouant le rôle A commence une session avec l'agent b qui joue le rôle B. L'agent b pense parler avec l'agent a et utilise la clef celui-ci. Le nonce n_b est donc destiné à l'intention de l'agent a. Après avoir reçu la réponse de b, l'intrus entame une deuxième session en tant que rôle A, cette fois-ci avec l'agent a, jouant le rôle B. Pour commencer cette session, il utilise le message qu'il a obtenu lors de la première session. L'agent a qui ne sais pas si le contenu du message reçu doit être lu comme $\langle\langle i, n_b\rangle, b\rangle$ ou $\langle i, \langle n_b, b\rangle\rangle$ suppose que la deuxième façon de lire est la juste. Il met donc la paire $\langle n_b, b\rangle$ à la place du nonce N_A et chiffre le tout avec la clef de l'intrus. Finalement, l'intrus peut déchiffrer ce message et apprendre le nonce n_b, originalement émis par l'agent b à l'intention de l'agent a. L'intrus peut maintenant s'authentifier auprès de l'agent b comme agent a. Notons que cette attaque repose sur une confusion de types en combinaison avec l'associativité de la fonction de la paire. Cependant, une confusion de types de cette manière semble naïve et facile à empêcher, si on se donne les moyens de pouvoir vérifier les types des données.

D'autres propriétés algébriques qu'on trouve parfois dans les primitives cryptographiques sont la commutativité (par exemple $\{\{x\}_{k_b}\}_{k_a} = \{\{x\}_{k_a}\}_{k_b}$) ou l'homomorphisme (par exemple $\{\langle x, y\rangle\}_k = \langle\{x\}_k, \{y\}_k\rangle$). Un résumé sur les protocoles utilisant des fonction avec des propriétés algébriques se trouve dans [CDL06].

Attaque par rejeu

Enfin, il y a un groupe d'attaques où des messages anciens sont rejoués par un participant malhonnête. Ainsi, dans l'attaque présentée dans la figure 1.3, on peut considérer le mes-

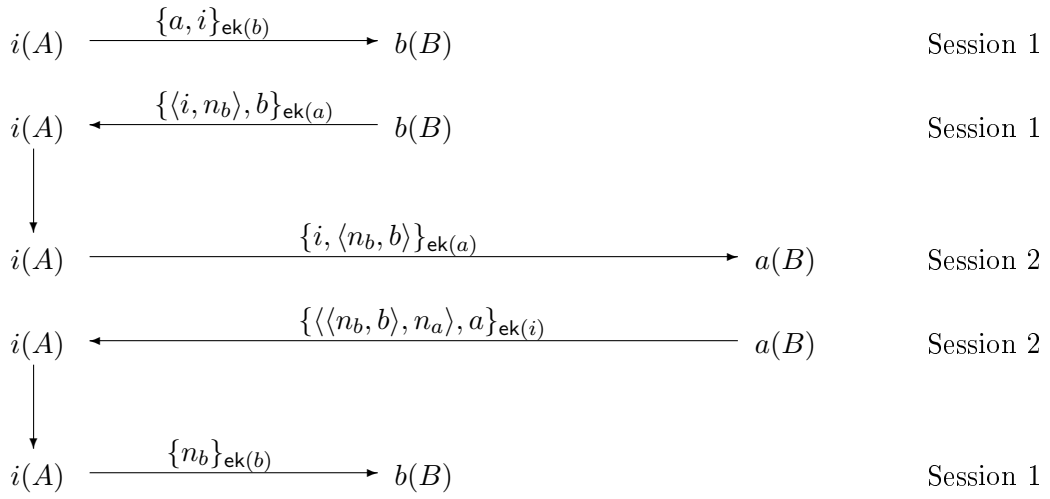


FIG. 1.4 – L'attaque contre le protocole de Needham-Schroeder modifié

sage $\{n_a, n_b\}_{\mathsf{ek}(a)}$, envoyé par l'agent b à l'agent a, en fait comme étant réutilisé par l'agent c, dans le sens où c le reçoit de b et l'achemine vers a.

1.5 Les modèles de vérification

Il y a plusieurs approches pour modéliser des protocoles cryptographiques. Premièrement, on peut utiliser un *modèle calculatoire*. Cette approche a été développé dans [BM82, Yao82, GM84, Gol93]. Elle permet de raisonner sur des questions de complexité et de probabilité, car les primitives cryptographiques sont considérées comme des vrais algorithmes et l'intrus y est modélisé comme un algorithme qui peut s'exécute efficacement. Le fait qu'un protocole satisfait une propriété dans ce modèle garantit qu'il n'existe pas d'algorithme efficace pour la violer. Ce modèle permet donc d'obtenir des résultats forts. Le point faible consiste en le fait que les preuves sont souvent pénibles et se prêtent mal à une automatisation.

Dans cette thèse, nous n'utilisons pas de modèle calculatoire, mais des *modèles symboliques*. Nous donnons d'abord un bref aperçu sur les caractéristiques des modèles symboliques utilisés. Ensuite, nous présentons les travaux reliés à cette approche.

1.5.1 Le modèle symbolique utilisé dans cette thèse

La notation que nous avons jusqu'alors utilisée pour décrire des protocoles cryptographiques révèle de nombreuses ambiguïtés. Nous avons par exemple présenté une attaque qui repose sur une confusion de types. Une telle attaque peut facilement être empêchée, si les participants testent les types des données transmises. Cependant, dans le système informel de spécification, le comportement des participants n'est pas clairement décrit. D'autres ambiguïtés sont décrites dans [Aba00]. D'où la nécessité de commencer l'étude des protocoles cryptographiques par une modélisation rigoureuse. Ensuite, une vérification formelle à l'aide de la modélisation permet de conclure que le protocole ne révèle en aucun cas une certaine faille ou qu'il préserve une certaine propriété. Les conclusions n'étant valides que pour la modélisation, elles permettent néanmoins souvent de détecter des attaques ou, au moins, des points faibles que la spécification d'un protocole peut comporter. Le point fort d'un modèle symbolique est le fait qu'il permet d'obtenir des

preuves simplifiées et souvent automatisées. Nous décrivons brièvement les caractéristiques des modèles symboliques utilisés dans cette thèse.

Les messages

Dans les modèles symboliques, les messages sont généralement représentés par des termes. Pour cette raison, la définition de nos modèles symboliques passe par la définition récursive d'une *algèbre de termes*. Par exemple, si k et m sont des termes, le chiffrement symétrique de m par k est noté par $\{\!|m|\!\}_k$. Les primitives cryptographiques sont donc représentées par des *symboles fonctionnels*. Les agents et nonces sont normalement modélisés par des *constantes*.

Les primitives cryptographiques

Les opérations des primitives cryptographiques sont modélisées par des *règles de déduction*, aussi appelées des *opérations Dolev-Yao* d'après les auteurs de l'article [DY81]. Ainsi, on peut par exemple définir la règle suivante : une personne qui connaît le terme $\{\!|m|\!\}_k$ peut en déduire le terme m, si et seulement si elle possède la clef k. Les primitives cryptographiques sont donc considérées comme des boîtes noires, c'est-à-dire qu'elles sont sûres par définition et qu'il n'existe pas d'algorithme pour les casser. Par conséquent, la notion des *informations partielles* n'existe pas. Une personne ne connaissant que le terme $\{\!|m|\!\}_k$ ne peut se procurer aucune information sur m. Cette hypothèse forte rend les propriétés prouvées moins fiables, car il peut quand même y avoir un algorithme qui les viole.

Les systèmes de transitions

Pour donner une sémantique à la définition d'un protocole, nous utilisons des *systèmes de transitions*. Dans un tel système, l'exécution d'un protocole est modélisée par une séquence finie d'*états globaux* et des *transitions* entre ces états globaux. Un état global contient tous les informations sur les messages échangés et les *états locaux* des agents. L'état local d'un agent est comparable à l'état de son mémoire à un certain moment d'exécution du protocole. Nous obtenons ainsi une *trace d'exécution* d'un protocole qui est une suite alternante d'états globaux et de transitions entre ceux-ci. Nous pouvons alors exprimer les propriétés d'un protocole comme des propriétés des traces. Par exemple, nous pouvons demander qu'aucune trace d'exécution d'un protocole ne contient un état global qui permet de déduire le secret.

Le canal de communication

L'intrus contrôle entièrement l'échange des messages. Capable d'initialiser le protocole avec d'autres participants, il peut intercepter, rejouer ou modifier les messages pendant l'exécution. Les transitions de notre système de transitions sont considérées comme étant provoquées par l'intrus. Une transition correspond donc à une de ses capacités.

1.5.2 Modèles symboliques — les travaux reliés

Un des premiers travaux et l'article probablement le plus cité dans le domaine de la vérification formelle des protocoles cryptographiques est celui de Dolev et Yao [DY81] dont les idées ont été reprises dans [DEK82]. Dans ces articles, les auteurs représentent les protocoles par règles de réécriture. Le travail se concentre sur une classe de protocoles à deux participants autorisant le chiffrement asymétrique. La propriété considérée est le secret. Dans leur formalisme, une donnée

reste secrète si elle n'est pas accessible par réécriture. Les auteurs ont remarqué qu'un protocole, bien qu'il soit sûr en présence d'un intrus passif, puisse révéler des failles en présence d'un intrus actif. L'article de Book et Otto [BO86] reprend le travail de Dolev et Yao en considérant aussi des propriétés d'authentification.

Depuis les travaux de Dolev et Yao, la communauté scientifique a reconnu qu'une modélisation symbolique permet d'écrire des preuves simplifiées non seulement pour certaines protocoles, mais souvent pour des classes de protocoles. Cette découverte a mené au développement de nombreux modèles symboliques, à savoir, le modèle de Paulson [Pau97a,Pau97b,Pau98], le modèle des *strand spaces* [THG99], la modélisation des protocoles par règles de réécriture [CDL+99, CD99, RT01] ou leur représentations par clauses de Horn [Bla01, CLC03]. Dans ces modèles, l'intrus est décrit par règles d'inférence et l'exécution d'un protocole par une suite des états du protocole, appelée une trace. Les propriétés de sécurité sont normalement exprimée comme un ensemble d'états qui n'est pas accessible.

L'avantage d'une modélisation symbolique n'est pas seulement la possibilité d'obtenir des preuves mathématiques des propriétés, mais aussi le fait que les modèles sous-jacents permettent souvent une automatisation de celles-ci. Des outils spéciaux comme TA4SP [BHKO04] ou le *NRL protocol analyser* [Mea96] ont été développés à cette fin. D'autres approches consistent en l'application des outils généraux, tels que les *model checkers* FDR [Low96] et ASTRAL [DK97] ou des assistants de preuve comme Isabelle [Pau97b].

Finalement, un bon nombre d'outils a été mis en place pour la recherche d'attaque pour certains scénarii d'exécution d'un protocole. Les situations considérées sont normalement les cas où le nombre des participants ou la taille des messages est borné. Des outils qui permettent la vérification dans ces circonstances sont par exemple Athena [Son99], Caspar [Low97a], CAPSL [DM00], Casrul [JRV00] et la plupart de moteurs d'analyse d'AVISPA [ABB+05].

1.6 Plan et objectifs de la thèse

1.6.1 Première partie

Nous avons mentionné qu'une grand diversité des modèles symboliques existe. Il n'est cependant pas clair à quel point les résultats de vérification, obtenus à l'aide de ces modèles, sont comparables. L'objectif de la première partie de cette thèse est de mettre formellement en relation les différents modèles symboliques qui sont utilisés dans les outils de vérification automatique. Ces modèles se distinguent par les algèbres des termes utilisées pour la modélisation des messages. Les algèbres mis à disposition par certains outils sont par fois peu expressives dans le sens où les symboles fonctionnels autorisés représentent seulement une petite partie des primitives cryptographiques dont on peut avoir besoin : les vérificateurs des protocoles cryptographiques sont donc souvent forcés à recourir à des astuces pour coder les primitives cryptographiques. Nous avons par exemple déjà vu qu'on peut utiliser un chiffrement asymétrique à clef privée comme ersatz de la signature.

Nous avons formellement prouvé que certaines astuces courantes de codage sont correctes pour une certaine classe de propriétés. Un protocole modélisé et vérifié dans un modèle peu expressif préserve ces propriétés dans le cas où nous le modélisons et vérifions dans un modèle plus expressif. Si nous prouvons par exemple une propriété dans un modèle où nous avons dû coder la signature par un chiffrement asymétrique à clef privée, alors nous pouvons être sûrs que cette propriété est aussi satisfaite dans un modèle qui nous permet d'exprimer syntaxiquement la signature.

Dans le chapitre 2, nous définissons en détails les modèles symboliques avec lesquels nous travaillons dans cette partie. Le chapitre 3 contient nos preuves de correction. Dans le chapitre 4, nous avons utilisé nos résultats dans une application réelle. Elle consiste en un module écrit pour la plate-forme de vérification AVISPA. À l'aide de ce module, en combinaison avec les résultats de l'article [CKKW06], nous parvenons à transférer certaines propriétés prouvées dans un modèle symbolique vers un modèle calculatoire. Ces propriétés comprennent l'authentification faible et, dans des cas particuliers, aussi l'authentification forte. Ce transfert des propriétés vers un modèle calculatoire est une première dans le domaine de la vérification automatique des protocoles cryptographiques. Finalement, nous avons analysé la bibliothèque de protocoles d'AVISPA pour tester l'efficacité de notre module. Nous terminons ce chapitre avec des conclusions sur la viabilité de notre méthode, ainsi que des propositions d'amélioration.

1.6.2 Deuxième partie

Nous nous somme ensuite tournés vers l'étude des protocoles récursifs. Dans cette classe de protocoles, le nombre de participants ou la taille des messages est non borné. Ces inconnues les font difficiles à vérifier. Jusqu'à ce jour, peu de travaux existent dans ce domaine. Vu qu'un nombre non négligeable de protocoles utilisent la récursivité, l'intérêt d'approfondir l'étude de cette classe de protocoles est évident.

Après un bref aperçu sur les formes de la récursivité trouvée dans des protocoles cryptographiques au chapitre 5, nous définissons, dans le chapitre 6, un modèle symbolique qui nous permet de les décrire formellement. Le chapitre 7 contient l'étude d'un protocole particulier, présenté dans [KAG98]. Nous montrons d'abord qu'à l'aide de notre modèle, nous sommes capables de retrouver une attaque qui a été décrite précédemment contre une des propriétés revendiquées par les auteurs. Nous proposons ensuite des modifications du protocole original dans le but d'obtenir un protocole qui satisfasse cette propriété. Nous prouvons que cela est possible pour une propriété légèrement amendée.

Première partie

Comparaison des modèles symboliques

Chapitre 2

Un système formel pour la spécification des propriétés de sécurité

La modélisation symbolique d'un protocole cryptographique passe, en toute généralité, par une étape où une algèbre de termes est définie. Cette algèbre est utilisée pour représenter les messages qui sont échangés sur le réseau. Étant donné que le domaine de vérification des protocoles cryptographiques était en pleine expansion pendant ces dernières années, il n'est guère étonnant qu'une multitude d'algèbres existent. Elles se distinguent essentiellement par deux aspects :

Premièrement, elles sont, selon le groupe particulier de protocoles visés par la modélisation, plus ou moins riches en symboles fonctionnels. Parmi ces symboles peuvent figurer les représentations des modes de chiffrement, la signature ou encore les fonctions de hachage. Et deuxièmement, deux algèbres, comparables de par les symboles fonctionnels mis à disposition, peuvent se distinguer quand même par leur syntaxe.

Cependant, si les représentations symboliques des primitives cryptographiques et les combinaisons possibles de celles-ci sont multiples, le nombre de primitives pris en compte pour la vérification automatique reste jusqu'à ce jour très limité. Un regard sur des outils existants et leurs modèles sous-jacents affirme cette vérité.

Capacités syntaxiques des outils pour la vérification automatique

Un inventaire des outils de vérification automatique les plus populaires, à savoir ProVerif [Bla01], CASPER [Low97b], Athena [Son99] et AVISPA [ABB+05], montre que les primitives cryptographiques prises en compte sont en effet au nombre de quatre : le chiffrement symétrique, le chiffrement asymétrique, la signature et une fonction de hachage. La figure 2.1 résume cette liste. Un plus (+) indique que l'outil fournit le moyen pour exprimer syntaxiquement la fonctionnalité en question, tandis qu'un minus (−) signifie que la fonctionnalité ne peut pas être exprimée.

En outre, certains outils permettent l'utilisation de fonctions autres que purement cryptographiques, comme, par exemple, les fonctions *xor* ou *exp* dont le comportement correspond au *ou* exclusif ou à l'exponentiation. Une fonction qui est présente dans tous les outils est la fonction de la paire.

Le fait qu'un outil ne possède pas de symbole pour exprimer une primitive cryptographique syntaxiquement n'exclut cependant pas qu'un protocole qui en a besoin puisse être modélisé dans le contexte d'un tel outil. Un manque qui saute aux yeux est, par exemple, l'absence d'une fonction de signature dans tous les outils, à l'exception de ProVerif. Il n'est pour autant pas question que les autres outils renoncent à la modélisations des protocoles qui sont basés sur une primitive

Outil	chiffrement symétrique	chiffrement asymétrique	signature	fonctions de hachage
ProVerif	+	+	+	+
CASPER	+	+	−	+
Athena	−	+	−	−
AVISPA	+	+	−	+

FIG. 2.1 – Capacités des outils différents. Peut être exprimer syntaxiquement (+), ne peut pas être exprimer syntaxiquement(−).

---

de signature. En effet, une astuce couramment employée est la suivante : disposant du chiffrement asymétrique, on note $\mathsf{ek}(a)$ la clef publique et $\mathsf{dk}(a)$ la clef privée d'un agent a. Un terme $\{m\}_{\mathsf{dk}(a)}$ qui représente le chiffrement d'un message m par la clef privée peut alors être considéré comme le message m signé par l'agent a, puisque celui-ci est le seul qui dispose de la clef privée $\mathsf{dk}(a)$. Les autres participants pourraient, en déchiffrant avec la clef publique $\mathsf{ek}(a)$, aisément vérifier la provenance du message $\{m\}_{\mathsf{dk}(a)}$. De même, on peut utiliser le terme $\{hash(m)\}_{\mathsf{dk}(a)}$ où $hash$ est une fonction de hachage comme signature de m. On remarque qu'il existe une différence entre ces deux modélisations de la signature : la première permet à un participant, en déchiffrant avec la clef publique $\mathsf{ek}(a)$, de connaître le message m. Cela n'est pas le cas dans la deuxième modélisation. Ici, un participant doit déjà connaître m, puis calculer $hash(m)$. Finalement, il compare cet hachage avec le terme qu'il obtient en déchiffrant $\{hash(m)\}_{\mathsf{dk}(a)}$ afin de vérifier la provenance du message m. Cependant, rien ne lui permet de connaître m en connaissant la signature $\{hash(m)\}_{\mathsf{dk}(a)}$. Cette différence entre les deux modélisations peut être éliminée, si on remplace le deuxième terme $\{hash(m)\}_{\mathsf{dk}(a)}$ par la paire $\langle m, \{hash(m)\}_{\mathsf{dk}(a)}\rangle$.

L'absence d'une fonction de hachage est un deuxième point faible qu'exposent certains outils comme, par exemple, Athena. Mais on peut aussi remédier à ce manque : si $\mathsf{ek}(h)$ est la clef publique d'un agent h qui ne participe pas au protocole, on peut utiliser le terme $\{m\}_{\mathsf{ek}(h)}$ comme ersatz d'un hachage de m. Ce contournement repose sur le fait qu'aucun participant ne possède la clef de décryptage $\mathsf{dk}(h)$. Le calcul de $\{m\}_{\mathsf{ek}(h)}$ à partir de m revient alors à calculer une valeur de hachage.

Une dernière remarque porte sur l'absence du chiffrement symétrique dont souffrent certains outils, notamment CASPER et Athena. Afin d'exprimer le chiffrement symétrique, le premier outil permet de déclarer une clef quelconque comme étant l'inverse d'elle-même. Si une clef est déclarée comme telle, alors la fonction de chiffrement est symétrique, bien que le symbole fonctionnel utilisé ne se distingue pas du chiffrement asymétrique. Dans d'autres mot, si un participant connaît les termes $\mathsf{ek}(a)$ et $\{m\}_{\mathsf{ek}(a)}$, et la clef $\mathsf{ek}(a)$ a été déclarée inverse d'elle-même, alors il peut déchiffrer pour apprendre le message m. Sinon, il doit connaître la clef $\mathsf{dk}(a)$.

Motivation

Nous avons vu dans le paragraphe précédant que les modèles symboliques, sur lesquels sont basés les outils, proposent des combinaisons de fonctionnalités différentes dans leur syntaxe de spécification. On a vu aussi que, malgré les déficits exposés par certains modèles, ceux-ci peuvent souvent être rattrapés par des astuces de codage. Nous avons pu remarquer que ces codages sont parfois équivalents à la fonction modélisée, mais qu'ils peuvent aussi s'avérer problématiques, dans le sens où ils suppriment ou rajoutent des avantages que peut prendre un intrus éventuel. C'était notamment le cas pour la signature, où l'utilisation du hachage seul, chiffré à clef privée, comme

ersatz n'était pas équivalente, bien qu'un participant qui connaissait le message de départ ait été en mesure d'effectuer la vérification. Il nous semble donc une tâche intéressante de comparer les modèles différents à l'aide d'une base formelle. Cela nous permettra ultérieurement de relier les résultats de vérification automatique. Nous pourrons notamment transférer des propriétés de protocole, vérifiées pour un modèle, vers un autre modèle sans devoir les vérifier à nouveau. Cela nous aidera aussi de justifier les codages usuels, ainsi que mettre en évidence les pièges que peuvent comporter ceux-ci.

Contenu de ce chapitre

Pour traiter les question soulevées dans le paragraphe ci-dessus, nous avons besoin d'un cadre formel que nous développons dans ce chapitre.

Nous avons dans un premier temps besoin des algèbres plus ou moins puissantes. L'algèbre la plus riche inclut les fonctions de chiffrement symétrique et asymétrique, la signature et une fonction de hachage. Cette algèbre a été conçue de façon à ce que le chiffrement et la signature soient représentées dans leurs versions probabilistes. La prise en compte du caractère probabiliste de ces fonctions rapproche la vérification dans un modèle symbolique de la réalité, où les primitives cryptographiques sont des vrais algorithmes probabilistes. Une deuxième algèbre est obtenue en remplaçant les symboles fonctionnels probabilistes par leurs pendants déterministes. Il résulte une algèbre néanmoins riche, mais qui est plus proche de ce qui est modélisé habituellement dans les outils de vérification automatique. Cette algèbre est par exemple apte à prendre en compte les fonctionnalités présentes dans les outils de la figure 2.1. Dans la troisième algèbre, nous avons supprimé la fonction de la signature. Nous devons alors recourir au codage de la signature déjà mentionné précédemment, si nous voulons modéliser cette fonctionnalité. Cette algèbre est aussi puissante que celle utilisée dans les outils CASPER et AVISPA. La quatrième algèbre est obtenue en supprimant aussi la fonction de hachage. Elle est proche de celle utilisée dans l'outil Athena.

Toutes ces algèbres nous servent à spécifier des protocoles cryptographiques. Afin qu'une telle spécification ait une sémantique, nous définissons, dans une deuxième étape, un modèle d'exécution symbolique. Ce modèle, quoiqu'indépendant d'une algèbre particulière, est paramétré par celles-ci. Il nous sert à obtenir des traces d'exécution d'un protocole spécifié dans une algèbre particulière.

Finalement, nous proposons des logiques qui nous permettent de raisonner sur des propriétés des traces d'exécution d'un protocole, et ainsi sur le protocole lui-même. Une logique particulière est aussi, comme le modèle d'exécution, paramétrée par une des algèbres proposées. Nous finissons ce chapitre en donnant des exemples des propriétés de traces pour montrer que nos logiques sont en effet expressives.

Perspective

Le chapitre 3 apporte des réponses sur les questions soulevées dans le paragraphe de la motivation ci-dessus. On y pose notamment la question suivante : étant donnée une algèbre particulière, une propriété prouvée pour un protocole spécifié dans cette algèbre est-elle aussi valide, si le protocole est spécifié en utilisant une autre algèbre ? Comme exemple, nous pouvons encore citer le cas de la signature. La question concrète se pose alors de la façon suivante : si une propriété est vérifiée pour un modèle qui ne connaît pas la signature, alors que peut-on conclure pour un modèle qui contient une fonction de signature ? Cela implique la question, s'il est formellement justifié de coder la fonction de la signature par un chiffrement à clef privée.

Nous verrons que certaines restrictions nous permettent en effet de formuler des propriétés qui peuvent être transférées d'un modèle à l'autre.

2.1 Termes et Algèbres

Les algèbres que nous définissons servent à la fois pour spécifier des protocoles de sécurité et pour représenter des messages concrets échangés sur le réseau. Par conséquent, les définitions des algèbres reflètent cet emploi double :

D'une côté, elles comprennent des variables qui sont utiles afin de représenter les *patrons* des messages. Un patron représente la forme attendue d'un message. Prenons par exemple le rôle d'un protocole, où un participant doit envoyer son nom à un autre participant. Dans une instance d'exécution, le message correspondant serait de la forme a où la constante a représente le nom du participant concret. Il n'est cependant pas pratique de spécifier en générale la forme du message par la constante a. On s'imagine seulement le cas, où un agent b veut exécuter ce rôle. Alors, l'envoi du message a ne correspondrait plus au protocole. Pour cette raison, on utilise des variables. On peut par exemple dire que la variable A représente le participant qui exécute le rôle, c'est-à-dire que le message réellement émis s'adapte à l'agent actuel : dans le premier cas, le message envoyé serait a, tandis que, dans le deuxième, il serait b. On considère donc la variable A comme patron d'un message.

D'autre côté, comme on vient de voir, nous avons besoin d'exprimer les messages réellement envoyés. On ne trouve plus de variables dans ces messages, car elles on été instanciées.

Dans cette section, nous définissons d'abord les termes de base qui consistent en des termes représentant des constantes et variables. Dans la suite, nous présentons les symboles fonctionnels et définissons les algèbres de termes de manière récursive.

2.1.1 Termes de base

Agents, nonces et clefs à court terme

Nous utilisons des constantes afin de représenter les *identités d'agents*. Nous notons cet ensemble ID et des identités particulières sont souvent notées a_1, a_2, ...

Nous notons Nonce l'un ensemble de *nonces*, c'est-à-dire de valeurs qui servent comme messages de base. Elles sont comparables aux données échangées sur le réseaux. L'ensemble des nonces qu'un agent $a \in \mathsf{ID}$ peut créer est donné par $\mathsf{Nonce}(a) = \{n(a, j, s) \mid j, s \in \mathbb{N}\}$. Les nonces d'un agent ont les propriétés suivantes : premièrement, aucun agent ne peut produire un nonce d'un autre agent. Son ensemble de nonces est donc disjoint des ensembles de nonces des autres agents. Deuxièmement, les nonces se renouvellent à chaque session s. À chaque fois que le participant commence à jouer un rôle d'un protocole, les nonces qu'il utilise sont différents des nonces émis auparavant. Finalement, si un agent utilise plusieurs nonces différents dans une sessions, ceux-ci se distinguent par l'entier j. L'ensemble Nonce est donné par $\mathsf{Nonce} = \bigcup_{a \in \mathsf{ID}} \mathsf{Nonce}(a)$.

En ce qui concerne les clefs de chiffrement symétrique, nous devons en distinguer deux types : les *clefs à court terme* et les *clefs à long terme*. Nous expliquerons ultérieurement ce dernier type de clefs. Commençons d'abord avec les clefs symétriques à court terme. L'ensemble de clefs à court terme d'un agent $a \in \mathsf{ID}$ est noté $\mathsf{SKShort}(a) = \{k(a, j, s) \mid j, s \in \mathbb{N}\}$. Ce type de clefs se comporte comme les nonces, c'est-à-dire que chaque agent possède ses propres clefs qui changent avec le numéro de la clef j utilisée dans une session et le numéro de la session s. L'ensemble de clefs symétriques à court terme est donné par $\mathsf{SKShort} = \bigcup_{a \in \mathsf{ID}} \mathsf{SKShort}(a)$. Pour utiliser ce type

de clef symétrique, les agents doivent d'abord se mettre d'accord sur la clef à utiliser pendant une communication.

Étiquettes

Les algorithmes de chiffrement sont en réalité probabilistes, c'est-à-dire que deux chiffrements successifs d'un même message n'aboutissent pas forcément à deux séquences de bits identiques. Pour ce faire, les algorithmes se servent des nombres aléatoires. Au niveau symbolique, nous avons recours aux *étiquettes* ou *labels* qui peuvent être considérées comme équivalents symboliques des nombres aléatoires. Nous montrons d'abord un exemple de l'utilisation des étiquettes, avant de procéder à une définition formelle dans le cadre de nos algèbres.

Exemple 2. Considérons la spécification informelle du protocole suivant :

$$A \to B \quad : \quad \langle \{n_A\}_{\mathsf{ek}(B)}, \{\{n_A\}_{\mathsf{ek}(B)}\}_{\mathsf{ek}(B)} \rangle$$

L'agent A envoie un message qui contient deux occurrence de la forme $\{n_A\}_{\mathsf{ek}(B)}$ où $\mathsf{ek}(B)$ est la clef publique du rôle B. Pendant une implémentation de ce protocole, il est, sous l'hypothèse du chiffrement probabiliste, tout à fait sensé de se demander, si les deux occurrences diffèrent ou si l'une est une copie de l'autre. Nous résolvons cet ambiguïté par l'emploi des étiquettes. Ainsi, nous distinguons les deux protocoles

$$A \to B \quad : \quad \langle \{n_A\}^{l_1}_{\mathsf{ek}(B)}, \{\{n_A\}^{l_1}_{\mathsf{ek}(B)}\}^{l_2}_{\mathsf{ek}(B)} \rangle \qquad (1)$$

et

$$A \to B \quad : \quad \langle \{n_A\}^{l_1}_{\mathsf{ek}(B)}, \{\{n_A\}^{l_2}_{\mathsf{ek}(B)}\}^{l_3}_{\mathsf{ek}(B)} \rangle \qquad (2).$$

Étant donné que, dans le premier cas, la même étiquette l_1 est utilisée pour chiffrer le nonce n_A avec la clef publique de B, nous en déduisons que l'une des occurrences est une copie de l'autre. Dans le deuxième cas, nous utilisons deux étiquettes différentes l_1 et l_2. Cela indique que les occurrences diffèrent, c'est-à-dire que l'algorithme de chiffrement doit être exécuté séparément pour chacune des occurrences. En conclusion, on peut dire que les étiquettes représentent l'aléa utilisé dans le chiffrement probabiliste.

Nous souhaitons tenir compte du fait qu'en réalité, les agents honnêtes et l'adversaire ne peuvent pas utiliser la même source d'aléa. D'autre part, nous ne différencions pas plus la source d'aléa utilisée par les agents honnêtes, même si chaque agent utilise réellement une source différente. Ce n'est pourtant pas une restriction grave de notre modélisation, parce que les agents honnêtes ne cherchent par définition pas à frauder. Aussi suffit-il d'avoir la même source d'aléa pour tous ces participants honnêtes. Nous définissons donc à l'aide de deux symboles fonctionnels $\mathsf{ag}(\_)$ et $\mathsf{adv}(\_)$ deux ensembles d'étiquettes, $\mathsf{Label}_{\mathsf{ag}} = \{\mathsf{ag}(i) \mid i \in \mathbb{N}\}$ et $\mathsf{Label}_{\mathsf{adv}} = \{\mathsf{adv}(i) \mid i \in \mathbb{N}\}$, qui tiendront lieu de ce qui sont en réalité les sources de nombres aléatoires. Si nous voulons parler de l'ensemble d'étiquettes sans distinction particulière, nous notons $\mathsf{Label} = \mathsf{Label}_{\mathsf{ag}} \cup \mathsf{Label}_{\mathsf{adv}}$.

Exemple 3. Pour donner un exemple d'utilisation de notre modélisation symbolique des sources d'aléa, nous regardons les termes $\{n\}^{\mathsf{ag}(1)}_{\mathsf{ek}(a)}$ et $\{n\}^{\mathsf{adv}(1)}_{\mathsf{ek}(a)}$. Dans le premier cas, un agent honnête a chiffré la nonce n avec la clef publique de l'agent a, noté $\mathsf{ek}(a)$. Dans le deuxième cas, c'était

l'adversaire. Les deux termes sont donc bien distincts et, dans notre modélisation, un adversaire ne parviendra jamais à produire un chiffrement semblable à celui d'un agent honnête, car il n'a pas accès à l'ensemble $\mathsf{Label}_{\mathsf{ag}}$. De même, nous pouvons différencier entre $\{n\}^{\mathsf{ag}(1)}_{\mathsf{ek}(a)}$ et $\{n\}^{\mathsf{ag}(2)}_{\mathsf{ek}(a)}$. Ces deux termes indiquent qu'en réalité, on a dû employer deux fois l'algorithme de chiffrement pour obtenir les deux chiffrements du nonce n avec la clef publique de l'agent a. En même temps, les termes ne révèlent pas, s'ils ont été créés par le même agent ou par deux agents différents, puisque tous les agents honnêtes ont accès à la même source d'étiquettes $\mathsf{Label}_{\mathsf{ag}}$.

Nous n'avons jusqu'à présent considéré que le chiffrement asymétrique. Il reste à mentionner que le chiffrement symétrique et la signature se servent de manière analogue des étiquettes.

Variables

Afin de modéliser non seulement les messages émis, mais aussi les patrons de messages qui sont à envoyer ou attendus pendant l'exécution d'un protocole, nous avons aussi besoin de *variables*. Soit $X^{lhs} = X.n \cup X.a \cup X.k \cup X.c \cup X.s \cup X.h \cup X.l$ l'ensemble réunissant respectivement les variables de type nonce, agent, clef symétrique à court terme, message chiffré, message signée, fonction de hachage et étiquette. Nous adoptons aussi les notations $X^{hs} = X^{lhs} \setminus X.l$, $X^{h} = X^{hs} \setminus X.s$ et $X = X^{h} \setminus X.h$ pour les ensembles de variables auxquels nous ôtons successivement les variables d'étiquettes, de signature et de fonction de hachage. Chacun de ces ensembles X^{lhs}, X^{hs}, X^{h} et X correspond à une des algèbres que nous définirons dans la section 2.1.2. Finalement, nous dénotons par $X^{\mathsf{ID}}_{a} = X.a \cup \mathsf{ID}$ l'union des variables et des identités d'agents.

2.1.2 Les algèbres

Avant de nous lancer dans les définitions formelles des algèbres, nous donnons une explication informelle des symboles fonctionnels utilisés. Nous avons déjà vu les symboles $n(\_,\_,\_)$ et $k(\_,\_,\_)$ utilisés pour représenter les nonces et les clefs symétriques à court terme. Les autres symboles fonctionnels utilisés sont les différentes clefs à long terme, le chiffrement asymétrique et symétrique, la signature, la fonction de hachage et la paire. Notons que nous avons deux types de symboles pour les chiffrements et la signature. Le premier type est utilisé pour exprimer le caractère probabiliste de ces primitives cryptographiques. C'est seulement la première algèbre qui utilise ce type de symboles. Ce choix a aussi permis d'établir un lien entre ce modèle symbolique et les modèles cryptographiques [CW05]. Le chapitre 4 en présente une application. Le deuxième type de symboles pour les chiffrements et la signature est utilisé dans toutes les autres algèbres.

Les clefs à long terme

Nous utilisons les quatre symboles fonctionnels $\mathsf{ek}(\_)$, $\mathsf{dk}(\_)$, $\mathsf{sk}(\_)$ et $\mathsf{vk}(\_)$ pour représenter respectivement les clefs de chiffrement, de déchiffrement, de signature et de vérification. Ces symboles prennent comme argument une identité $a \in \mathsf{ID}$. Ainsi, $\mathsf{ek}(a)$ et $\mathsf{dk}(a)$ désignent la clef publique, respectivement privée de chiffrement asymétrique de l'agent a. De même, les termes $\mathsf{sk}(a)$ et $\mathsf{vk}(a)$ représentent ses clefs de signature et de vérification. Formellement, nous définissons les quatre ensembles disjoints EKey, DKey, SKey et VKey par $\mathsf{EKey} = \{\mathsf{ek}(a) \mid a \in \mathsf{ID}\}$, *etc.*

Quant au chiffrement symétrique, nous définissons les *clefs à long terme* à l'aide du symbole fonctionnel $\mathsf{symk}(\_,\_)$. Si un agent $a \in \mathsf{ID}$ veut envoyer un message symétriquement chiffré à un agent $b \in \mathsf{ID}$, alors il pourra utiliser la clef $\mathsf{symk}(a,b)$ qui est aussi connue par l'agent b. De même, il peut déchiffrer un message chiffré symétriquement avec la clef $\mathsf{symk}(b,a)$. L'ensemble

des clefs symétriques qui sont à la disposition d'un agent $a \in \mathsf{ID}$ est donné par $\mathsf{SKLong}(a) = \{\mathsf{symk}(a,b), \mathsf{symk}(b,a) \mid b \in \mathsf{ID}\}$.

Les autres symboles fonctionnels

Le *chiffrement asymétrique* est représenté par les deux symboles fonctionnels $\{\_\}\_$ et $\{\_\}\_$. La première algèbre que nous allons définir utilise le symbole ternaire. Il prend comme argument le message m à chiffrer, une clef de chiffrement, par exemple $\mathsf{ek}(a)$, et une étiquette qui représente l'aléa qui est utilisé pendant le chiffrement. Ainsi, $\{m\}^{\mathsf{ag}(1)}_{\mathsf{ek}(a)}$ est un terme qui représente un message asymétriquement chiffré. Toutes les autres algèbres utilisent le symbole binaire, qui ne permet pas d'exprimer le caractère probabiliste du chiffrement asymétrique. Par conséquent, on omet l'étiquette comme argument. Un terme qui représente un message chiffré dans ces dernières algèbres est par exemple $\{m\}_{\mathsf{ek}(a)}$.

En ce qui concerne le *chiffrement symétrique*, nous utilisons les symboles $\{\!|\_|\!\}\_$ et $\{\!|\_|\!\}\_$. La raisons pour laquelle nous avons un symbole ternaire et un symbole binaire est la même que pour le chiffrement asymétrique. Nous utilisons le premier symbole dans la première algèbre et le deuxième dans toutes les autres algèbres. Dans la position de la clef, nous autorisons non seulement les deux types de clefs symétriques que nous avons définis auparavant, mais également des termes quelconques. Une clef qui consiste en un terme quelconque est appelée une *clef composée*.

Quant à la *signature*, elle est exprimée par les symboles $[\_]\_$ et $[\_]\_$. Ici aussi, nous avons deux symboles que nous utilisons suivant, si nous voulons exprimer la signature comme fonction probabiliste ou déterministe. C'est donc dans la première algèbre que nous utilisons le premier symbole, tandis que le deuxième est réservé aux autres algèbres.

La fonction de *hachage* est représentée par le symbole fonctionnel $hash(\_)$. Nous notons $hash(m)$ le haché de m. En réalité, le haché d'une donnée m est une séquence de bits courte, calculée de façon à ce qu'il soit difficile à trouver une autre donnée m' telle que $hash(m') = hash(m)$.

Finalement, nous considérons comme fonction de concaténations la *paire* $\langle \_, \_ \rangle$. Afin d'exprimer la concaténations de plusieurs éléments, on doit utiliser cette fonction dans des éventuelles imbrications. On pourrait par exemple noter $\langle a_1, \langle a_2, a_3 \rangle\rangle$ pour a_1, a_2, a_3 ou encore $\langle\langle a_1, a_2\rangle, a_3\rangle$.

Une convention de notation

Nous pouvons maintenant définir les algèbres conçues pour spécifier des protocoles, ainsi que pour représenter des messages échangés. Nous avons choisi comme convention, afin de distinguer les algèbres, de les dénoter par T^{lhs}, T^{hs}, T^{h} et T où les exposants lhs, hs et h indiquent, si elles mettent à disposition le chiffrement probabiliste (l pour labels), la signature (s) et une fonction de hachage (h). L'algèbre la plus simple T ne porte pas d'exposant.

L'algèbre T^{lhs}

C'est l'algèbre la plus riche de notre système. Elle contient une fonction de hachage et, quant aux chiffrement symétrique et asymétrique, ainsi qu'à la signature, ces fonctions s'y trouvent dans leurs versions probabilistes. Cette algèbre est formellement définie par

$$
\begin{array}{lll}
\mathcal{L} & ::= & X.l \mid \mathsf{ag}(i) \mid \mathsf{adv}(i) \\
T^{lhs} & ::= & X^{hs} \mid a \mid n(a,j,s) \mid \mathsf{ek}(X_a^{\mathsf{ID}}) \mid \mathsf{dk}(X_a^{\mathsf{ID}}) \mid \mathsf{sk}(X_a^{\mathsf{ID}}) \mid \mathsf{vk}(X_a^{\mathsf{ID}}) \mid \mathsf{symk}(X_a^{\mathsf{ID}}, X_a^{\mathsf{ID}}) \mid k(a,j,s) \\
 & & \mid \langle T^{lhs}, T^{lhs}\rangle \mid \{\!|T^{lhs}|\!\}^{\mathcal{L}}_{T^{lhs}} \mid \{T^{lhs}\}^{\mathcal{L}}_{\mathsf{ek}(X_a^{\mathsf{ID}})} \mid \{T^{lhs}\}^{\mathcal{L}}_{\mathsf{dk}(X_a^{\mathsf{ID}})} \mid [T^{lhs}]^{\mathcal{L}}_{\mathsf{sk}(X_a^{\mathsf{ID}})} \mid hash(T^{lhs})
\end{array}
$$

où $i, j, s \in \mathbb{N}$ sont des entiers, $a \in \mathsf{ID}$ est une identité d'agent et $X^{hs} = X^{lhs} \setminus X.l$ est l'ensemble de variables sans les variables d'étiquettes. Rappelons encore que $X_a^{\mathsf{ID}} = X.a \cup \mathsf{ID}$ est l'ensemble des variables et des identités d'agents.

Avec les symboles fonctionnels probabilistes, cette algèbre offre bien plus de pouvoir de spécification que celui qui est habituellement pris en compte par les outils de vérification automatique. Elle est par contre utile, si on veut transférer un résultat de vérification qui a été établi dans un modèle symbolique vers un modèle computationnel. Nous verrons dans le chapitre 4 un exemple pour un tel transfert.

L'algèbre T^{hs}

L'algèbre T^{hs} est définie de manière similaire à T^{lhs} à l'exception du chiffrement et de la signature dont le caractère probabiliste n'est plus pris en compte. Par conséquent, leurs symboles fonctionnels ont été remplacés par leur pendants déterministes :

$$\begin{aligned} T^{hs} \quad ::= \quad & X^{hs} \mid a \mid n(a,j,s) \mid \mathsf{ek}(X_a^{\mathsf{ID}}) \mid \mathsf{dk}(X_a^{\mathsf{ID}}) \mid \mathsf{sk}(X_a^{\mathsf{ID}}) \mid \mathsf{vk}(X_a^{\mathsf{ID}}) \mid \mathsf{symk}(X_a^{\mathsf{ID}}, X_a^{\mathsf{ID}}) \mid k(a,j,s) \\ & \mid \langle T^{hs}, T^{hs} \rangle \mid \{\!|T^{hs}|\!\}_{T^{hs}} \mid \{T^{hs}\}_{\mathsf{ek}(X_a^{\mathsf{ID}})} \mid \{T^{hs}\}_{\mathsf{dk}(X_a^{\mathsf{ID}})} \mid [T^{hs}]_{\mathsf{sk}(X_a^{\mathsf{ID}})} \mid hash(T^{hs}) \end{aligned}$$

où $j, s \in \mathbb{N}$ sont des entiers, $a \in \mathsf{ID}$ est une identité d'agent et $X^{hs} = X^{lhs} \setminus X.l$ est l'ensemble de variables sans les variables d'étiquettes. Cette algèbre correspond à l'algèbre utilisé dans l'outil ProVerif [Bla01].

L'algèbre T^h

Dans l'algèbre T^h, le symbole fonctionnel de signature est supprimé. Les clefs de signature et de vérification sont donc, elles aussi, ôtées de l'algèbre. Cette algèbre est formellement définie par :

$$\begin{aligned} T^{h} \quad ::= \quad & X^{h} \mid a \mid n(a,j,s) \mid \mathsf{ek}(X_a^{\mathsf{ID}}) \mid \mathsf{dk}(X_a^{\mathsf{ID}}) \mid \mathsf{symk}(X_a^{\mathsf{ID}}, X_a^{\mathsf{ID}}) \mid k(a,j,s) \\ & \mid \langle T^{h}, T^{h} \rangle \mid \{\!|T^{h}|\!\}_{T^{h}} \mid \{T^{h}\}_{\mathsf{ek}(X_a^{\mathsf{ID}})} \mid \{T^{h}\}_{\mathsf{dk}(X_a^{\mathsf{ID}})} \mid hash(T^{h}) \end{aligned}$$

où $j, s \in \mathbb{N}$ sont des entiers, $a \in \mathsf{ID}$ est une identité d'agent et $X^{h} = X^{lhs} \setminus (X.l \cup X.s)$ est l'ensemble des variables sans celles d'étiquettes et de signature. Les outils CASPER [Low97b] et AVISPA [ABB+05] utilisent une algèbre similaire à T^h.

L'algèbre T

L'algèbre T ne contient plus que le chiffrement et la fonction de paire. La fonction de hachage n'y figure plus. Cette algèbre est définie par

$$\begin{aligned} T \quad ::= \quad & X \mid a \mid n(a,j,s) \mid \mathsf{ek}(X_a^{\mathsf{ID}}) \mid \mathsf{dk}(X_a^{\mathsf{ID}}) \mid \mathsf{symk}(X_a^{\mathsf{ID}}, X_a^{\mathsf{ID}}) \mid k(a,j,s) \\ & \mid \langle T, T \rangle \mid \{\!|T|\!\}_{T} \mid \{T\}_{\mathsf{ek}(X_a^{\mathsf{ID}})} \mid \{T\}_{\mathsf{dk}(X_a^{\mathsf{ID}})} \end{aligned}$$

où $j, s \in \mathbb{N}$ sont des entiers, $a \in \mathsf{ID}$ est une identité d'agent et $X = X^{lhs} \setminus (X.l \cup X.s \cup X.h)$. Cette algèbre correspond aux primitives cryptographiques de l'outil Athena [Son99], mais elle contient de plus le chiffrement symétrique comme symbole fonctionnel distinct.

2.1.3 Notions complémentaires des algèbres de termes

Convention de notation

Pour ne pas répéter les définitions et faits qui sont valables pour toutes les algèbres, nous adoptons la convention de notation suivante : l'étoile $\star$ représente un élément de l'ensemble

des exposants $\{lhs, hs, h, \_\}$ où $\_$ note la chaîne de caractère vide. Ainsi, $T^\star$ représente un élément de l'ensemble $\{T^{lhs}, T^{hs}, T^h, T\}$ et $X^\star$ l'élément correspondant de $\{X^{lhs}, X^{hs}, X^h, X\}$. Nous étendons cette convention de notation au fur et à mesure, lorsque nous présentons des nouvelles notions.

Notions des algèbres de termes

Nous définissons la notion de sous-termes d'un terme. Pour ce faire nous avons d'abord besoin d'expliquer ce que nous entendons par l'ensemble des positions d'un terme : étant donnée un symbole fonctionnel f d'arité k, nous disons que, dans le terme $f(t_1, t_2, ..., t_k)$, le terme t_1 occupe la première position, le terme t_2 la deuxième et ainsi de suite. Nous donnons une définition formelle de la notion de position.

Définition 1. Soit $t \in T^\star$ un terme. L'ensemble $\mathcal{P}os(t) \subseteq \mathbb{N}^*$ des *positions* de t est défini par
- $\mathcal{P}os(t) = \{\varepsilon\}$ si t est une variable ou une constante et
- $\mathcal{P}os(t) = \{\varepsilon\} \cup \bigcup_{1 \leq i \leq n} \{i.p \mid p \in \mathcal{P}os(t_i)\}$, si $t = f(t_1, ..., t_n)$

où ε est le mot vide et f est un symbole de fonction de l'algèbre $T^\star$.

Exemple 4. Soit $t = \langle a, \langle \langle a, b \rangle, b \rangle \rangle \in T^\star$ où $a, b \in \mathsf{ID}$. Alors, on a $\mathcal{P}os(t) = \{\varepsilon, 1, 2, 2.1, 2.2, 2.1.1, 2.1.2\}$.

Ensuite, on définit récursivement le sous-terme t' d'un terme t à la position p.

Définition 2. Soit $p \in \mathcal{P}os(t)$ une position d'un terme $t \in T^\star$. Le *sous-terme* $t' = t|_p$ est défini par
- $t|_p = t$, si $p = \varepsilon$ et
- $t|_{i.p} = t_i|_p$, si $t = f(t_1, ..., t_n)$ et $1 \leq i \leq n$, sinon $t|_{i.p}$ n'est pas défini.

Exemple 5. L'ensemble de sous-termes de $t = \langle a, \langle \langle a, b \rangle, b \rangle \rangle \in T^\star$ où $a, b \in \mathsf{ID}$ est $\{\langle a, \langle \langle a, b \rangle, b \rangle \rangle, a, \langle \langle a, b \rangle, b \rangle, \langle a, b \rangle, b\}$. On remarque qu'un terme est un sous-terme de soi-même et qu'il peut contenir plusieurs sous-termes égaux.

Les substitutions sont des fonctions utilisées afin de remplacer les occurrences des variable dans un terme par des termes. Leur domaine de définition comprend les termes et les variables de l'algèbre.

Définition 3. Une *substitution* σ est une fonction de X^{lhs} dans $T^{lhs} \cup \mathcal{L}$. Le *domaine* d'une substitution est fini. Il consiste en l'ensemble de variables x, telle que $\sigma(x) \neq x$. Une substitution $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ est un ensemble de paires de variables de X^{lhs} et termes de $T^{lhs} \cup \mathcal{L}$. Elle est définie par $\sigma(x) = x$ pour toute variable $x \notin \{x_1, \ldots, x_n\}$ et par $\sigma(x_i) = t_i$ pour tout indice $1 \leq i \leq n$. Nous disons qu'une substitution est *clos*, si aucun des termes t_i ne contient une variable.

Nous étendons le domaine de définition d'une substitution σ sur les termes dans $T^{lhs} \cup \mathcal{L}$: pour tous les symboles fonctionnels f d'arité k qui apparaissent dans $T^{lhs} \cup \mathcal{L}$ et tous les termes $u_1, \ldots, u_k \in T^{lhs} \cup \mathcal{L}$ nous définissons $\sigma(f(u_1, \ldots, u_k)) = f(\sigma(u_1), \ldots, \sigma(u_k))$. Pour une constante $u \in T^{lhs} \cup \mathcal{L}$, nous posons $\sigma(u) = u$.

Pour un terme $m \in T^{lhs} \cup \mathcal{L}$, nous dénotons $\sigma(m)$ aussi par $m\sigma$. Nous notons $\mathbb{S}$ l'ensemble de substitutions.

À l'aide des substitutions, on peut unifier des termes. Informellement, cela signifie qu'on remplace les occurrences des variables par des termes de façon à ce que les termes de départ deviennent égaux.

Définition 4. Soient $t_1, t_2 \in T^\star$ deux termes. On dit qu'ils sont *unifiables*, s'il existe une substitution σ, telle que $\sigma(t_1) = \sigma(t_2)$. On dit que σ est l'*unificateur le plus général*, si, pour tout autre substitution θ, telle que $\theta(t_1) = \theta(t_2)$, il existe une substitution θ', telle que $\theta = \sigma\theta'$.

Exemple 6. Soient $t_1 = \langle x, a\rangle$, $t_2 = \langle b, a\rangle$, $t_3 = x$ trois termes. t_1 et t_2 sont unifiable avec l'unificateur le plus général $\sigma = [x \leftarrow b]$, ainsi que t_2 et t_3, avec l'unificateur le plus général $\sigma' = [x \leftarrow \langle b, a\rangle]$. Les termes t_1 et t_3 ne sont pas unifiables.

Comme nous l'avons déjà mentionné antérieurement, nous utilisons les termes contenant des variables pour spécifier les patrons des messages attendus et envoyés pendant l'exécution d'un protocole. Les termes qui représentent les messages réellement émis sont complètement instanciés et ne contiennent pas de variables. Nous appelons un tel terme *clos*.

Définition 5. Si un terme ne contient pas de variables, il est *clos*.

2.2 Systèmes de déduction : le pouvoir de l'intrus

Admettons maintenant qu'un intrus connaît l'ensemble $S \subseteq T^\star$ de termes clos. Quelles informations peut-il en tirer ? Nous voulons, par exemple, qu'il puisse former une paire $\langle m_1, m_2\rangle$ à partir de deux termes $m_1, m_2 \in S$. Ou encore, que, si $m \in S$ est un terme quelconque et $\mathsf{ek}(a)$ la clef publique d'un agent a, l'adversaire doit pouvoir construire le terme $\{m\}_{\mathsf{ek}(a)}$ qui correspond au fait qu'il a le pouvoir de chiffrer m par $\mathsf{ek}(a)$. On le veut aussi capable de réutiliser les termes construits pour en construire d'autres. Pour définir le pouvoir d'un intrus formellement, nous avons recours à des systèmes de déduction. Ceux-ci consistent en des ensembles de règles d'inférence et engendrent une relation entre l'ensemble S et un terme $m \in T^\star$, notée par $S \vdash^\star m$. La signification de cette relation est la suivante : si l'intrus a la connaissance S, il peut en déduire m. Les systèmes de déduction ayant été pour la première fois utilisés par Dolev et Yao [DY81], les règles d'inférence sont aussi appelées les *opérations Dolev-Yao*.

Prenons l'exemple de l'algèbre T^{hs}. Le système de déduction se trouve dans la figure 2.3. Il induit la relation $\vdash^{hs}$ inductivement. Nous distinguons deux types de règles : les règles de base et les règles générales. Ces dernières se divisent en règles de composition et règles de décomposition.

Les règles de base que nous avons étiquetées par *Connaissance initiale* spécifient la connaissance de base de l'intrus. Celle-ci se constitue en les termes déjà présents dans l'ensemble S (la règle $\mathcal{RHS}_1^{init}$), ainsi que les connaissances publiques comme les noms d'agents, leurs clefs publiques de chiffrement et de vérification. De tels termes sont donc toujours déductibles de S (la règle $\mathcal{RHS}_2^{init}$).

Les règles de *composition* ont, en toute généralité, la forme

$$\frac{S \vdash m_1 \quad \ldots \quad S \vdash m_n}{S \vdash f(m_1, \ldots, m_n)}\,.$$

Elles sont à lire de la manière suivante : si on peut déduire les termes $m_1, \ldots, m_n$ de S, alors on peut déduire aussi le terme $f(m_1, \ldots, m_n)$ où f est un symbole fonctionnel d'arité n de l'algèbre considérée. Dans le système de déduction donné dans la figure 2.3, les règles $\mathcal{RHS}_{comp}^{pair}$, $\mathcal{RHS}_c^{sym}$, $\mathcal{RHS}_c^{asym}$, $\mathcal{RHS}_{cinv}^{asym}$, $\mathcal{RHS}_{sig}^{sign}$ et $\mathcal{RHS}^{hash}$ sont des règles de composition.

Toutes les autres règles sont des règles de *décomposition*. Prenons pour exemple concret la règle de déchiffrement à clef symétrique. Supposons que l'intrus a réussi à déduire les termes $\{\!|m|\!\}_k$ et k à partir de S. En appliquant cette règle, il peut alors déduire m de S. Il est donc naturel d'appeler cette règle une décomposition, car le terme obtenu m est un sous-termes d'un des termes utilisé pour la déduction, à savoir du terme $\{\!|m|\!\}_k$.

$\mathcal{RLHS}_1^{init}$	$\dfrac{}{S \vdash^{lhs} m}\ m \in S$	$\mathcal{RLHS}_2^{init}$	$\dfrac{}{S \vdash^{lhs} a, \mathsf{ek}(a), \mathsf{vk}(a)}\ a \in \mathsf{ID}$	Connaissance initiale
$\mathcal{RLHS}_{comp}^{pair}$	$\dfrac{S \vdash^{lhs} m_1 \quad S \vdash^{lhs} m_2}{S \vdash^{lhs} \langle m_1, m_2 \rangle}$	$\mathcal{RLHS}_{dec}^{pair}$	$\dfrac{S \vdash^{lhs} \langle m_1, m_2 \rangle}{S \vdash^{lhs} m_i}\ i \in \{1,2\}$	Composition et décomposition des paires
$\mathcal{RLHS}_c^{sym}$	$\dfrac{S \vdash^{lhs} k \quad S \vdash^{lhs} m}{S \vdash^{lhs} \{\!\lvert m \rvert\!\}_k^{\mathsf{adv}(i)}}$	$\mathcal{RLHS}_d^{sym}$	$\dfrac{S \vdash^{lhs} \{\!\lvert m \rvert\!\}_k^l \quad S \vdash^{lhs} k}{S \vdash^{lhs} m}$	Chiffrement et déchiffrement symétrique
$\mathcal{RLHS}_c^{asym}$	$\dfrac{S \vdash^{lhs} \mathsf{ek}(a) \quad S \vdash^{lhs} m}{S \vdash^{lhs} \{m\}_{\mathsf{ek}(a)}^{\mathsf{adv}(i)}}\ i \in \mathbb{N}$	$\mathcal{RLHS}_d^{asym}$	$\dfrac{S \vdash^{lhs} \{m\}_{\mathsf{ek}(a)}^l \quad S \vdash^{lhs} \mathsf{dk}(a)}{S \vdash^{lhs} m}$	Chiffrement et déchiffrement asymétrique
$\mathcal{RLHS}_{cinv}^{asym}$	$\dfrac{S \vdash^{h} \mathsf{dk}(a) \quad S \vdash^{h} m}{S \vdash^{h} \{m\}_{\mathsf{dk}(a)}}$	$\mathcal{RLHS}_{dinv}^{asym}$	$\dfrac{S \vdash^{h} \{m\}_{\mathsf{dk}(a)} \quad S \vdash^{h} \mathsf{ek}(a)}{S \vdash^{h} m}$	Chiffrement asymétrique avec clefs inversées
$\mathcal{RLHS}_{sig}^{sign}$	$\dfrac{S \vdash^{lhs} \mathsf{sk}(a) \quad S \vdash^{lhs} m}{S \vdash^{lhs} [m]_{\mathsf{sk}(a)}^{\mathsf{adv}(i)}}\ i \in \mathbb{N}$	$\mathcal{RLHS}_{ded}^{sign}$	$\dfrac{S \vdash^{lhs} [m]_{\mathsf{sk}(a)}^l}{S \vdash^{lhs} m}$	Signature
$\mathcal{RLHS}^{hash}$	$\dfrac{S \vdash^{lhs} m}{S \vdash^{lhs} hash(m)}$			Fonction de hachage

FIG. 2.2 – Règles de déduction pour l'algèbre T^{lhs}.

$\mathcal{RHS}_1^{init}$	$\dfrac{}{S \vdash^{hs} m}\ m \in S$	$\mathcal{RHS}_2^{init}$	$\dfrac{}{S \vdash^{hs} a, \mathsf{ek}(a), \mathsf{vk}(a)}\ a \in \mathsf{ID}$	Connaissance initiale
$\mathcal{RHS}_{comp}^{pair}$	$\dfrac{S \vdash^{hs} m_1 \quad S \vdash^{hs} m_2}{S \vdash^{hs} \langle m_1, m_2 \rangle}$	$\mathcal{RHS}_{dec}^{pair}$	$\dfrac{S \vdash^{hs} \langle m_1, m_2 \rangle}{S \vdash^{hs} m_i}\ i \in \{1,2\}$	Composition et décomposition des paires
$\mathcal{RHS}_c^{sym}$	$\dfrac{S \vdash^{hs} k \quad S \vdash^{hs} m}{S \vdash^{hs} \{\!\|m\|\!\}_k}$	$\mathcal{RHS}_d^{sym}$	$\dfrac{S \vdash^{hs} \{\!\|m\|\!\}_k \quad S \vdash^{hs} k}{S \vdash^{hs} m}$	Chiffrement et déchiffrement symétrique
$\mathcal{RHS}_c^{asym}$	$\dfrac{S \vdash^{hs} \mathsf{ek}(a) \quad S \vdash^{hs} m}{S \vdash^{hs} \{m\}_{\mathsf{ek}(a)}}$	$\mathcal{RHS}_d^{asym}$	$\dfrac{S \vdash^{hs} \{m\}_{\mathsf{ek}(a)} \quad S \vdash^{hs} \mathsf{dk}(a)}{S \vdash^{hs} m}$	Chiffrement et déchiffrement asymétrique
$\mathcal{RHS}_{cinv}^{asym}$	$\dfrac{S \vdash^{h} \mathsf{dk}(a) \quad S \vdash^{h} m}{S \vdash^{h} \{m\}_{\mathsf{dk}(a)}}$	$\mathcal{RHS}_{dinv}^{asym}$	$\dfrac{S \vdash^{h} \{m\}_{\mathsf{dk}(a)} \quad S \vdash^{h} \mathsf{ek}(a)}{S \vdash^{h} m}$	Chiffrement asymétrique avec clefs inversées
$\mathcal{RHS}_{sig}^{sign}$	$\dfrac{S \vdash^{hs} \mathsf{sk}(a) \quad S \vdash^{hs} m}{S \vdash^{hs} [m]_{\mathsf{sk}(a)}}$	$\mathcal{RHS}_{ded}^{sign}$	$\dfrac{S \vdash^{hs} [m]_{\mathsf{sk}(a)}}{S \vdash^{hs} m}$	Signature
$\mathcal{RHS}^{hash}$	$\dfrac{S \vdash^{hs} m}{S \vdash^{hs} hash(m)}$			Fonction de hachage

FIG. 2.3 – Règles de déduction pour l'algèbre T^{hs}.

$\mathcal{RH}_1^{init}$	$\dfrac{}{S \vdash^h m}\ m \in S$	$\mathcal{RH}_2^{init}$	$\dfrac{}{S \vdash^h a, \mathsf{ek}(a)}\ a \in \mathsf{ID}$	Connaissance initiale
$\mathcal{RH}_{comp}^{pair}$	$\dfrac{S \vdash^h m_1 \quad S \vdash^h m_2}{S \vdash^h \langle m_1, m_2 \rangle}$	$\mathcal{RH}_{dec}^{pair}$	$\dfrac{S \vdash^h \langle m_1, m_2 \rangle}{S \vdash^h m_i}\ i \in \{1,2\}$	Composition et décomposition des paires
$\mathcal{RH}_c^{sym}$	$\dfrac{S \vdash^h k \quad S \vdash^h m}{S \vdash^h \{\!\|m\|\!\}_k}$	$\mathcal{RH}_d^{sym}$	$\dfrac{S \vdash^h \{\!\|m\|\!\}_k \quad S \vdash^h k}{S \vdash^h m}$	Chiffrement et déchiffrement symétrique
$\mathcal{RH}_c^{asym}$	$\dfrac{S \vdash^h \mathsf{ek}(a) \quad S \vdash^h m}{S \vdash^h \{m\}_{\mathsf{ek}(a)}}$	$\mathcal{RH}_d^{asym}$	$\dfrac{S \vdash^h \{m\}_{\mathsf{ek}(a)} \quad S \vdash^h \mathsf{dk}(a)}{S \vdash^h m}$	Chiffrement et déchiffrement asymétrique
$\mathcal{RH}_{cinv}^{asym}$	$\dfrac{S \vdash^h \mathsf{dk}(a) \quad S \vdash^h m}{S \vdash^h \{m\}_{\mathsf{dk}(a)}}$	$\mathcal{RH}_{dinv}^{asym}$	$\dfrac{S \vdash^h \{m\}_{\mathsf{dk}(a)} \quad S \vdash^h \mathsf{ek}(a)}{S \vdash^h m}$	Chiffrement asymétrique avec clefs inversées
$\mathcal{RH}^{hash}$	$\dfrac{S \vdash^h m}{S \vdash^h hash(m)}$			Fonction de hachage

FIG. 2.4 – Règles de déduction pour l'algèbre T^h.

$\mathcal{R}_1^{init}$	$\dfrac{}{S \vdash m}\ m \in S$	$\mathcal{R}_2^{init}$	$\dfrac{}{S \vdash a, \mathsf{ek}(a)}\ a \in \mathsf{ID}$	Connaissance initiale
$\mathcal{R}_{comp}^{pair}$	$\dfrac{S \vdash m_1 \quad S \vdash m_2}{S \vdash \langle m_1, m_2 \rangle}$	$\mathcal{R}_{dec}^{pair}$	$\dfrac{S \vdash \langle m_1, m_2 \rangle}{S \vdash m_i}\ i \in \{1,2\}$	Composition et décomposition des paires
$\mathcal{R}_c^{sym}$	$\dfrac{S \vdash k \quad S \vdash m}{S \vdash \{\!\|m\|\!\}_k}$	$\mathcal{R}_d^{sym}$	$\dfrac{S \vdash \{\!\|m\|\!\}_k \quad S \vdash k}{S \vdash m}$	Chiffrement et déchiffrement symétrique
$\mathcal{R}_c^{asym}$	$\dfrac{S \vdash \mathsf{ek}(a) \quad S \vdash m}{S \vdash \{m\}_{\mathsf{ek}(a)}}$	$\mathcal{R}_d^{asym}$	$\dfrac{S \vdash \{m\}_{\mathsf{ek}(a)} \quad S \vdash \mathsf{dk}(a)}{S \vdash m}$	Chiffrement et déchiffrement asymétrique
$\mathcal{R}_{cinv}^{asym}$	$\dfrac{S \vdash \mathsf{dk}(a) \quad S \vdash m}{S \vdash \{m\}_{\mathsf{dk}(a)}}$	$\mathcal{R}_{dinv}^{asym}$	$\dfrac{S \vdash \{m\}_{\mathsf{dk}(a)} \quad S \vdash \mathsf{ek}(a)}{S \vdash m}$	Chiffrement asymétrique avec clefs inversées

FIG. 2.5 – Règles de déduction pour l'algèbre T.

$$\dfrac{\dfrac{\dfrac{S \vdash^{hs} k(a,1,1) \quad S \vdash^{hs} \{\!|\langle a,b\rangle|\!\}_{k(a,1,1)}}{S \vdash^{hs} \langle a,b\rangle}}{S \vdash^{hs} a} \qquad \dfrac{S \vdash^{hs} [c]_{\mathsf{sk}(a)}}{S \vdash^{hs} c}}{\dfrac{S \vdash^{hs} \langle a,c\rangle}{S \vdash^{hs} \{\langle a,c\rangle\}_{\mathsf{ek}(b)}} \qquad S \vdash^{hs} \mathsf{ek}(b)}$$

FIG. 2.6 – Exemple d'un arbre de déduction pour l'algèbre T^{hs}.

À titre d'exemple, nous présentons un arbre de déduction dans la figure 2.6. Nous avons choisi $S = \{k(a,1,1), \{\!|\langle a,b\rangle|\!\}_{k(a,1,1)}, [c]_{\mathsf{sk}(a)}\}$ et comme ensemble d'identités d'agents $\mathsf{ID} = \{a,b,c\}$. Dans cet arbre, on peut trouver tous les types de règles de déduction que nous venons de voir.

Les système de déduction pour l'algèbre T^{lhs}

Ce système de déduction contient, à côté des règles de base, des règles de la construction et la déconstruction de paires, du chiffrement et déchiffrement symétrique et asymétrique, de la signature et de la fonction de hachage.

Un aspect particulier de la modélisation de la signature est que nous avons choisi pour convention qu'un terme de signature, par exemple $[m]^{\mathsf{ag}(1)}_{\mathsf{sk}(a)}$, représente à la fois le message et la signature du message. Cela se reflète dans la règle $\mathcal{RLHS}^{sign}_{ded}$. Si l'intrus connaît le terme $[m]^{\mathsf{ag}(1)}_{\mathsf{sk}(a)}$, alors il pourra en déduire le message m. Nous considérons donc que ce terme représente le contenu m en clair, c'est-à-dire non chiffré, et sa signature que, en effet, nous ne représentons pas de façon à ce qu'on puisse la dissocier du terme m.

Cette conception implique l'exclusion de certains scénarii dans notre modélisation. Par exemple, on peut s'imaginer qu'un agent a possède le message m' et un autre agent une signature $[m]'_{\mathsf{sk}(b)}$ qui ne permet pas de récupérer le message m. Que l'agent a peut-il déduire voyant cette signature ? D'une côté, il peut vérifier, si cette signature appartient au message m' qu'il possède. Si oui, il connaît m, puisqu'il connaît m'. Sinon, il ne connaît toujours pas m. Notre concept de signature n'admet cependant pas un tel scénario, car l'agent aurait pu récupérer m, tout en ne connaissant que la signature. Ce scénario étant très artificiel, son impact en pratique n'est certainement pas important.

Si on supprimait la règle $\mathcal{RLHS}^{sign}_{ded}$ de ce système de déduction, alors un terme $[m]^{\mathsf{ag}(1)}_{\mathsf{sk}(a)}$ ne représenterait rien de plus que la signature elle-même. Nous pensons que cette suppression ne porterait pas atteinte aux résultats de correspondance que nous présentons dans le chapitre 3. Le cas échéant, on pourrait toute fois simuler le comportement de la signature, tel comme nous l'avons choisi, en remplaçant toute les signatures par la paire $\langle m, [m]^{\mathsf{ag}(1)}_{\mathsf{sk}(a)}\rangle$. Maintenant, le terme m est toujours accessible grâce à la règle de décomposition d'une paire. Si nous avons quand même choisi de garder cette règle, c'est, parce qu'ainsi l'algèbre T^{lhs}, restreinte au chiffrement asymétrique, la signature et la paire, coïncide avec celle de [CW05], ce qui nous permettra d'utiliser le résultat de correspondance entre modèle symbolique et modèle computationnel présenté dans cet article.

Les système de déduction pour l'algèbre T^{hs}

Ce système de déduction est donné par la figure 2.3. Étant donné que l'algèbre T^{hs} est définie de manière similaire à l'algèbre T^{lhs}, ce système ressemble beaucoup au système précédent. Nous y trouvons pour chaque règle une règle correspondante. Les changements ne portent que sur les symboles fonctionnels probabilistes qui sont remplacés par les symboles déterministes.

Les système de déduction pour l'algèbre T^{h}

Dans le système de déduction de la figure 2.4, nous avons éliminé les règles de déduction de la ligne *Signature*. Toutes les autres règles n'ont pas été changées par rapport au système de déduction de la figure 2.3.

Les système de déduction pour l'algèbre T

Le système de déduction pour cette algèbre est donné dans la figure 2.5. Il est presque identique au système de déduction pour l'algèbre T^{h}, à l'exception que nous y avons supprimé la règle pour le fonction de hachage, car cette algèbre ne contient plus ce symbole fonctionnel.

2.3 Protocoles

Dans ce paragraphe, nous expliquons comment les protocoles sont spécifiés syntaxiquement dans nos modèles. Ceux-ci se distinguent seulement par l'algèbre utilisée. Nous adaptons la notation $T^{\star}$, comme précédemment, pour une des algèbres T^{lhs}, T^{hs}, T^{h} ou T. L'ensemble $X^{\star}$ désigne alors les variables correspondant et la relation $\vdash^{\star}$ le système de déduction utilisé.

Indépendemment de l'algèbre sous-jacente, nous présentons ensuit un modèle d'exécution pour donner une sémantique aux spécifications syntaxiques.

2.3.1 Syntaxe

Les rôles et protocoles

Regardons l'exemple de la section 1.1. On a défini deux rôles A et B. Le premier rôle envoie un message de la forme $\langle A, N_A \rangle$ et le deuxième répond en envoyant le nonce N_B. Nous voulons formellement décrire ce comportement. Nous donnons donc d'abord la définition d'une étape, puis d'un rôle qui est une séquence d'étapes.

Définition 6. Nous appelons *étape* une paire dans $((\{\mathsf{init}\} \cup T^{\star}) \times (T^{\star} \cup \{\mathsf{stop}\}))$.

Définition 7. Une *rôle* est une séquence d'étapes. L'ensemble de rôles $\mathsf{Roles}^{\star}$ est donné par $\mathsf{Roles}^{\star} = ((\{\mathsf{init}\} \cup T^{\star}) \times (T^{\star} \cup \{\mathsf{stop}\}))^{*}$.

Un rôle définit le comportement à suivre par un participant d'un protocole. Il décrit une séquence de messages reçus, ainsi que les réponses respectives. La sémantique informelle d'un rôle $(l_i, r_i)_{1 \leq i \leq n}$ est la suivante : les étapes sont parcouru de 1 jusqu'à n pendant l'exécution du rôle. À l'étape i, un participant attend un message m, unifiable avec le terme l_i, c'est-à-dire qu'un unificateur σ existe, tel que $m = l_i\sigma$. Puis, il envoie le message $r_i\sigma$. Les termes l_i et r_i ne sont donc pas les messages eux-même, mais des patrons qui décrivent la forme d'un message. Lorsqu'un participant a répondu au message attendu, il passe à l'étape suivante $i + 1$.

La constante init nous sert à marquer qu'un participant peut entamer spontanément l'exécution d'un rôle, comme c'est le cas pour le rôle A de l'exemple 1.1. La constante stop est utilisée,

lorsqu'un participant a reçu le dernier message destiné à son rôle et n'y répond plus, parce qu'il considère l'exécution comme terminée. C'est également le cas pour le rôle A de l'exemple mentionné.

Nous généralisons maintenant la notion de protocole : un protocole à k participants est donné par k rôles.

Définition 8. Un *protocole à k participants* est une application $\Pi : [k] \to \mathsf{Roles}^\star$, où $[k]$ note l'ensemble $\{1, \ldots, k\}$. Pour $i \in [k]$, nous appelons $\Pi(i)$ le i-ième rôle du protocole Π.

Une convention pour le nommage de variables

Dans la spécification d'un rôle de protocole, il est nécessaire de distinguer les variables qui sont instanciées par ce rôle des variables qui prennent des valeurs déterminées par les messages reçus. En outre, il est nécessaire de spécifier quelles variables représentent les agents, quelles les nonces et quelles les clefs.

Pour éviter à chaque fois ces précisions fastidieuses concernant les variables, nous adoptons la convention suivante : soit $k \in \mathbb{N}$ le nombre des rôles d'un protocole. Alors, nous notons l'ensemble des variables d'agents par $X.a = \{A_1, ..., A_k\}$ où A_1 représente l'agent qui joue le premier rôle, A_2 l'agent qui joue le deuxième rôle et ainsi de suite. Nous désignons par $X_n(A)$ les variables des nonces générés par l'agent représenté par la variable $A \in \mathsf{X}.a$, défini par $X_n(A) = \{X_A^j \mid j \in \mathbb{N}\}$. L'ensemble des variables de nonces est alors donné par $X.n = \bigcup_{A \in X.a} X_n(A)$. De manière tout à fait analogue, nous définissons l'ensemble $X_k(A)$ des variables des clefs symétriques à court terme d'un agent représenté par la variable $A \in X.a$ par $X_k(A) = \{K_A^j \mid j \in \mathbb{N}\}$. L'ensemble $X.k$ des variables des clefs symétriques à court terme est donc défini par $X.k = \bigcup_{A \in X.a} X_k(A)$.

Cette convention nous permet d'omettre, lors d'une spécification de protocole, la précision quelle variable représente l'agent qui joue le rôle, ainsi que quels nonces doivent être générés pendant l'exécution du rôle.

Exemple 7. Le protocole de Needham-Schroeder-Lowe [Low96] est un protocole créé pour assurer l'authentification mutuelle de deux participants. Il est informellement donné par les règles

$$\begin{aligned} A \to B &: \{N_A, A\}_{\mathsf{ek}(B)} \\ B \to A &: \{N_A, N_B, B\}_{\mathsf{ek}(A)} \\ A \to B &: \{N_B\}_{\mathsf{ek}(B)}. \end{aligned}$$

Dans un premier temps, l'agent qui joue le rôle A envoie la paire $\langle N_A, A\rangle$, chiffrée asymétriquement avec la clef publique $\mathsf{ek}(B)$, à celui qui joue le rôle B. Dans ce message, N_A représente un nonce généré par cet agent et A représente son nom. Pendant la deuxième étape, l'agent qui joue le rôle B renvoie le nonce N_A qu'il a reçu avec un nonce N_B généré par lui-même et son nom B. Avant d'émettre ce message, il chiffre le tout avec la clef publique $\mathsf{ek}(A)$. L'agent A qui reconnaît son nonce N_A dans ce message pense qu'il a parlé à B, puisque c'est le seul agent qui a pu déchiffrer le message envoyé précédemment et ainsi obtenir ce nonce. Il renvoie donc le nonce N_B qu'il a reçu de l'agent B, chiffré avec la clef publique de celui-ci. Lorsque celui-ci reçoit ce dernier message, il pense avoir parlé à A, car c'est le seul agent qui a pu déchiffrer le deuxième message, contenant le nonce N_B. À la fin, les agents A et B peuvent conclure qu'ils ont communiqué l'un avec l'autre.

Nous voulons traduire ce protocole dans notre formalisme. À cette fin, nous choisissons l'algèbre T^{lhs}. Respectant la convention que nous avons choisie pour le nommage de variables, le protocole s'écrit alors de la manière suivante :

$$\begin{aligned}\Pi(1) &= (\mathsf{init}, \{X^1_{A_1}, A_1\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_2)}), (\{X^1_{A_1}, X^1_{A_2}, A_2\}^{L}_{\mathsf{ek}(A_1)}, \{X^1_{A_2}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_2)})\\ \Pi(2) &= (\{X^1_{A_1}, A_1\}^{L_1}_{\mathsf{ek}(A_2)}, \{X^1_{A_1}, X^1_{A_2}, A_2\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_1)}), (\{X^1_{A_2}\}^{L_2}_{\mathsf{ek}(A_2)}, \mathsf{stop})\end{aligned}$$

La variable A_1 représente le nom du participant qui joue le rôle de l'émetteur. Le nonce qu'il génère pendant l'exécution est représenté par la variable $X^1_{A_1}$. La variable A_2 représente le nom de l'agent qui joue le rôle du receveur. Le nonce qu'il génère est représenté par la variable $X^1_{A_2}$. Mais étant donné que nous avons choisi une convention de nommage, nous ne sommes plus obligés de préciser cela.

Les protocoles exécutables

Notons qu'il n'est pas exclu par le formalisme de spécifier des protocoles qui ne sont pas exécutables. Considérons par exemple un rôle $\Pi(1) = (\mathsf{init}, \{X^1_{A_2}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_1)}) \in \mathsf{Roles}^{lhs}$ où le participant, représenté par la variable A_1, doit envoyer un nonce chiffré avec sa clef publique. Le problème provient du fait que la variable de nonce $X^1_{A_2}$ qui figure ici n'appartient pas à l'ensemble de variables qui sont instanciées par le participant. En effet, elle représente une valeur de nonce qui est générée par l'agent qui joue le deuxième rôle du protocole Π. Le participant représenté par la variable A_1 ne peut pas connaître ce nonce. D'autres situations où un protocole peut s'avérer non exécutable apparaissent, lorsqu'un participant doit déchiffré un message avec une clef privée qui n'est pas la sienne, ou lorsqu'il doit signer un message avec une clef de signature autre que sa propre clef de signature.

Intuitivement, on peut dire qu'un protocole est exécutable, s'il peut être implémenté. Au niveau symbolique, cela signifie qu'un participant peut, à l'aide d'un système de déduction, construire tous les messages qu'il émet à partir des connaissances publiques, de ses propres connaissances initiales, ainsi que des messages qu'il a déjà reçus au cours de l'exécution du protocole. Les connaissances publiques sont les identités d'agents et leurs clefs publiques. Les connaissances initiales d'un agent consistent normalement en ses nonces et ses clefs privées. Cependant, qu'un protocole soit exécutable, cela n'est pas une condition sine qua non pour notre résultat de correspondance, présenté dans le chapitre 3. Nous nous contentons donc de cette explication informelle et renvoyons sur l'article [CW05] pour une définition formelle.

2.3.2 Modèle d'exécution

Il est évident que nous ne pourrons raisonner sur les propriétés d'un protocole que si nous lui donnons une sémantique. Pour ce faire, nous définissons un *modèle de transitions* qui est adapté a nos besoins et que nous appelons aussi *modèle d'exécution*. Dans un tel modèle, l'exécution d'un protocole est modélisée par une séquence finie d'*états globaux* et des *transitions* entre ces états globaux. Un état global contient tous les informations disponibles sur les messages échangés et les *états locaux* des agents. L'état local d'un agent est comparable à l'état de son mémoire à un certain moment d'exécution du protocole. Quant aux transitions, nous considérons qu'elles sont provoquées par l'intrus. Une transition correspond à une de ses capacités, à savoir à corrompre des agents, commencer de nouvelles sessions et envoyer des messages. Nous obtenons ainsi une *trace d'exécution* d'un protocole qui est une suite d'états globaux et de transitions. Les propriétés d'un protocole peuvent être définies en fonction des traces d'exécution.

Les connaissances d'un agent

Dans notre modèle d'exécution, les *connaissances* d'un agent consistent en ses clefs privées, à savoir des clefs de chiffrement symétrique et asymétrique, ainsi que sa clef de signature, si l'algèbre utilisée contient cette fonction. Étant donné que nous n'avons plus la signature dans les algèbres T^h et T, nous définissons deux types de connaissances pour un agent $a \in \mathsf{ID}$: $\mathbf{kn}^{lhs}(a) = \mathbf{kn}^{hs}(a) = \mathsf{SKLong}(a) \cup \{\mathsf{dk}(a), \mathsf{sk}(a)\}$ et $\mathbf{kn}^{h}(a) = \mathbf{kn}(a) = \mathsf{SKLong}(a) \cup \{\mathsf{dk}(a)\}$. Nous remarquons que la connaissance d'un agent a ne comprend pas l'ensemble de ses nonces $\mathsf{Nonce}(a)$ ni celui de ses clefs symétriques à court terme $\mathsf{SKShort}(a)$. Nous considérons les valeurs de ce genre comme étant générées par une source d'aléa pour une session unique. Par conséquent, un agent ne les connaît qu'au moment où il les utilise dans cette session.

Nous adoptons la notation $\mathbf{kn}^{\star}(a)$ pour l'ensemble de connaissances qui correspond à l'utilisation de l'algèbre $T^{\star}$.

Identifiants de sessions et états locaux

Pour exécuter un protocole, nous avons besoin de la notion de session. Nous distinguons celles-ci à l'aide des identifiants de session.

Définition 9. Soit $\Pi : [k] \to \mathsf{Roles}^{\star}$ un protocole à k participants. Un *identifiant de session* est un triplet dans $\mathsf{SID} = \mathbb{N} \times [k] \times \mathsf{ID}^k$.

Soit $(n, j, (a_{i_1}, \ldots, a_{i_k}))$ un identifiant de session. L'entier n donne le numéro de la session et le k-uplet $(a_{i_1}, \ldots, a_{i_k})$ consiste en les agents qui sont impliqués dans cette session. L'entier j donne le numéro du rôle qui est exécuté. Nous verrons plus tard que c'est l'agent a_{i_j} qui exécute ce rôle.

Pour un identifiant de session, nous définissons l'état de cette session.

Définition 10. L'état d'une session est une paire dans $\mathbb{S} \times \mathbb{N}$. Nous appelons une telle paire un *état local*.

Rappelons que $\mathbb{S}$ est l'ensemble de substitutions, c'est-à-dire de fonctions de l'ensemble de variables X^{lhs} dans l'ensemble de termes $T^{lhs} \cup \mathcal{L}$. Soit (σ, p) l'état d'une session. L'entier p donne l'étape actuelle, à laquelle se trouve l'exécution du rôle joué dans cette session. Le domaine de la substitution σ contient toutes les variables qui apparaissent dans la définition de ce rôle et qui ont déjà été instanciées pendant son exécution. Nous nous servons donc de cette substitutions pour stocker les termes qui ont été affectés à ces variables lors d'arrivée de nouveaux messages.

Exemple 8. Nous définissons plus tard formellement comment identifiants de sessions et des états locaux sont créés et utilisés, mais nous illustrons déjà leurs significations à travers d'un exemple. Regardons le protocole de Needham-Schroeder-Lowe, exemple 7. Nous considérons une exécution normale, pendant laquelle deux agents a et $b \in \mathsf{ID}$ communiquent. L'identifiant de cette session, concernant le premier rôle, est donné par $(1, 1, (a, b))$. Au début d'exécution, on associe à cet identifiant l'état local $(\sigma_1, 1)$. L'entier 1 signifie qu'il s'agit de la première étape et que l'agent a attend donc un message de la forme init. La substitution σ_1 est donné par $\{X^1_{A_1} \leftarrow n(a,1,1), A_1 \leftarrow a, A_2 \leftarrow b\}$. Après avoir reçu la constante attendue, l'état de la session change en $(\sigma_2, 2)$ où $\sigma_2 = \sigma_1$. L'agent a attend maintenant un message unifiable avec $\{X^1_{A_1}, X^1_{A_2}, A_2\}^L_{\mathsf{ek}(A_1)}\sigma = \{n(a,1,1), X^1_{A_2}, b\}^L_{\mathsf{ek}(a)}$. Après avoir reçu un tel message, l'état local de la session est donné par $(\sigma_3, 3)$ où $\sigma_3 = \sigma_2 \cup \{X^1_{A_2} \leftarrow n(b,1,1), L \leftarrow \mathsf{ag}(1)\}$. Précisons encore une fois que nous avons considéré une exécution normale du protocole et que nous définissons les éléments manquants pour comprendre entièrement cet exemple dans la suite.

Les états globaux

Nous définissons maintenant la notion d'état global.

Définition 11. Un *état global* est un triplet (SId, f, H) dans $\mathsf{GState}^\star = 2^{\mathsf{SID}} \times (\mathsf{SID} \to (\mathbb{S} \times \mathbb{N})) \times 2^{T^\star}$, où $\mathsf{SId} \subseteq \mathsf{SID}$ est un ensemble d'identifiants de session, $f : \mathsf{SId} \to \mathbb{S} \times \mathbb{N}$ une fonction qui associe un état local à chaque identifiant de session dans SId et $H \subseteq T^\star$ un ensemble de termes.

Dans un état global (SId, f, H), l'ensemble de termes $H \subseteq T^\star$ modélise l'ensemble de messages qui ont été émis par les participants pendant l'exécution du protocole jusqu'à cet état. C'est cet ensemble dont l'intrus peut se servir pour déduire de nouveaux messages afin de les envoyer ensuite aux participants. Nous rappelons que l'étoile $\star$ désigne un modèle (*lhs*, *hs*, *h* ou _) et ne doit pas être confondu avec l'étoile $*$ qui est utilisée pour définir les séquences finies des éléments d'un ensemble.

Les transitions

Définition 12. Soit $\Pi : [k] \to \mathsf{Roles}^\star$ un protocole à k participants. Nous définissons l'ensemble de *transitions* par $\mathsf{Trans}^\star = (\{\mathbf{corrupt}\} \times \mathsf{ID}^*) \cup (\{\mathbf{new}\} \times [k] \times \mathsf{ID}^k) \cup (\{\mathbf{send}\} \times \mathsf{SID} \times T^\star)$.

Les transition du type *corrupt* sont aussi dénotées par $\mathbf{corrupt}(a_1, \ldots a_l)$ et les transitions du type *new* par $\mathbf{new}(j, a_1, \ldots a_k)$. De même, nous notons $\mathbf{send}(\mathsf{sid}, m)$ les transitions du type *send*. Les trois types de transitions sont provoqués par l'intrus. Nous donnons leur description.

- L'intrus peut corrompre un nombre arbitraire $l \in \mathbb{N}$ d'agents :

$$(\mathsf{SId}, f, H) \xrightarrow{\mathbf{corrupt}(a_{i_1}, \ldots, a_{i_l})} (\mathsf{SId}, f, \bigcup_{1 \le j \le l} \mathbf{kn}^\star(a_{i_j}) \cup H).$$

  Cela lui permet de connaître leurs clefs privées. La corruption n'est permise que pour la première transition d'une trace d'exécution.
- L'intrus peut commencer une nouvelle session d'un rôle $\Pi(j)$ du protocole, où $j \in [k]$:

$$(\mathsf{SId}, f, H) \xrightarrow{\mathbf{new}(j, a_{i_1}, \ldots, a_{i_k})} (\mathsf{SId}', f', H').$$

  Le nouveau état (SId', f', H') est donné comme suit : nous notons $|\mathsf{SId}|$ la cardinalité de l'ensemble d'identifiants de session SId. Le numéro de la nouvelle session est $n = |\mathsf{SId}| + 1$ et l'identifiant de cette session est $\mathsf{sid} = (n, j, (a_{i_1}, \ldots, a_{i_k}))$. Le nouvel ensemble SId' est défini par $\mathsf{SId}' = \mathsf{SId} \cup \{\mathsf{sid}\}$.
  La fonction f' est définie par
  - $f'(\mathsf{sid}) = f(\mathsf{sid})$ pour tout identifiant de session $\mathsf{sid} \in \mathsf{SId}$. Cela signifie que les états locaux des anciennes sessions ne changent pas.
  - $f'((n, j, (a_{i_1}, \ldots, a_{i_k}))) = (\sigma, 1)$. L'entier 1 signifie que l'exécution du rôle $\Pi(j)$ se trouve à la première étape. La substitution σ est donnée par

$$\begin{array}{lcll} \sigma(A_m) & = & a_{i_m} & m \in [k] \\ \sigma(X^m_{A_j}) & = & n(a_{i_j}, m, n) & m \in \mathbb{N} \\ \sigma(K^m_{A_j}) & = & k(a_{i_j}, m, n) & m \in \mathbb{N} \end{array}$$

  Nous rappelons que les variables qui représentent les participants d'un protocole à k sessions sont appelées $A_1, \ldots, A_k$ et que c'est la variable A_j qui représente l'agent qui joue le

rôle $\Pi(j)$. Toutes ces variables d'agents sont initialisées avec les valeurs d'identités d'agents $a_{i_1}, \ldots, a_{i_k}$. Cela signifie en particulier que c'est l'agent a_{i_j} qui exécute le rôle et qu'il pense exécuter le protocole avec les agents a_{i_m}, $m \neq j$, mais les sessions correspondantes n'existent pas forcément. Toutes les variables $X^m_{A_j}$ et $K^m_{A_j}$ qui représentent les nonces et clefs générés par A_j sont aussi initialisées avec des termes représentant ces valeurs.
Finalement, nous posons $H' = H$. Cela modélise qu'aucun message n'a été émis sur le réseau pendant cette transition.

– L'intrus peut envoyer des messages $m \in T^\star$:

$$(\mathsf{SId}, f, H) \xrightarrow{\mathbf{send}(\mathsf{sid}, m)} (\mathsf{SId}', f', H')$$

où $\mathsf{sid} \in \mathsf{SId}$ est l'identifiant d'une session. L'ensemble des identifiants de session ne change pas. Nous posons $\mathsf{SId}' = \mathsf{SId}$. La fonction f' et l'ensemble de messages H' sont définis de la façon suivante : nous posons $f'(\mathsf{sid}') = f(\mathsf{sid}')$ pour toute session $\mathsf{sid}' \neq \mathsf{sid}$ dans SId, c'est-à-dire que les états locaux des sessions qui ne sont pas concernées dans cette transition ne changent pas.
L'identifiant de session sid est un triplet de la forme $\mathsf{sid} = (s, j, (a_{i_1}, \ldots, a_{i_k}))$ et $f(\mathsf{sid}) = (\sigma, p)$ est l'état local attribué à cet identifiant. Soit $\Pi(j) = (l_i, r_i)_{1 \leq i \leq n}$ le j-ème rôle du protocole. Nous devons considérer deux cas :
  – soit, $p \leq n$ et l'unificateur le plus général θ entre le message m et le patron $l_p\sigma$ existe. Dans ce cas, nous avons $m = l_p\sigma\theta$. Alors, nous posons $f'(\mathsf{sid}) = (\sigma \cup \theta, p + 1)$ et $H' = H \cup \{r_p\sigma\theta\}$. La mise à jour de l'état local consiste alors à remplacer la substitution actuelle σ par la substitution étendue $\sigma \cup \theta$ et à incrémenter le compteur d'étapes p. De plus, on ajoute la réponse émise par l'agent a_{i_j} à l'ensemble H.
  – soit, nous posons $f'(\mathsf{sid}) = f(\mathsf{sid})$ and $H' = H$, c'est-à-dire l'état global ne change pas. Ce cas arrive lorsqu'un participant considère l'exécution du rôle comme terminée, c'est-à-dire $p > n$, ou lorsque il reçoit un message qui n'est pas conforme à ses attentes, c'est-à-dire qu'il n'y a pas d'unificateur θ tel que $m = l_p\sigma\theta$.

Les traces d'exécution

Nous définissons maintenant l'ensemble de traces d'exécution. Une trace décrit l'historique de l'exécution d'un protocole. Elle consiste en les séquences alternantes d'états globaux et de transitions. Ensuite, nous donnons un exemple d'une trace d'exécution.

Définition 13. Nous notons $\mathsf{SymbTr}^\star$ l'ensemble de *traces d'exécution*, donné par $\mathsf{SymbTr}^\star = \mathsf{GState}^\star(\mathsf{Trans}^\star \times \mathsf{GState}^\star)^*$.

Exemple 9. Considérons encore le protocole de l'exemple 7. Nous donnons le début d'une trace d'exécution qui se joue entre deux participants. Elle correspond au premier envoi et à la première réception d'un message :

$$\begin{aligned}(\emptyset, f_1, H_1) &\xrightarrow{\mathbf{new}(1,a_1,a_2)} (\{\mathsf{sid}_1\}, f_2, H_2) \\ &\xrightarrow{\mathbf{send}(\mathsf{sid}_1,\mathsf{init})} (\{\mathsf{sid}_1\}, f_3, H_3), \xrightarrow{\mathbf{new}(2,a_1,a_2)} (\{\mathsf{sid}_1, \mathsf{sid}_2\}, f_4, H_4) \\ &\xrightarrow{\mathbf{send}(\mathsf{sid}_2,\{n(a_1,1,1),a_1\}^{\mathsf{ag}(1)}_{\mathsf{ek}(a_1)})} (\{\mathsf{sid}_1, \mathsf{sid}_2\}, f_5, H_5) \ldots\end{aligned}$$

Nous posons $H_1 = \mathbf{kn}^{lhs}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}$ où $\mathsf{int} \in \mathsf{ID}$ est une identité d'agent particulier. Nous l'utilisons au début d'une exécution du protocole pour mettre des termes

de nonce et des clefs symétriques à court terme à la disposition de l'intrus. À chaque transition, les mises à jour sont les suivants :

1. $\mathbf{new}(1, a_1, a_2) : \mathsf{sid}_1 = (1, 1, (a_1, a_2))$ et $f_2(\mathsf{sid}_1) = (\sigma_1, 1)$, avec $\sigma_1 = \{A_1 \leftarrow a_1, A_2 \leftarrow a_2, X^1_{A_1} \leftarrow n(a_1, 1, 1)\}$, $H_2 = H_1$.
2. $\mathbf{send}(\mathsf{sid}_1, \mathsf{init})$: $f_3(\mathsf{sid}_1) = (\sigma'_1, 2)$. Remarquons que cette transition n'est qu'auxiliaire pour démarrer l'exécution du protocole. Cela correspond au scénario où un participant entame spontanément une communication. Nous avons $\sigma'_1 = \sigma_1$ et $H_3 = H_2 \cup \{\{n(a_1, 1, 1), a_1\}^{\mathsf{ag}(1)}_{\mathsf{ek}(a_1)}\}$.
3. $\mathbf{new}(2, a_1, a_2) : \mathsf{sid}_2 = (2, 2, (a_1, a_2))$, $f_4(\mathsf{sid}_1) = f_3(\mathsf{sid}_1)$ et $f_4(\mathsf{sid}_2) = (\sigma_2, 1)$ avec $\sigma_2 = \{A_1 \leftarrow a_1, A_2 \leftarrow a_2, X^1_{A_2} \leftarrow n(a_2, 1, 2)\}$, $H_4 = H_3$.
4. $\mathbf{send}(\mathsf{sid}_2, \{n(a_1, 1, 1), a_1\}^{\mathsf{ag}(1)}_{\mathsf{ek}(a_1)})$: $f_5(\mathsf{sid}_1) = f_4(\mathsf{sid}_1)$, $f_5(\mathsf{sid}_2) = (\sigma'_2, 2)$ avec $\sigma'_2 = \sigma_2 \cup \{X^1_{A_1} \leftarrow n(a_1, 1, 1), L_1 \leftarrow \mathsf{ag}(1)\}$, $H_5 = H_4 \cup \{\{n(a_1, 1, 1), n(a_2, 1, 2), a_2\}^{\mathsf{ag}(2)}_{\mathsf{ek}(a_1)}\}$.

2.4 Traces valides et propriétés de traces

Dans cette section, nous distinguons d'abord un sous-ensemble de traces symboliques. Afin de motiver la nécessité de cette distinction, nous commençons avec un exemple. Dans la deuxième partie, nous définissons ce que c'est qu'une propriété de protocole.

Les traces valides

Exemple 10. Nous choisissons pour cet exemple l'algèbre T^{hs}. Considérons la trace

$$tr = (\{\mathsf{sid}\}, f_1, H_1) \xrightarrow{\mathbf{send}(\mathsf{sid}, n(a_1,1,1))} (\{\mathsf{sid}\}, f_2, H_2)$$

où $H_1 = \{\{n(a_1, 1, 1)\}_{\mathsf{ek}(a)}\}$. Nous avons $tr \in \mathsf{SymbTr}^{hs}$. Le problème dans cette trace provient du fait que l'intrus envoie le nonce $n(a_1, 1, 1)$. Comment a-t-il pu obtenir ce nonce ? Il y a une seule réponse à cette question : l'intrus a pu déduire le terme à partir des messages qu'il connaît. Le nonce serait donc déductible à partir de l'ensemble H_1, c'est-à-dire $H_1 \vdash^{hs} n(a_1, 1, 1)$, ce qui n'est pas le cas.

La réponse au problème présenté dans cet exemple est que, bien qu'il s'agisse d'une trace symbolique, elle ne correspond à aucune exécution possible d'un protocole. Nous devons donc distinguer un sous-ensemble des traces symboliques de $\mathsf{SymbTr}^\star$.

Une trace d'exécution doit satisfaire certaines propriétés afin d'être valide. Intuitivement, chaque transition dans une trace doit être une transition **new** ou **send** à l'exception de la première qui peut être une transition **corrupt**. En outre, les messages envoyés doivent être déductibles à partir de la connaissance de l'intrus qui se compose, d'une part, des clefs privées des agents corrompus et, d'autre part, des messages que les participants émettent pendant l'exécution du protocole. Dans la définition des traces valides, nous nous servons d'une identité d'agent $\mathsf{int} \in \mathsf{ID}$ pour donner des termes de nonces et des clefs symétriques à court terme à l'intrus. Par conséquent, la constante int appartient toujours aux agents corrompus.

Définition 14. Soit $\Pi : [k] \to \mathsf{Roles}^\star$ un protocole à k participants et soit $\mathsf{int} \in \mathsf{ID}$ une identité d'agent. La *trace d'exécution* $(\mathsf{SId}_1, f_1, H_1)$ *est valide*, si

– $\mathsf{SId}_1 = \emptyset$ et $H_1 = \mathbf{kn}^\star(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}$.

Une *trace d'exécution* $(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2 \le i \le n}$ *est valide*, si de plus

– la transition $\mathbf{trans}_1$ est une des trois transitions décrites ci-dessus,
– pour $i \geq 2$, la transition $\mathbf{trans}_i$ est une des deux dernières transitions décrites ci-dessus (**new** ou **send**),
– pour toute transition $(\mathsf{SId}_i, f_i, H_i) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}_{i+1}, f_{i+1}, H_{i+1})$, nous avons $H_i \vdash^\star m$.

L'ensemble des traces d'exécution valides est dénote par $\mathsf{Exec}^\star(\Pi)$.

Les propriétés d'un protocole

Pour un protocole donné, nous nous intéressons aux propriétés que satisfait ce protocole. Une propriété, en toute généralité, est un sous-ensemble des traces symboliques. Nous disons qu'un protocole satisfait une propriété, si toutes les traces valides sont inclues dans ce sous-ensemble :

Définition 15. Soit $\Pi : [k] \to \mathsf{Roles}^\star$ un protocole à k participants et soit $\phi \subseteq \mathsf{SymbTr}^\star$ une propriété de trace. Si toutes les traces d'exécution valides du protocole satisfont la propriété ϕ, c'est-à-dire $\mathsf{Exec}^\star(\Pi) \subseteq \phi$, alors nous disons que le protocole Π *satisfait la propriété* ϕ.

Exemple 11. Soit $\Pi(1) = (l_i^1, r_i^1)_{1 \leq i \leq k} \in \mathsf{Roles}^{lhs}$ le rôle d'un protocole, dans lequel la variable de nonce $X^1_{A_1}$ apparaît. La propriété suivante exprime que le nonce représenté par la variable $X^1_{A_1}$ est secret, tant que le participant qui le génère ne soit pas corrompu :

$$\begin{aligned}\phi = \{(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2 \leq i \leq n} \mid \\ \text{pour } \mathbf{trans}_1 = \mathbf{corrupt}(a_1, \ldots, a_l), \text{ si } \forall \mathsf{sid} = (s, j, (\ldots)) \in \mathsf{SId}_n \\ \text{telle que } f_n(\mathsf{sid}) = (\sigma, p) \text{ où } \sigma(X^1_{A_1}) = n(\sigma(A_1), 1, s), \text{ alors} \\ \sigma(A_1) \notin \{a_1, \ldots, a_l\} \implies H_n \not\vdash^{lhs} \sigma(X^1_{A_1})\}\end{aligned}$$

Rappelons que, dans le premier rôle, c'est la variable A_1 qui représente le participant qui joue le rôle et la variable $X^1_{A_1}$ qui représente le premier nonce. Si nous avons $\sigma(X^1_{A_1}) = n(\sigma(A_1), 1, n)$ pour un état local (σ, p), alors nous pouvons être sûr que c'est l'identité d'agent $\sigma(A_1)$ qui a instancié la variable $X^1_{A_1}$. La session sid est donc une session, dans laquelle le premier rôle est exécuté. Par conséquent, si l'agent représenté par A_1 est non corrompu, alors le nonce représenté par $X^1_{A_1}$ restera secret.

L'exemple montre qu'une propriété qui peut intuitivement exprimer dans une phrase simple est dans une représentation formelle difficile à lire même pour quelqu'un qui connaît les détails du modèle.

2.5 Logiques pour la spécification des propriétés de sécurité

Dans cette section nous proposons des logiques conçues pour la spécification des propriétés de traces. Nous précisons d'abord quelques notions supplémentaires qui nous aiderons à définir ensuite les algèbres qui sont utilisées dans nos formules. Ensuite, nous définissons les formules et leur interprétation. À la fin de ce paragraphe, nous donnons quelques exemples qui montrent que ces logiques sont en effet expressives.

Définitions préliminaires

Dans la définition suivant, nous utilisons les mêmes ensembles de variables comme avant.

Définition 16. Nous définissons l'ensemble des *expressions substitutionnelles des variables de V* par

$$SubExpr(V) ::= Sub(V)$$

où Sub note un ensemble de variables de substitution et V est un des ensembles de variables $X.n$, $X.a$, $X.k$, $X.c$, $X.s$, $X.h$, $X.l$, X^{lhs}, X^{hs}, X^h ou X. Pour une variable $x \in V$ et une substitution $\varsigma \in Sub$, nous appelons $\varsigma(x)$ une *expression substitutionnelle de x*.

Les expressions substitutionnelles remplacent les variables comme termes de base.

2.5.1 Les termes et les formules

Étant donné que nous voulons exprimer des propriétés de traces pour plusieurs modèles d'exécution qui se distinguent principalement par l'algèbre utilisée pour la spécification des protocoles, nous devons tenir compte de ces différences dans la définition des termes utilisés dans les formules. Analoguement au nommage des quatre algèbres T^{lhs}, T^{hs}, T^h et T, nous notons les algèbres utilisées dans les formules T^{lhs}_{Sub}, T^{hs}_{Sub}, T^h_{Sub} et T_{Sub}. Nous adoptons aussi la notation $T^{\star}_{Sub}$ de manière analogue à la notation $T^{\star}$.

Avant de donner les définitions des algèbres, nous définissons un ensemble

$$SubExprId ::= \mathsf{ID} \cup SubExpr(X.a)$$

qui contient les identités des agents et les expressions substitutionnelles des variables de $X.a$.

Les termes de l'algèbre T^{lhs}_{Sub}

Nous commençons par l'algèbre T^{lhs}_{Sub}. Elle est définie par

$$\begin{aligned}
\mathcal{L}_{Sub} \quad &::= \quad SubExpr(X.l) \mid \mathsf{ag}(i) \mid \mathsf{adv}(i) \\
T^{lhs}_{Sub} \quad &::= \quad a \mid n(a,j,s) \mid k(a,j,s) \mid SubExpr(X^{hs}) \\
&\quad\; \mid \mathsf{ek}(SubExprId) \mid \mathsf{dk}(SubExprId) \\
&\quad\; \mid \mathsf{sk}(SubExprId) \mid \mathsf{vk}(SubExprId) \\
&\quad\; \mid \mathsf{symk}(SubExprId, SubExprId) \mid \langle T^{lhs}_{Sub}, T^{lhs}_{Sub} \rangle \mid \{\!|T^{lhs}_{Sub}|\!\}^{\mathcal{L}_{Sub}}_{T^{lhs}_{Sub}} \\
&\quad\; \mid \{T^{lhs}_{Sub}\}^{\mathcal{L}_{Sub}}_{\mathsf{ek}(SubExprId)} \mid \{T^{lhs}_{Sub}\}^{\mathcal{L}_{Sub}}_{\mathsf{dk}(SubExprId)} \mid [T^{lhs}_{Sub}]^{\mathcal{L}_{Sub}}_{\mathsf{sk}(SubExprId)} \mid hash(T^{lhs}_{Sub})
\end{aligned}$$

où $a \in \mathsf{ID}$ est une identité d'agent et $j, s \in \mathbb{N}$ sont des entiers. L'algèbre T^{lhs}_{Sub} est définie de manière tout à fait similaire à l'algèbre T^{lhs}. La différence réside dans fait que nous avons remplacé les variables par des expressions substitutionnelles. Une expression substitutionnelle autorisée par T^{lhs}_{Sub} est du même type que le type de variable autorisé par T^{lhs}. Si, par exemple, une variable du type agent était autorisé dans T^{lhs}, nous autorisons dans T^{lhs}_{Sub} une expression substitutionnelle d'une variable de $X.a$.

Les termes des algèbres T^{hs}_{Sub},T^h_{Sub} et T_{Sub}

Nous donnons aussi les définitions des algèbres T^{hs}_{Sub},T^h_{Sub} et T_{Sub}. Ce que nous avons dit sur les variables et les expressions substitutionnelles dans le paragraphe précédant est aussi valable pour ces algèbres.

$$
\begin{aligned}
T^{hs}_{Sub} \quad ::= \quad & a \mid n(a,j,s) \mid k(a,j,s) \mid SubExpr(X^{hs}) \\
& \mid \mathsf{ek}(SubExprId) \mid \mathsf{dk}(SubExprId) \\
& \mid \mathsf{sk}(SubExprId) \mid \mathsf{vk}(SubExprId) \\
& \mid \mathsf{symk}(SubExprId, SubExprId) \mid \langle T^{hs}_{Sub}, T^{hs}_{Sub} \rangle \mid \{\!|T^{hs}_{Sub}|\!\}_{T^{hs}_{Sub}} \\
& \mid \{T^{hs}_{Sub}\}_{\mathsf{ek}(SubExprId)} \mid \{T^{hs}_{Sub}\}_{\mathsf{dk}(SubExprId)} \mid [T^{hs}_{Sub}]_{\mathsf{sk}(SubExprId)} \mid hash(T^{hs}_{Sub}).
\end{aligned}
$$

$$
\begin{aligned}
T^{h}_{Sub} \quad ::= \quad & a \mid n(a,j,s) \mid k(a,j,s) \mid SubExpr(X^{h}) \\
& \mid \mathsf{ek}(SubExprId) \mid \mathsf{dk}(SubExprId) \\
& \mid \mathsf{symk}(SubExprId, SubExprId) \mid \langle T^{h}_{Sub}, T^{h}_{Sub} \rangle \mid \{\!|T^{h}_{Sub}|\!\}_{T^{h}_{Sub}} \\
& \mid \{T^{h}_{Sub}\}_{\mathsf{ek}(SubExprId)} \mid \{T^{h}_{Sub}\}_{\mathsf{dk}(SubExprId)} \mid hash(T^{h}_{Sub}).
\end{aligned}
$$

$$
\begin{aligned}
T_{Sub} \quad ::= \quad & a \mid n(a,j,s) \mid k(a,j,s) \mid SubExpr(X) \\
& \mid \mathsf{ek}(SubExprId) \mid \mathsf{dk}(SubExprId) \\
& \mid \mathsf{symk}(SubExprId, SubExprId) \mid \langle T_{Sub}, T_{Sub} \rangle \mid \{\!|T_{Sub}|\!\}_{T_{Sub}} \\
& \mid \{T_{Sub}\}_{\mathsf{ek}(SubExprId)} \mid \{T_{Sub}\}_{\mathsf{dk}(SubExprId)}
\end{aligned}
$$

Les formules

Nous proposons quatre logiques que nous nommons $\mathcal{L}_1^{lhs}$, $\mathcal{L}_1^{hs}$, $\mathcal{L}_1^{h}$, et $\mathcal{L}_1$. Elles se distinguent par l'ensemble de termes utilisé dans leurs formules. Ces ensembles sont respectivement les ensembles T^{lhs}_{Sub}, T^{hs}_{Sub}, T^{h}_{Sub} et T_{Sub} que nous venons de définir. Chacune de ces logiques permet d'exprimer des propriétés de traces d'exécution pour un protocole défini en utilisant respectivement une des algèbres T^{lhs}, T^{hs}, T^{h} et T. Si nous définissons par exemple un protocole avec l'algèbre T^{lhs}, nous devons utiliser la logique $\mathcal{L}_1^{lhs}$ pour exprimer des propriétés de traces pour ce protocole. Dans la logique elle-même, nous utilisons des termes de l'algèbre T^{lhs}_{Sub}.

Nous avons déjà dit que les logiques permettent d'exprimer des propriétés de traces d'exécution. Pour cette raison, la fonction de l'interprétation d'une formule prend en sus une trace comme paramètre. Elle renvoie vraie, si cette trace satisfait la formule, sinon, elle renvoie fausse. Plus précisément, on peut dire que nous exprimons des propriétés sur les états locaux qui apparaissent dans une trace d'exécution. Nous voulons, par exemple, dire qu'un certain état local ne fait pas partie des états qui peuvent apparaître pendant l'exécution d'un rôle, tant que l'agent qui joue ce rôle n'est pas corrompu. Nous définissons donc d'abord l'ensemble des états locaux qui peuvent apparaître à une certaine étape d'exécution d'un certain rôle.

Définition 17. Soit $(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n} \in \mathsf{SymbTr}^\star$ une trace symbolique. L'ensemble d'états locaux du rôle j à l'étape p est défini par

$$
\begin{aligned}
&\mathcal{LS}_{j,p}((\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n}) = \\
&\qquad \{(\sigma,p) \mid \exists i \in \{1,\dots,n\}\ \exists \mathsf{sid} = (s, j, (a_{i_1}, \dots, a_{i_k})) \in \mathsf{SId}_i, \text{ telle que } f_i(\mathsf{sid}) = (\sigma,p)\}
\end{aligned}
$$

Nous donnons maintenant la définition des formules. Dans la suite, nous expliquons l'intuition derrière certaines constructions syntaxiques avant de donner la définition formelle de l'interprétation d'une formule.

Définition 18. Les *formules* de la logique $\mathcal{L}_1^\star$ sont inductivement définies par

$$\begin{aligned} F \quad ::= \quad & NC(a) \mid (m_1 = m_2) \mid \neg F \mid F \wedge F \mid F \vee F \\ & \mid \forall \mathcal{LS}_{i,p}.\varsigma\ F \mid \exists \mathcal{LS}_{i,p}.\varsigma\ F \end{aligned}$$

où a, m_1 et $m_2 \in T^\star_{Sub}$ sont des termes, i et $p \in \mathbb{N}$ sont des entiers et $\varsigma \in Sub$ est une variable de substitution. Pour deux formules ϕ_1 et $\phi_2 \in \mathcal{L}_1^\star$, nous autorisons aussi la notation $\phi_1 \rightarrow \phi_2$ en tant qu'abréviation pour $\neg\phi_1 \vee \phi_2$.

Le prédicat NC permet de tester, si un agent est corrompu dans une trace d'exécution. Le prédicat $=$ teste, s'il y a égalité syntaxique entre deux termes. Les connecteurs $\wedge$ et $\vee$, ainsi que la négation sont interprétés d'une façon standard.

Les logiques possèdent deux quantificateurs particuliers : $\forall \mathcal{LS}_{i,p}.\varsigma$ et $\exists \mathcal{LS}_{i,p}.\varsigma$. Ils nous permettent de quantifier sur les états locaux qui peuvent apparaître dans une trace tr pour un rôle i à l'étape p. La variable de substitution ς sert comme caractère générique afin d'indiquer dans une formule les emplacements des substitutions qui apparaissent dans les états locaux sur lesquels on quantifie. Par exemple, la formule $\forall \mathcal{LS}_{1,p}.\varsigma\ NC(\varsigma(A_1))$ exprime la propriété qu'un agent qui joue le rôle 1 d'un protocole n'est jamais corrompu à l'étape p. Autrement dit, pour tout état local $(\sigma, p) \in \mathcal{LS}_{1,p}(tr)$ où tr est une trace du protocole, l'agent $\sigma(A_1)$ est non corrompu.

L'interprétation des formules

Définition 19. Si toutes les variables de substitution qui apparaissent dans les termes d'une formule sont quantifiées, alors la formule est *close*.

Exemple 12. Dans la formule $\forall \mathcal{LS}_{i,p}.\varsigma\ (\varsigma(x) = \tau(x))$, la substitution ς est quantifiée, tandis que la substitution τ ne l'est pas. Dans la formule $\forall \mathcal{LS}_{i,p}.\varsigma\ \exists \mathcal{LS}_{j,q}.\tau\ (\varsigma(x) = \tau(x))$, toutes les substitutions sont quantifiées. C'est une formule close.

Afin d'évaluer la valeur booléenne d'une formule, nous avons besoin d'une fonction d'interprétation. Elle est définie pour les formules closes et des traces d'exécution.

Définition 20. Nous notons 0 pour *faux* et 1 pour *vrai*. L'*interprétation* d'une formule close pour une trace est une application $[\![\_, \_]\!] : \mathcal{L}_1^\star \times \mathsf{SymbTr}^\star \rightarrow \{0, 1\}$.

L'interprétation est récursivement définie comme suit :

1. Le prédicat NC.

$$[\![NC(a), tr]\!] = \begin{cases} 1 & \text{si } a \in \mathsf{ID} \text{ et } a \neq \mathsf{int} \text{ n'apparaît pas dans une transition } \textbf{corrupt} : \\ & \text{pour } tr = e_1 e_2 \ldots e_n \text{ et } a_1, ..., a_k, \\ & \text{tels que } e_1 \xrightarrow{\textbf{corrupt}(a_1,\ldots,a_k)} e_2 \text{ nous avons } a \neq a_i \text{ et } a \neq \mathsf{int} \\ 0 & \text{sinon} \end{cases}$$

   Étant donnée une trace $tr = e_1 \ldots e_n$ et l'identité d'un agent $a \in \mathsf{ID}$, le prédicat $NC(a)$ est vrai, si l'agent a n'a pas été corrompu dans l'exécution décrite par la trace tr, c'est-à-dire qu'il ne paraît pas dans une transition **corrupt**. Remarquons que la corruption est

statique : elle ne se fait que au début de la trace. Pour cette raison, il suffit de regarder la transition entre les états globaux e_1 et e_2.

Remarquons que le prédicat NC est évalué après avoir appliqué les substitutions qui peuvent y apparaître. Prenons par exemple une formule $NC(\varsigma(A))$ où la variable de substitution ς prend la valeur $\tau = \{A \leftarrow a\}$. L'interprétation de cette formule pour une trace tr se fait par $[\![NC(\varsigma(A))[\varsigma \leftarrow \tau], tr]\!] = [\![NC(\tau(A)), tr]\!] = [\![NC(a), tr]\!] = \ldots$. Dans cet exemple, on voit bien, qu'en effet, les éléments de Sub remplissent la fonction des variables qui prennent comme valeur une substitution. Après l'application de celle-ci, le prédicat NC est évalué.

2. Égalité syntaxique.
$$[\![(m_1 = m_2), tr]\!] = \begin{cases} 1 & \text{si } m_1 = m_2 \\ 0 & \text{sinon} \end{cases}$$
Les tests d'égalité se passent au niveau syntaxique après avoir appliquer d'éventuelles substitutions. Considérons par exemple la substitution $\tau = \{A \leftarrow b, B \leftarrow a\}$. L'interprétation d'une formule comme $(\varsigma(A) = b)$ pour une trace tr où la variable de substitution $\varsigma \in Sub$ prend la valeur τ se fait par $[\![(\varsigma(A) = b)[\varsigma \leftarrow \tau], tr]\!] = [\![(\tau(A) = b), tr]\!] = [\![(b = b), tr]\!] = 1$, tandis que la formule $(\varsigma(B) = b)$ est interprétée par $[\![(\varsigma(B) = b)[\varsigma \leftarrow \tau], tr]\!] = [\![(\tau(B) = b), tr]\!] = [\![(a = b), tr]\!] = 0$.

3. La négation, la conjonction et la disjonction.
$$\begin{aligned} [\![\neg F, tr]\!] &= \begin{cases} 1 & \text{si } [\![F, tr]\!] = 0 \\ 0 & \text{sinon} \end{cases} \\ [\![F_1 \wedge F_2, tr]\!] &= [\![F_1, tr]\!] \wedge [\![F_2, tr]\!] \\ [\![F_1 \vee F_2, tr]\!] &= [\![F_1, tr]\!] \vee [\![F_2, tr]\!] \end{aligned}$$
La négation, la conjonction et la disjonction suivent l'interprétation habituelle de la logique de premier ordre.

4. La quantification universelle.
$$[\![\forall \mathcal{LS}_{i,p}.\varsigma\ F, tr]\!] = \begin{cases} 1 & \text{si } \forall(\theta, p) \in \mathcal{LS}_{i,p}(tr), \text{ nous avons } [\![F[\varsigma \leftarrow \theta], tr]\!] = 1, \\ 0 & \text{sinon.} \end{cases}$$
Nous quantifions sur les états locaux de la trace tr qui appartiennent à un rôle i se trouvant à une étape p. Plus précisément, nous quantifions sur les substitutions qui apparaissent dans ces états locaux. Après avoir remplacé le caractère générique ς par une telle substitution, la formule F est évaluée pour la trace tr. Si pour toute ces substitutions dans les états locaux de $\mathcal{LS}_{i,p}(tr)$, la formule F est vraie, alors la formule $\forall \mathcal{LS}_{i,p}.\varsigma\ F$ est vraie pour la trace tr.

5. La quantification existentielle.
$$[\![\exists \mathcal{LS}_{i,p}.\varsigma\ F, tr]\!] = \begin{cases} 1 & \text{si } \exists(\theta, p) \in \mathcal{LS}_{i,p}(tr), \text{ tel que } [\![F[\varsigma \leftarrow \theta], tr]\!] = 1, \\ 0 & \text{sinon.} \end{cases}$$
La quantification existentielle se fait d'une manière similaire à la quantification universelle. S'il y a un état local $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$, telle que la formule F est vraie pour la trace tr, après avoir remplacé les occurrence du caractère générique ς par la substitution θ, alors toute la formule $\exists \mathcal{LS}_{i,p}.\varsigma\ F$ est vraie pour la trace tr.

Les propriétés d'un protocole

Dans la définition 15, nous avons regardé les propriétés de traces comme des sous-ensembles des traces de $\mathsf{SymbTr}^\star$. Nous avons maintenant le moyen de spécifier ces sous-ensembles avec des formules de nos logiques et nous pouvons demander, si les traces valides d'un protocole satisfont une formule. Cette question donne lieu à la définition suivante.

Définition 21. Soit $\Pi : [k] \to \mathsf{Roles}^\star$ un protocole à k participants et soit $\phi \in \mathcal{L}_1^\star$ une propriété de trace. Si nous avons $[\![\phi, tr]\!] = 1$ pour toute trace $tr \in \mathsf{Exec}^\star(\Pi)$, alors le protocole Π *satisfait la propriété* ϕ. Nous notons cela aussi $\Pi \models \phi$. Nous notons $\mathsf{Tr}^\star(\phi) = \{tr \in \mathsf{SymbTr}^\star \mid [\![\phi, tr]\!] = 1\}$ l'ensemble des traces qui satisfont la formule ϕ.

Nous utilisons la notation $\mathsf{Exec}^\star(\Pi) \subseteq \mathsf{Tr}^\star(\phi)$ comme une notation équivalente à $\Pi \models \phi$.

2.5.2 Exemples de propriétés

Nous donnons deux exemples pour illustrer l'expressivité des logiques $\mathcal{L}_1^\star$.

Le secret

Soient $\Pi(1)$ et $\Pi(2)$ les deux rôles d'un protocole où les agents qui jouent les rôles sont représentés par les variables A_1, respectivement A_2. Pour cet exemple, il n'est pas important dans quel modèle nous nous situons. Supposons que le rôle $\Pi(1)$ envoie au rôle $\Pi(2)$ un message qui contient le nonce secret $X^1_{A_1}$. Nous utilisons un codage standard pour spécifier la propriété de secret. Ce codage consiste en l'ajout d'un troisième rôle $\Pi(3) = (X^1_{A_1}, \mathit{stop})$ qui peut considérer comme témoin qui écoute la communication entre les rôles $\Pi(1)$ et $\Pi(2)$. La propriété exprime le fait suivant : supposons que les agents représentés par A_1 et A_2 sont honnêtes. S'il existe une trace, telle que le nonce secret est affecté à la variable $X^1_{A_1}$ dans le rôle $\Pi(3)$, alors le protocole ne satisfait pas la propriété du secret. Dans notre formalisme :

$$\phi = \forall \mathcal{LS}_{1,1}.\varsigma\ \forall \mathcal{LS}_{3,2}.\varsigma'\ NC(\varsigma(A_1)) \wedge NC(\varsigma(A_2)) \to \neg(\varsigma'(X^1_{A_1}) = \varsigma(X^1_{A_1}))$$

Notons que la premier quantification se fait sur les états locaux du rôle $\Pi(1)$ à l'étape 1. En effet, l'étape n'est pas important, parce que le nonce $X^1_{A_1}$ qui appartient au premier rôle est initialisé tout au début. La deuxième quantification se fait sur les états locaux du rôle $\Pi(3)$ à l'étape 2, c'est-à-dire quand il a reçu un nonce et, par conséquent, terminé.

L'authentification

Nous donnons la propriété de l'authentification faible, décrite dans [Low97c]. Nous considérons un protocole à deux rôles où le premier rôle finit après n étapes et le deuxième après p étapes. Informellement, la propriété est la suivant : pour chaque instance du deuxième rôle qui se termine, jouée avec des participants honnêtes, il y a une instance du premier rôle qui s'est terminée. De plus, les deux rôles sont d'accord d'avoir parlé l'un avec l'autre et se sont mis d'accord sur une certaine valeur. Supposons que ce valeur est représenté par la variable $X^1_{A_1}$. En quantifiant sur les états appropriés, nous exprimons cette propriété dans notre formalisme par

$$\begin{aligned}\phi = \forall \mathcal{LS}_{2,p}.\varsigma\ NC(\varsigma(A_1)) \wedge NC(\varsigma(A_2)) \to \\ \exists \mathcal{LS}_{1,n}.\varsigma'\ (\varsigma(A_1) = \varsigma'(A_1)) \wedge (\varsigma(A_2) = \varsigma'(A_2)) \wedge (\varsigma(X^1_{A_1}) = \varsigma'(X^1_{A_1})).\end{aligned}$$

Chapitre 3

Mise en relation des modèles

La figure 2.1 du chapitre 2 nous montre que les outils qui existent pour la vérification automatique des protocoles cryptographiques possèdent des capacités différentes. Dans le chapitre précédent, nous avons introduit des modèles pour la spécification des protocoles qui reflètent ces différents niveaux des capacités. Étant donné qu'on n'a pas toujours accès à l'outil le plus puissant, certaines habitudes de codage se sont installées chez les vérificateurs de protocoles cryptographiques, afin de simuler les fonctionnalités manquantes. Ces habitudes sont notamment

1. l'utilisation des symboles fonctionnels déterministes au lieu des symboles probabilistes,
2. le codage du symbole de la signature par un chiffrement à clef privée et
3. la représentation des valeurs de hachage par un chiffrement asymétrique à clef publique.

Ces codages ont été utilisés jusqu'alors sans justification formelle. Il faudrait pourtant s'assurer que ces transformations préservent les propriétés d'un protocole. Pour donner un exemple concret, on peut se poser la question suivant : si on vérifie une propriété d'un protocole qui utilise une fonction de signature dans un modèle où on remplace cette fonction par le chiffrement asymétrique à clef privée, alors peut-on en déduire que la version originale du protocole satisfait la même propriété ? Nous nous intéressons donc à des *théorèmes de correction* que nous décrivons de la manière suivante :

Un protocole Π génère un ensemble de traces valides. Si nous modifions ce protocole à l'aide d'une fonction f, nous obtenons un autre ensemble de traces valides, généré cette fois-ci par le protocole $f(\Pi)$. En outre, une formule ϕ de notre logique $\mathcal{L}_1^\star$ décrit un ensemble de traces. Si nous transformons cette formule à l'aide de la fonction f, nous obtenons un ensemble de traces décrit par la formule $f(\phi)$. Nous disons que la transformation effectuée par la fonction f est *correcte*, si l'implication suivante est juste : si les traces générées par le protocole $f(\Pi)$ sont un sous-ensembles des traces décrites par $f(\phi)$, alors les traces générées par le protocole Π sont un sous-ensemble des traces décrites par ϕ. Avec le formalisme que nous venons de développer dans le chapitre précédant, le théorème de correction est donné par $f(\Pi) \models f(\phi) \implies \Pi \models \phi$.

Comme nous avons déjà vu au début du chapitre 2, les modèles, sur lesquels les outils de vérification automatique sont basés, ne mettent pas toutes les fonctionnalités souhaitables à la disposition. Si, par exemple, un vérificateur veut utiliser l'outil AVISPA pour vérifier un protocole qui a besoin des signatures, alors il est amené de modéliser cette fonctionnalité par un chiffrement à clef privée (voir figure 2.1). Un théorème, tel que nous venons de le décrire, permettrait de conclure que, si le protocole, dans lequel la signature est remplacée par un chiffrement à clef privée, satisfait une propriété, alors le protocole original qui utilise la signature satisfait aussi cette propriété. C'est dans ce contexte que notre théorème de correction est intéressant.

Nous nous intéressons aussi au théorème de correction complémentaire. Ce théorème peut être décrit par l'implication $\Pi \models \phi \implies f(\Pi) \models f(\phi)$. Ce théorème peut être intéressant dans la situation suivante : supposons que nous voulons implémenter un protocole qui a besoin de la signature, mais que la bibliothèque des fonctions cryptographiques ne contient pas cette fonction. Dans ce cas, nous pourrons utiliser le chiffrement à clef privée pour implémenter la signature. Le théorème de correction nous dit alors que les propriétés du protocole restent préservées, si nous utilisons, dans une implémentation, le chiffrement à clef privée au lieu de la signature. Cet exemple est néanmoins un peut artificiel, parce que on aurait déjà pu vérifier le protocole dans un modèle qui remplace la signature par le chiffrement à clef privée.

La façon de procéder

Nous commençons par relier les modèles que nous avons défini dans le chapitre précédant. Pour ce faire, nous définissons dans un premier temps trois fonctions :

$\frac{lhs \to hs}{\cdot}$ Cette fonction prend un protocole $\Pi : [k] \to \mathsf{Roles}^{lhs}$ et le transforme en un protocole $\Pi' : [k] \to \mathsf{Roles}^{hs}$ en remplaçant les symboles fonctionnels probabilistes par leurs pendants déterministes.

$\frac{hs \to h}{\cdot}$ Cette fonction prend un protocole $\Pi : [k] \to \mathsf{Roles}^{hs}$ et le transforme en un protocole $\Pi' : [k] \to \mathsf{Roles}^{h}$ en remplaçant le symbole de la signature par un chiffrement à clef privée.

$\frac{h \to \_}{\cdot}$ Cette fonction prend un protocole $\Pi : [k] \to \mathsf{Roles}^{h}$ et le transforme en un protocole $\Pi' : [k] \to \mathsf{Roles}$ en remplaçant les fonctions de hachage par un chiffrement asymétrique à clef publique.

Chacune de ces fonctions correspond à une des transformations que nous avons mentionnées auparavant. Nous étendons leurs définitions aussi sur les traces d'exécution et sur les formules de la logique $\mathcal{L}_1^\star$.

Dans un deuxième temps, nous essayons de prouver les théorèmes de correction pour les trois fonctions. Si cela n'est pas possible, nous donnons des contre-exemples. Nous identifions les causes de l'échec et proposons une solution.

Les résultats

Nous constatons que, pour une formule de la logique $\mathcal{L}_1^\star$, la première propriété de correspondance, à savoir $f(\Pi) \models f(\phi) \implies \Pi \models \phi$, n'est en effet juste que entre le modèle sans étiquettes, signature et hachage vers le modèle qui autorise le hachage. La cause est la suivante : étant donné que nous avons permis dans la logique $\mathcal{L}_1^\star$ des tests d'égalité entre deux termes arbitraires m_1 et m_2, l'égalité entre ces termes dans un modèle réduit, c'est-à-dire $f(m_1) = f(m_2)$, doit impliquer l'égalité dans le modèle d'origine, c'est-à-dire $m_1 = m_2$. Or, cela n'est le cas que pour la fonction $\frac{h \to \_}{\cdot}$ comme nous le montrons. Néanmoins, nous sommes obligés d'introduire une condition supplémentaire pour obtenir ce résultat. Quant à la deuxième propriété de correspondance, $\Pi \models \phi \implies f(\Pi) \models f(\phi)$, il n'est possible de la démontrer pour aucune des trois fonctions de transformations. Cela est dû au fait que nous n'avons pas l'implication $m_1 \neq m_2 \implies f(m_1) \neq f(m_2)$.

Nous concluons que la logique $\mathcal{L}_1^\star$ définit une classe de propriétés trop grande, pour que nous puissions prouver une correspondance d'un modèle à l'autre. Étant donné que nous ne voulons pas changer les définitions des fonctions, car elles correspondent aux codages que les

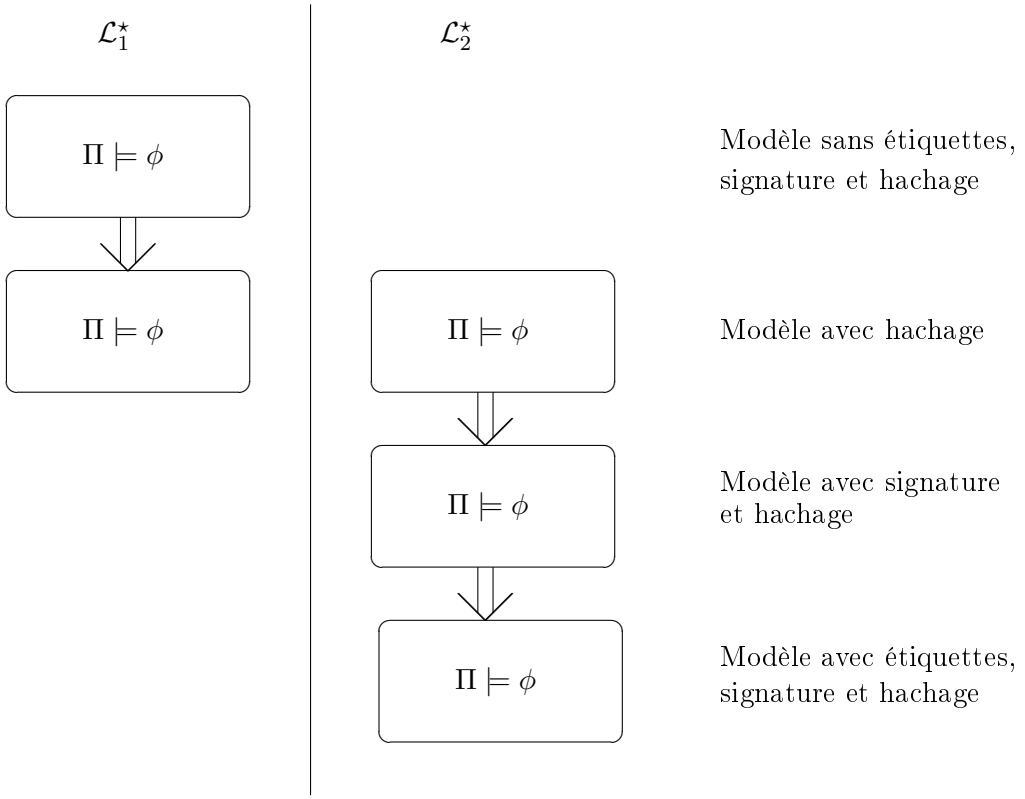


FIG. 3.1 – Les résultats de correspondance.

vérificateurs utilisent en réalité, nous identifions un fragment de la logique, appelé $\mathcal{L}_2^\star$, pour lequel nous pouvons prouver le premier théorème de correction pour toutes les trois fonctions. Comme nous avons déjà la correspondance entre le modèle sans étiquettes, signature et hachage et le modèle qui autorise le hachage, nous ne le prouvons que pour les correspondances qui manquent, à savoir, entre le modèle avec hachage et le modèle avec signature et hachage, ainsi que ce dernier et le modèle avec étiquettes, signature et hachage. En ce qui concerne la deuxième propriété de correspondance, il n'est toujours pas possible de la démontrer. Nous donnons des contre-exemples. La figure 3.1 résume les résultats obtenus.

3.1 Les fonctions

Rappel sur la notation

Nous commençons par la définition de trois fonctions. Elles sont définies à la fois sur les termes de l'algèbre $T^\star$ et de l'algèbre $T^\star_{Sub}$. Nous étendons leurs définitions sur les protocoles, les traces symboliques $\mathsf{SymbTr}^\star$ et les formules de la logique $\mathcal{L}_1^\star$. La notation de l'étoile $\star$ est utilisée comme avant.

3.1.1 Du modèle avec étiquettes vers le modèle sans étiquettes

Le modèle qui permet l'utilisation des étiquettes pour modéliser le caractère probabiliste de certaines fonctions, comme du chiffrement et de la signature, est peu répandu. À notre connais-

sance, aucun outil de la vérification automatique utilise un tel modèle. Il convient donc de définir une fonction $\overline{\cdot}^{lhs \to hs}$ qui enlève les étiquettes à tout symbole fonctionnel qui peut en révéler. Nous obtenons ainsi des protocoles situés dans un modèle plus proche à ceux qui sont utilisés en pratique. Un exemple pour un outil qui utilise un tel modèle est ProVerif [Bla01].

De $T^{lhs} \cup T^{lhs}_{Sub}$ vers $T^{hs} \cup T^{hs}_{Sub}$

La fonction $\overline{\cdot}^{lhs \to hs} : T^{lhs} \cup T^{lhs}_{Sub} \to T^{hs} \cup T^{hs}_{Sub}$ est définie récursivement sur les termes. La définition de la fonction pour les termes se trouve dans la figure 3.2. Elle remplace étape par étape les symboles fonctionnels probabilistes par leurs pendants déterministes. Les autres symboles fonctionnels ne sont pas touchés.

Extension sur les protocoles

Nous étendons cette fonction d'abord sur les rôles, puis sur des protocoles entiers : soit $\Pi(i) = (l_j, r_j)_{1 \leq j \leq n}$ le rôle d'un protocole Π. Nous définissons $\overline{\Pi(i)}^{lhs \to hs} = (\overline{l_j}^{lhs \to hs}, \overline{r_j}^{lhs \to hs})_{1 \leq j \leq n}$. Nous dénotons par $\overline{\Pi}^{lhs \to hs}$ le protocole qui est obtenu par application de la fonction $\overline{\cdot}^{lhs \to hs}$ à tous les rôles du protocole Π.

Extension sur les traces symboliques

Soit σ une substitution. Nous définissons la substitution $\overline{\sigma}^{lhs \to hs}$ que nous obtenons à partir de la substitution σ par $\overline{\sigma}^{lhs \to hs} = \{x \leftarrow \overline{m}^{lhs \to hs} \mid x \leftarrow m \in \sigma \wedge x \notin X.l\}$. Notons que le domaine de la substitution $\overline{\sigma}^{lhs \to hs}$ ne contient pas de variables du type étiquette. Quant aux autres variables du domaine de σ, il appartiennent aussi au domaine de $\overline{\sigma}^{lhs \to hs}$, mais les termes qui leur sont affectés subissent une application de la fonction $\overline{\cdot}^{lhs \to hs}$. Soit $f(\mathsf{sid}) = (\sigma, p)$ l'état local affecté à un identifiant de session $\mathsf{sid} \in \mathsf{SID}$. Nous posons $\overline{f}^{lhs \to hs}(\mathsf{sid}) = (\overline{\sigma}^{lhs \to hs}, p)$. Pour un ensemble de termes $S \subseteq T^{lhs}$, nous définissons $\overline{S}^{lhs \to hs} = \{\overline{m}^{lhs \to hs} \mid m \in S\}$.

Soit $\Pi : [k] \to \mathsf{Roles}^{lhs}$ un protocole à k participants et soit $tr \in \mathsf{Exec}^{lhs}(\Pi)$ une trace d'exécution valide. Nous définissons récursivement la trace $\overline{tr}^{lhs \to hs}$:

- si $tr = (\mathsf{SId}_1, f_1, H_1)$, alors $\overline{tr}^{lhs \to hs} = (\mathsf{SId}_1, \overline{f_1}^{lhs \to hs}, \overline{H_1}^{lhs \to hs})$
- si la dernière transition ($\mathbf{trans}_n$) est une transition **corrupt** ou **new**, alors

$$\overline{tr}^{lhs \to hs} = \overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \ldots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{trans}_n} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})}^{lhs \to hs} =$$
$$\overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \ldots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n)}^{lhs \to hs} \xrightarrow{\mathbf{trans}_n} (\mathsf{SId}_{n+1}, \overline{f_{n+1}}^{lhs \to hs}, \overline{H_{n+1}}^{lhs \to hs})$$

- si la dernière transition est une transition **send**, alors

$$\overline{tr}^{lhs \to hs} = \overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \ldots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{send}(\mathsf{sid},\mathsf{m})} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})}^{lhs \to hs} =$$
$$\overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \ldots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n)}^{lhs \to hs} \xrightarrow{\mathbf{send}(\mathsf{sid},m')} (\mathsf{SId}_{n+1}, \overline{f_{n+1}}^{lhs \to hs}, \overline{H_{n+1}}^{lhs \to hs})$$

Pour les termes de $T^{lhs} \cup T^{lhs}_{Sub}$	
$\overline{x}^{lhs\to hs} = x$	$x \in X^{hs}$
$\overline{\varsigma(x)}^{lhs\to hs} = \varsigma(x)$	$\varsigma(x) \in SubExpr(X^{hs})$
$\overline{m}^{lhs\to hs} = m$	$m = a, n(a,j,s), k(a,j,s)$ où $a \in \mathsf{ID}$ $j, s \in \mathbb{N}$
$\overline{\mathsf{k}(A)}^{lhs\to hs} = \mathsf{k}(\overline{A}^{lhs\to hs})$	$\mathsf{k} \in \{\mathsf{ek}, \mathsf{dk}, \mathsf{sk}, \mathsf{vk}\}$, $A, B \in X^{\mathsf{ID}}_a \cup SubExpr(X.a)$
$\overline{\mathsf{symk}(A,B)}^{lhs\to hs} = \mathsf{symk}(\overline{A}^{lhs\to hs}, \overline{B}^{lhs\to hs})$	$A, B \in X^{\mathsf{ID}}_a \cup SubExpr(X.a)$
$\overline{\langle m_1, m_2\rangle}^{lhs\to hs} = \langle \overline{m_1}^{lhs\to hs}, \overline{m_2}^{lhs\to hs}\rangle$	$m, m_1, m_2 \in T^{lhs} \cup T^{lhs}_{Sub}$
$\overline{\{\!|m_1|\!\}^l_{m_2}}^{lhs\to hs} = \{\!|\overline{m_1}^{lhs\to hs}|\!\}^{lhs\to hs}_{\overline{m_2}}$	
$\overline{\{m\}^l_{\mathsf{ek}(a)}}^{lhs\to hs} = \{\overline{m}^{lhs\to hs}\}^{lhs\to hs}_{\mathsf{ek}(a)}$	
$\overline{\{m\}^l_{\mathsf{dk}(a)}}^{lhs\to hs} = \{\overline{m}^{lhs\to hs}\}^{lhs\to hs}_{\mathsf{dk}(a)}$	
$\overline{[m]^l_{\mathsf{sk}(a)}}^{lhs\to hs} = [\overline{m}^{lhs\to hs}]^{lhs\to hs}_{\mathsf{sk}(a)}$	
$\overline{hash(m)}^{lhs\to hs} = hash(\overline{m}^{lhs\to hs})$	
$\overline{m}^{lhs\to hs} = m$	$m \in \{\mathsf{init}, \mathsf{stop}\}$

Pour les formules de $\mathcal{L}^{lhs}_1$	
$\overline{NC(a)}^{lhs\to hs} = NC(\overline{a}^{lhs\to hs})$	$a \in T^{lhs}_{Sub}$
$\overline{(m_1 = m_2)}^{lhs\to hs} = (\overline{m_1}^{lhs\to hs} = \overline{m_2}^{lhs\to hs})$	$m_1, m_2 \in T^{lhs}_{Sub}$
$\overline{\neg F}^{lhs\to hs} = \neg\, \overline{F}^{lhs\to hs}$	$F, F_1, F_2, \in \mathcal{L}^{lhs}_1$
$\overline{F_1 \wedge F_2}^{lhs\to hs} = \overline{F_1}^{lhs\to hs} \wedge \overline{F_2}^{lhs\to hs}$	
$\overline{F_1 \vee F_2}^{lhs\to hs} = \overline{F_1}^{lhs\to hs} \vee \overline{F_2}^{lhs\to hs}$	
$\overline{\forall \mathcal{LS}_{i,p}.\varsigma\ F}^{lhs\to hs} = \forall \mathcal{LS}_{i,p}.\varsigma\ \overline{F}^{lhs\to hs}$	
$\overline{\exists \mathcal{LS}_{i,p}.\varsigma\ F}^{lhs\to hs} = \exists \mathcal{LS}_{i,p}.\varsigma\ \overline{F}^{lhs\to hs}$	

FIG. 3.2 – La fonction $lhs \to hs$ pour les termes de $T^{lhs} \cup T^{lhs}_{Sub}$ et formules de $\mathcal{L}^{lhs}_1$.

où, pour $\mathsf{sid} = (s, j, (\dots))$, $f_n(\mathsf{sid}) = (\sigma, p)$ et $\Pi(j) = \dots (l_p, r_p) \dots$, nous posons

$$m' = \begin{cases} a & \text{pour une identité d'agent } a \in \mathsf{ID}, \text{ si le terme } l_p \text{ n'est pas défini,} \\ \overline{m}^{lhs \to hs} & \text{si } l_p\sigma \text{ et } m \text{ sont unifiables} \\ & \text{ou si } \overline{l_p\sigma}^{lhs \to hs} \text{ et } \overline{m}^{lhs \to hs} \text{ ne sont pas unifiables,} \\ a & \text{sinon, pour une identité d'agent } a \in \mathsf{ID}. \end{cases}$$

L'intuition derrière la définition du terme m' est la suivant : nous voulons montrer ultérieurement l'implication $tr \in \mathsf{Exec}^{lhs}(\Pi) \implies \overline{tr}^{lhs \to hs} \in \mathsf{Exec}^{hs}(\overline{\Pi}^{lhs \to hs})$ pour prouver notre premier théorème de correction. Pour ce faire, nous devons choisir le terme m' de façon à ce que, si m et $l_p\sigma$ ne s'unifient pas, alors m' et $\overline{l_p\sigma}^{lhs \to hs}$ ne s'unifient pas, mais si m et $l_p\sigma$ s'unifient, alors m' et $\overline{l_p\sigma}^{lhs \to hs}$ s'unifient.

Pour bien choisir m', nous testons d'abord si le terme l_p est défini. Il peut en effet ne pas être défini. Cette situation arrive, lorsque l'exécution du rôle joué dans la session sid est finie. Dans ce cas, l'entier p est plus grand que le nombre d'étapes de ce rôle. Nous choisissons $m' = a \in \mathsf{ID}$, puisque cela nous semble la façon la plus commode pour définir m', mais on pourrait choisir n'importe quel autre terme pour m'.

Après avoir traité ce cas particulier, nous devons assure que, si le terme m s'unifie avec $l_p\sigma$, alors le terme m' s'unifiera avec $\overline{l_p\sigma}^{lhs \to hs}$. En effet, le fait que $l_p\sigma$ et m sont unifiables implique que $\overline{l_p\sigma}^{lhs \to hs}$ et $\overline{m}^{lhs \to hs}$ sont unifiables. Nous prouvons cette implication dans le lemme 10. Nous choisissons donc $m' = \overline{m}^{lhs \to hs}$.

La contra-position de ce lemme dit que le fait que $\overline{l_p\sigma}^{lhs \to hs}$ et $\overline{m}^{lhs \to hs}$ ne sont pas unifiables implique que $l_p\sigma$ et m ne sont pas unifiables. Dans ce cas, nous voulons que m' ne s'unifie pas avec le terme $\overline{l_p\sigma}^{lhs \to hs}$ et nous choisissons donc $m' = \overline{m}^{lhs \to hs}$.

Finalement, si $l_p\sigma$ et m ne sont pas unifiables, mais $\overline{l_p\sigma}^{lhs \to hs}$ et $\overline{m}^{lhs \to hs}$ le sont, nous devons quand même choisir le terme m' de façon à ce qu'il ne soit pas unifiable avec $\overline{l_p\sigma}^{lhs \to hs}$. Le choix $m' = a \in \mathsf{ID}$ satisfait cette condition, comme nous verrons.

Le cas où $l_p\sigma$ et m sont unifiables, mais $\overline{l_p\sigma}^{lhs \to hs}$ et $\overline{m}^{lhs \to hs}$ ne le sont pas, n'existe pas et, par conséquent, nous ne le prenons pas en compte.

Exemple 13. Reprenons le protocole de Needham-Schroeder-Lowe [Low96] que nous avons déjà considéré dans l'exemple 7. La version déterministe $\overline{\Pi}^{lhs \to hs}$ de ce protocole est alors donnée par les deux rôles suivants :

$$\begin{aligned} \overline{\Pi(1)}^{lhs \to hs} &= (\mathsf{init}, \{X^1_{A_1}, A_1\}_{\mathsf{ek}(A_2)}), (\{X^1_{A_1}, X^1_{A_2}, A_2\}_{\mathsf{ek}(A_1)}, \{X^1_{A_2}\}_{\mathsf{ek}(A_2)}) \\ \overline{\Pi(2)}^{lhs \to hs} &= (\{X^1_{A_1}, A_1\}_{\mathsf{ek}(A_2)}, \{X^1_{A_1}, X^1_{A_2}, A_2\}_{\mathsf{ek}(A_1)}), (\{X^1_{A_2}\}_{\mathsf{ek}(A_2)}, \mathsf{stop}) \end{aligned}$$

La trace donnée dans l'exemple 9 est modifiée de la façon suivante :

$$\begin{aligned} (\emptyset, f'_1, H'_1) &\xrightarrow{\mathbf{new}(1,a_1,a_2)} (\{\mathsf{sid}_1\}, f'_2, H'_2) \\ &\xrightarrow{\mathbf{send}(\mathsf{sid}_1,\mathsf{init})} (\{\mathsf{sid}_1\}, f'_3, H'_3), \xrightarrow{\mathbf{new}(2,a_1,a_2)} (\{\mathsf{sid}_1, \mathsf{sid}_2\}, f'_4, H'_4) \\ &\xrightarrow{\mathbf{send}(\mathsf{sid}_2,\{n(a_1,1,1),a_1\}_{\mathsf{ek}(a_1)})} (\{\mathsf{sid}_1, \mathsf{sid}_2\}, f'_5, H'_5) \dots \end{aligned}$$

où $H'_1 = \overline{H_1}^{lhs\to hs}$, $H'_2 = \overline{H_2}^{lhs\to hs}$, $H'_3 = \overline{H_3}^{lhs\to hs} = H'_2 \cup \{\{n(a_1,1,1),a_1\}_{\mathsf{ek}(a_1)}\}$ et $H'_4 = \overline{H_4}^{lhs\to hs}$. Pour les quatre premiers états globaux, nous avons $f'_i = \overline{f_i}^{lhs\to hs}$.

Pour le cinquième état global, la fonction f'_5 est définie de la façon suivante : $f'_5(\mathsf{sid}_1) = \overline{f_5}^{lhs\to hs}(\mathsf{sid}_1)$. Rappelons que nous avons $f'_4(\mathsf{sid}_2) = \overline{f_4}^{lhs\to hs}(\mathsf{sid}_2) = (\overline{\sigma_2}^{lhs\to hs}, 1)$. Nous avons alors $f'_5(\mathsf{sid}_2) = \overline{f_5}^{lhs\to hs}(\mathsf{sid}_2) = (\overline{\sigma'_2}^{lhs\to hs}, 2)$ avec $\overline{\sigma'_2}^{lhs\to hs} = \overline{\sigma_2}^{lhs\to hs} \cup \{X^1_{A_1} \leftarrow n(a_1,1,1)\}$. On note que la variable L_1 n'appartient pas au domaine de la substitution $\overline{\sigma'_2}^{lhs\to hs}$.

De $\mathcal{L}_1^{lhs}$ vers $\mathcal{L}_1^{hs}$

Nous étendons l'application $\overline{\cdot}^{lhs\to hs}$ aux formules de la logique $\mathcal{L}_1^{lhs}$. Comme nous voulons que la formule $\overline{\phi}^{lhs\to hs} \in \mathcal{L}_1^{hs}$ exprime la même propriété pour un protocole $\overline{\Pi}^{lhs\to hs}$ que la formule d'origine $\phi \in \mathcal{L}_1^{lhs}$ pour le protocole Π, nous ne touchons pas à la négation, aux connecteurs, ainsi qu'aux quantificateurs. Nous passons donc la fonction $\overline{\cdot}^{lhs\to hs}$ de manière récursive jusqu'au niveau des termes où nous remplaçons les symboles fonctionnels probabilistes par des symboles déterministes. La définition de la fonction pour les formules se trouve dans la figure 3.2.

3.1.2 Du modèle avec signature vers le modèle sans signature

La fonction $\overline{\cdot}^{hs\to h}$ remplace les signatures par des chiffrements à clef privée. Ce remplacement correspond au codage habituellement utilisé dans les outils qui ne mettent pas à la disposition une fonction de signature. En effet, si on suppose que la clef privée $\mathsf{dk}(a)$ d'un agent reste secrète, alors c'est seulement lui qui peut créer un message $\{m\}_{\mathsf{dk}(a)}$. Un autre participant qui le déchiffre avec la clef publique $\mathsf{ek}(a)$ peut alors être sûr que ce message provient de l'agent a.

De $T^{hs} \cup T^{hs}_{Sub}$ vers $T^{h} \cup T^{h}_{Sub}$

La fonction $\overline{\cdot}^{hs\to h} : T^{hs} \cup T^{hs}_{Sub} \to T^{h} \cup T^{h}_{Sub}$ est injective de l'ensemble réunissant les variables de chiffrement et de signature $X.c \cup X.s$ vers l'ensemble de variables de chiffrement $X.c$. La définition de la fonction pour les termes se trouve dans la figure 3.3. Dans cette définition, un terme ne change pas, sauf s'il s'agite d'une clef de vérification ou de signature ou du symbole fonctionnel de la signature. Les clefs sont alors respectivement changé en clefs publiques et privées de chiffrement asymétrique et la signature est remplacée par un chiffrement à clef privée.

Extension sur les protocoles

Nous étendons la projection $\overline{\cdot}^{hs\to h}$ d'abord sur les rôles, puis sur des protocoles entiers : soit $\Pi(i) = (l_j, r_j)_{1\le j\le n}$ le rôle d'un protocole Π. Nous définissons $\overline{\Pi(i)}^{hs\to h} = (\overline{l_j}^{hs\to h}, \overline{r_j}^{hs\to h})_{1\le j\le n}$. Nous dénotons par $\overline{\Pi}^{hs\to h}$ le protocole qui est obtenu par application de la fonction $\overline{\cdot}^{hs\to h}$ à tous les rôles du protocole Π.

Extension sur les traces symboliques

Soit σ une substitution. Nous définissons la substitution $\overline{\sigma}^{hs\to h}$ par $\overline{\sigma}^{hs\to h} = \{\overline{x}^{hs\to h} \leftarrow \overline{m}^{hs\to h} \mid x \leftarrow m \in \sigma\}$. Soit $f(\mathsf{sid}) = (\sigma, p)$ l'état local affecté à un identifiant de session $\mathsf{sid} \in \mathsf{SID}$. Nous posons $\overline{f}^{hs\to h}(\mathsf{sid}) = (\overline{\sigma}^{hs\to h}, p)$. Pour un ensemble de termes $S \subseteq T^{hs}$, nous définissons $\overline{S}^{hs\to h} =$

Pour les termes de $T^{hs} \cup T^{hs}_{Sub}$	
$\overline{x}^{hs\to h} = x$	$x \in X^h$
$\overline{x}^{hs\to h} = y$	$x \in X.s$, $y \in X.c$
$\overline{\varsigma(x)}^{hs\to h} = \varsigma(x)$	$\varsigma(x) \in SubExpr(X^h)$
$\overline{\varsigma(x)}^{hs\to h} = \varsigma(y)$	$\varsigma(x) \in SubExpr(X.s)$, tel que $\overline{x}^{hs\to h} = y$
$\overline{m}^{hs\to h} = m$	$m = a, n(a,j,s), k(a,j,s)$ où $a \in \mathsf{ID}$, $j, s \in \mathbb{N}$
$\overline{\mathsf{k}(A)}^{hs\to h} = \mathsf{k}(\overline{A}^{hs\to h})$	$\mathsf{k} \in \{\mathsf{ek}, \mathsf{dk}\}$, $A \in X^{\mathsf{ID}}_a \cup SubExpr(X.a)$
$\overline{\mathsf{vk}(A)}^{hs\to h} = \mathsf{ek}(\overline{A}^{hs\to h})$	$A \in X^{\mathsf{ID}}_a \cup SubExpr(X.a)$
$\overline{\mathsf{sk}(A)}^{hs\to h} = \mathsf{dk}(\overline{A}^{hs\to h})$	
$\overline{\mathsf{symk}(A,B)}^{hs\to h} = \mathsf{symk}(\overline{A}^{hs\to h}, \overline{B}^{hs\to h})$	$A, B \in X^{\mathsf{ID}}_a \cup SubExpr(X.a)$
$\overline{\langle m_1, m_2\rangle}^{hs\to h} = \langle \overline{m_1}^{hs\to h}, \overline{m_2}^{hs\to h}\rangle$	$m, m_1, m_2 \in T^{hs} \cup T^{hs}_{Sub}$
$\overline{\{\!|m_1|\!\}_{m_2}}^{hs\to h} = \{\!|\overline{m_1}^{hs\to h}|\!\}_{\overline{m_2}^{hs\to h}}$	
$\overline{\{m\}_{\mathsf{ek}(a)}}^{hs\to h} = \{\overline{m}^{hs\to h}\}_{\overline{\mathsf{ek}(a)}^{hs\to h}}$	
$\overline{\{m\}_{\mathsf{dk}(a)}}^{hs\to h} = \{\overline{m}^{hs\to h}\}_{\overline{\mathsf{dk}(a)}^{hs\to h}}$	
$\overline{[m]_{\mathsf{sk}(a)}}^{hs\to h} = \{\overline{m}^{hs\to h}\}_{\overline{\mathsf{sk}(a)}^{hs\to h}}$	
$\overline{hash(m)}^{hs\to h} = hash(\overline{m}^{hs\to h})$.	
$\overline{m}^{hs\to h} = m$	$m \in \{\mathsf{init}, \mathsf{stop}\}$

Pour les formules de $\mathcal{L}^{hs}_1$	
$\overline{NC(a)}^{hs\to h} = NC(\overline{a}^{hs\to h})$	$a \in T^{hs}_{Sub}$
$\overline{(m_1 = m_2)}^{hs\to h} = (\overline{m_1}^{hs\to h} = \overline{m_2}^{hs\to h})$	$m_1, m_2 \in T^{hs}_{Sub}$
$\overline{\neg F}^{hs\to h} = \neg\, \overline{F}^{hs\to h}$	$F, F_1, F_2, \in \mathcal{L}^{hs}_1$
$\overline{F_1 \wedge F_2}^{hs\to h} = \overline{F_1}^{hs\to h} \wedge \overline{F_2}^{hs\to h}$	
$\overline{F_1 \vee F_2}^{hs\to h} = \overline{F_1}^{hs\to h} \vee \overline{F_2}^{hs\to h}$	
$\overline{\forall \mathcal{LS}_{i,p}.\varsigma\ F}^{hs\to h} = \forall \mathcal{LS}_{i,p}.\varsigma\ \overline{F}^{hs\to h}$	
$\overline{\exists \mathcal{LS}_{i,p}.\varsigma\ F}^{hs\to h} = \exists \mathcal{LS}_{i,p}.\varsigma\ \overline{F}^{hs\to h}$	

FIG. 3.3 – La fonction $hs \to h$ pour les termes de $T^{hs} \cup T^{hs}_{Sub}$ et formules de $\mathcal{L}^{hs}_1$.

$\{\overline{m}^{hs\to h} \mid m \in S\}$. Soit $\Pi : [k] \to \mathsf{Roles}^{hs}$ un protocole et $tr \in \mathsf{Exec}^{hs}(\Pi)$ une trace d'exécution. Nous définissons récursivement la trace $\overline{tr}^{hs\to h}$ par :

- si $tr = (\mathsf{SId}_1, f_1, H_1)$, alors $\overline{tr}^{hs\to h} = (\mathsf{SId}_1, \overline{f_1}^{hs\to h}, \overline{H_1}^{hs\to h})$
- si la dernière transition ($\mathbf{trans}_n$) est une transition **corrupt** ou **new**, alors

$$\overline{tr}^{hs\to h} = \overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \dots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{trans}_n} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})}^{hs\to h} =$$
$$\overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \dots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n)}^{hs\to h} \xrightarrow{\mathbf{trans}_n} (\mathsf{SId}_{n+1}, \overline{f_{n+1}}^{hs\to h}, \overline{H_{n+1}}^{hs\to h})$$

- si la dernière transition est une transition **send**, alors

$$\overline{tr}^{hs\to h} = \overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \dots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{send}(\mathsf{sid},\mathsf{m})} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})}^{hs\to h} =$$
$$\overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \dots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n)}^{hs\to h} \xrightarrow{\mathbf{send}(\mathsf{sid},m')} (\mathsf{SId}_{n+1}, \overline{f_{n+1}}^{hs\to h}, \overline{H_{n+1}}^{hs\to h})$$

où, pour $\mathsf{sid} = (s, j, (\dots))$, $f_n(\mathsf{sid}) = (\sigma, p)$ et $\Pi(j) = \dots (l_p, r_p) \dots$, nous posons

$$m' = \begin{cases} a & \text{pour une identité d'agent } a \in \mathsf{ID}, \text{ si le terme } l_p \text{ n'est pas défini,} \\ \overline{m}^{hs\to h} & \text{si } l_p\sigma \text{ et } m \text{ sont unifiables} \\ & \text{ou si } \overline{l_p\sigma}^{hs\to h} \text{ et } \overline{m}^{hs\to h} \text{ ne sont pas unifiables,} \\ a & \text{sinon, pour une identité d'agent } a \in \mathsf{ID}. \end{cases}$$

Les raisons pour le choix du terme m' sont les mêmes que celles que nous avons mentionnées dans la définition de la fonction $\overline{\cdot}^{lhs\to hs}$.

De $\mathcal{L}_1^{hs}$ vers $\mathcal{L}_1^h$

Nous étendons l'application $\overline{\cdot}^{hs\to h}$ aux formules de la logique $\mathcal{L}_1^{hs}$. La définition de la fonction pour les formules se trouve dans la figure 3.3.

3.1.3 Du modèle avec hachage vers le modèle sans hachage

L'application $\overline{\cdot}^{h\to}$ remplace la fonction de hachage par un chiffrement avec une clef publique $\mathsf{ek}(h)$ où $h \in \mathsf{ID}$ est une identité d'agent. Cela est un codage usuel dans les modèles qui ne contiennent pas de fonction de hachage et considéré comme équivalent à une fonction de hachage, tant que l'agent h ne joue pas de rôle ni n'est corrompu par l'intrus. Cette hypothèse implique en effet qu'aucun participant ne connaît la clef privée $\mathsf{dk}(h)$ et, dans ce cas, un terme $\{m\}_{\mathsf{ek}(h)}$ ne révèle pas plus d'information que le terme $hash(m)$.

De $T^h \cup T^h_{Sub}$ vers $T \cup T_{Sub}$

La fonction $\overline{\cdot}^{h\to} : T^h \cup T^h_{Sub} \to T \cup T_{Sub}$ est injective de l'ensemble $X.c \cup X.h$ qui réunit les variables de chiffrement et de hachage vers l'ensemble de variables de chiffrement $X.c$. La définition de cette fonction pour les termes se trouve dans la figure 3.4. Dans cette définition, l'identité d'agent $h \in \mathsf{ID}$ est fixée. On remplace donc le symbole fonctionnel de hachage par un chiffrement avec la clef publique $\mathsf{ek}(h)$.

Pour les termes de $T^h \cup T^h_{Sub}$	
$\overline{x}^{h\rightarrow\_} = x$	$x \in X$
$\overline{x}^{h\rightarrow\_} = y$	$x \in X.h$, $y \in X.c$
$\overline{\varsigma(x)}^{h\rightarrow\_} = \varsigma(x)$	$\varsigma(x) \in SubExpr(X)$
$\overline{\varsigma(x)}^{h\rightarrow\_} = \varsigma(y)$	$\varsigma(x) \in SubExpr(X.h)$, tel que $\overline{x}^{h\rightarrow\_} = y$
$\overline{m}^{h\rightarrow\_} = m$	$m = a, n(a,j,s), k(a,j,s)$ où $a \in \mathsf{ID}$, $j,s \in \mathbb{N}$
$\overline{\mathsf{k}(A)}^{h\rightarrow\_} = \mathsf{k}(\overline{A}^{h\rightarrow\_})$	$\mathsf{k} \in \{\mathsf{ek}, \mathsf{dk}\}$, $A \in X^{\mathsf{ID}}_a \cup SubExpr(X.a)$
$\overline{\mathsf{symk}(A,B)}^{h\rightarrow\_} = \mathsf{ek}(\overline{A}^{h\rightarrow\_}, \overline{B}^{h\rightarrow\_})$	$A, B \in X^{\mathsf{ID}}_a \cup SubExpr(X.a)$
$\overline{\langle m_1, m_2\rangle}^{h\rightarrow\_} = \langle \overline{m_1}^{h\rightarrow\_}, \overline{m_2}^{h\rightarrow\_}\rangle$	$m, m_1, m_2 \in T^h \cup T^h_{Sub}$, $h \in \mathsf{ID}$
$\overline{\{\!|m_1|\!\}_{m_2}}^{h\rightarrow\_} = \{\!|\overline{m_1}^{h\rightarrow\_}|\!\}_{\overline{m_2}^{h\rightarrow\_}}$	
$\overline{\{m\}_{\mathsf{ek}(a)}}^{h\rightarrow\_} = \{\overline{m}^{h\rightarrow\_}\}_{\overline{\mathsf{ek}(a)}^{h\rightarrow\_}}$	
$\overline{\{m\}_{\mathsf{dk}(a)}}^{h\rightarrow\_} = \{\overline{m}^{h\rightarrow\_}\}_{\overline{\mathsf{dk}(a)}^{h\rightarrow\_}}$	
$\overline{hash(m)}^{h\rightarrow\_} = \{\overline{m}^{h\rightarrow\_}\}_{\overline{\mathsf{ek}(h)}^{h\rightarrow\_}}$	
$\overline{m}^{h\rightarrow\_} = m$	$m \in \{\mathsf{init}, \mathsf{stop}\}$

Pour les formules de $\mathcal{L}^h_1$	
$\overline{NC(a)}^{h\rightarrow\_} = NC(\overline{a}^{h\rightarrow\_})$	$a \in T^h_{Sub}$
$\overline{(m_1 = m_2)}^{h\rightarrow\_} = (\overline{m_1}^{h\rightarrow\_} = \overline{m_2}^{h\rightarrow\_})$	$m_1, m_2 \in T^h_{Sub}$
$\overline{\neg F}^{h\rightarrow\_} = \neg\overline{F}^{h\rightarrow\_}$	$F, F_1, F_2, \in \mathcal{L}^h_1$
$\overline{F_1 \wedge F_2}^{h\rightarrow\_} = \overline{F_1}^{h\rightarrow\_} \wedge \overline{F_2}^{h\rightarrow\_}$	
$\overline{F_1 \vee F_2}^{h\rightarrow\_} = \overline{F_1}^{h\rightarrow\_} \vee \overline{F_2}^{h\rightarrow\_}$	
$\overline{\forall\mathcal{LS}_{i,p}.\varsigma\ F}^{h\rightarrow\_} = \forall\mathcal{LS}_{i,p}.\varsigma\ \overline{F}^{h\rightarrow\_}$	
$\overline{\exists\mathcal{LS}_{i,p}.\varsigma\ F}^{h\rightarrow\_} = \exists\mathcal{LS}_{i,p}.\varsigma\ \overline{F}^{h\rightarrow\_}$	

FIG. 3.4 – La fonction $h \rightarrow \_$ pour les termes de $T^h \cup T^h_{Sub}$ et formules de $\mathcal{L}^h_1$.

Extension sur les protocoles

Nous étendons la fonction $\overline{\cdot}^{h\rightarrow\_}$ d'abord sur les rôles, puis sur des protocoles entiers : soit $\Pi(i) = (l_j, r_j)_{1\leq j\leq n}$ le rôle d'un protocole Π. Nous définissons $\overline{\Pi(i)}^{h\rightarrow\_} = (\overline{l_j}^{h\rightarrow\_}, \overline{r_j}^{h\rightarrow\_})_{1\leq j\leq n}$. Nous dénotons par $\overline{\Pi}^{h\rightarrow\_}$ le protocole qui est obtenu par application de la fonction $\overline{\cdot}^{h\rightarrow\_}$ à tous les rôles du protocole Π.

Extension sur les traces symboliques

Soit σ une substitution. Nous définissons la substitution $\overline{\sigma}^{h\rightarrow\_}$ par $\overline{\sigma}^{h\rightarrow\_} = \{\overline{x}^{h\rightarrow\_} \leftarrow \overline{m}^{h\rightarrow\_} \mid x \leftarrow m \in \sigma\}$. Soit $f(\mathsf{sid}) = (\sigma, p)$ l'état local affecté à un identifiant de session $\mathsf{sid} \in \mathsf{SID}$. Nous posons $\overline{f}^{h\rightarrow\_}(\mathsf{sid}) = (\overline{\sigma}^{h\rightarrow\_}, p)$. Pour un ensemble de termes $S \subseteq T^h$, nous définissons $\overline{S}^{h\rightarrow\_} = \{\overline{m}^{h\rightarrow\_} \mid m \in S\}$. Remarquons particulièrement que $\overline{\mathbf{kn}^h(a)}^{h\rightarrow\_} = \mathbf{kn}(a)$ pour une identité d'agent $a \in \mathsf{ID}$. Soit $\Pi : [k] \rightarrow \mathsf{Roles}^h$ un protocole et $tr \in \mathsf{Exec}^h(\Pi)$ une trace d'exécution. Nous définissons récursivement la trace $\overline{tr}^{h\rightarrow\_}$:

- si $tr = (\mathsf{SId}_1, f_1, H_1)$, alors $\overline{tr}^{h\rightarrow\_} = (\mathsf{SId}_1, \overline{f_1}^{h\rightarrow\_}, \overline{H_1}^{h\rightarrow\_})$
- si la dernière transition ($\mathbf{trans}_n$) est une transition **corrupt** ou **new**, alors

$$\overline{tr}^{h\rightarrow\_} = \overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \dots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{trans}_n} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})}^{h\rightarrow\_} =$$
$$\overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \dots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n)}^{h\rightarrow\_} \xrightarrow{\mathbf{trans}_n} (\mathsf{SId}_{n+1}, \overline{f_{n+1}}^{h\rightarrow\_}, \overline{H_{n+1}}^{h\rightarrow\_})$$

- si la dernière transition est une transition **send**, alors

$$\overline{tr}^{h\rightarrow\_} = \overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \dots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{send}(\mathsf{sid},\mathsf{m})} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})}^{h\rightarrow\_} =$$
$$\overline{(\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{trans}_1} \dots \xrightarrow{\mathbf{trans}_{n-1}} (\mathsf{SId}_n, f_n, H_n)}^{h\rightarrow\_} \xrightarrow{\mathbf{send}(\mathsf{sid},m')} (\mathsf{SId}_{n+1}, \overline{f_{n+1}}^{h\rightarrow\_}, \overline{H_{n+1}}^{h\rightarrow\_})$$

où, pour $\mathsf{sid} = (s, j, (\dots))$, $f_n(\mathsf{sid}) = (\sigma, p)$ et $\Pi(j) = \dots (l_p, r_p) \dots$, nous posons

$$m' = \begin{cases} a & \text{pour une identité d'agent } a \in \mathsf{ID}, \text{ si le terme } l_p \text{ n'est pas défini,} \\ \overline{m}^{h\rightarrow\_} & \text{si } l_p\sigma \text{ et } m \text{ sont unifiables} \\ & \text{ou si } \overline{l_p\sigma}^{h\rightarrow\_} \text{ et } \overline{m}^{h\rightarrow\_} \text{ ne sont pas unifiables,} \\ a & \text{sinon, pour une identité d'agent } a \in \mathsf{ID}. \end{cases}$$

Comme pour la fonction $\overline{\cdot}^{hs\rightarrow h}$, les raisons pour le choix du terme m' sont les mêmes que celles que nous avons évoquées dans la définition de la fonction $\overline{\cdot}^{lhs\rightarrow hs}$.

De $\mathcal{L}_1^h$ vers $\mathcal{L}_1$

Nous étendons l'application $\overline{\cdot}^{h\rightarrow\_}$ aux formules de la logique $\mathcal{L}_1^h$. La définition de la fonction pour les formules se trouve dans la figure 3.4.

3.2 Les théorèmes de correction pour les logiques $\mathcal{L}_1^\star$

3.2.1 Énoncés des propriétés souhaitées

Nous donnons maintenant l'énoncé formel des propriétés que nous souhaitons prouver. Pour nous faciliter la tâche, nous élargissons notre convention de notation : nous notons $f^\star$ une des fonctions $\overline{\cdot}^{lhs\to hs}$, $\overline{\cdot}^{hs\to h}$ ou $\overline{\cdot}^{h\to}$, selon le contexte.

Propriété 1. *Soit* $\Pi : [k] \to \mathsf{Roles}^\star$ *un protocole à* k *participants et soit* $\phi \in \mathcal{L}_1^\star$ *un propriété de trace. Nous avons* $f^\star(\Pi) \models f^\star(\phi) \implies \Pi \models \phi$.

Nous donnons aussi l'énoncé de la propriété complémentaire.

Propriété 2. *Soit* $\Pi : [k] \to \mathsf{Roles}^\star$ *un protocole à* k *participants et soit* $\phi \in \mathcal{L}_1^\star$ *un propriété de trace. Nous avons* $\Pi \models \phi \implies f^\star(\Pi) \models f^\star(\phi)$.

3.2.2 Le cas de la fonction $lhs \to hs$

La fonction $\overline{\cdot}^{lhs\to hs}$ ne satisfait pas les deux propriétés énoncées, c'est-à-dire que nous n'avons pas $\overline{\Pi}^{lhs\to hs} \models \overline{\phi}^{lhs\to hs} \implies \Pi \models \phi$, ni l'implication inverse. Nous donnons deux contre-exemples.

Contre-exemple pour la propriété 1

Considérons la première étape d'un protocole $\Pi : [3] \to \mathsf{Roles}^{lhs}$ où le premier rôle envoie un message au deuxième rôle. Une partie de ce message est destinée à être envoyée à un troisième rôle que nous ne spécifions pas explicitement, puisqu'il n'importe pas pour nos fins. Informellement, le protocole est donné par

$$A \quad \to B : \quad \{N_a, \{N_a\}_{\mathsf{ek}(C)}, \{N_a\}_{\mathsf{ek}(C)}\}_{\mathsf{ek}(B)}$$

Utilisant des termes dans T^{lhs}, les rôles A et B qui correspond à cette étape du protocole peuvent être spécifiés de manière suivante :

$$\begin{aligned} \Pi(1) &= (\mathsf{init}, \{\langle X^1_{A_1}, \langle \{X^1_{A_1}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_3)}, \{X^1_{A_1}\}^{\mathsf{ag}(2)}_{\mathsf{ek}(A_3)}\rangle\rangle\}^{\mathsf{ag}(3)}_{\mathsf{ek}(A_2)}) \\ \Pi(2) &= (\{\langle X^1_{A_1}, \langle C_1, C_2\rangle\rangle\}^{L}_{\mathsf{ek}(A_2)}, \mathsf{stop}) \end{aligned}$$

où nous notons par C_1 et $C_2 \in X.c$ deux variables du type chiffrement. Nous supposons dans cette spécification que le premier rôle exécute deux fois l'algorithme qui chiffre le nonce représenté par la variable $X^1_{A_1}$, parce que deux étiquettes différentes $\mathsf{ag}(1)$ et $\mathsf{ag}(2)$ ont été utilisées. Notons que le deuxième rôle s'arrête après avoir reçu un message, parce que cela suffit pour notre but, mais il pourrait continuer pour construire un exemple plus réaliste.

Nous considérons maintenant une propriété de trace qui dit que, si les deux rôles sont d'accord sur la valeur de nonce qui est affectée à la variable $X^1_{A_1}$, alors le deuxième rôle doit avoir reçu deux fois le même chiffrement du nonce $X^1_{A_1}$, ce qui est impossible dans une exécution normale, parce que le premier rôle utilise toujours des étiquettes différents :

$$\begin{aligned} \phi = \forall \mathcal{LS}_{1,2}.\varsigma\ \forall \mathcal{LS}_{2,2}.\varsigma' \\ NC(\varsigma(A_1)) \wedge NC(\varsigma(A_2)) \wedge (\varsigma(X^1_{A_1}) = \varsigma'(X^1_{A_1})) \to (\varsigma'(C_1) = \varsigma'(C_2)) \end{aligned} \tag{3.1}$$

Nous avons donc $\Pi \not\models \phi$. D'autre côté, il semble clair que $\overline{\Pi}^{lhs\to hs} \models \overline{\phi}^{lhs\to hs}$, bien que nous ne le démontrions pas formellement. L'argument intuitif est le suivant : les deux chiffrements du nonce $X^1_{A_1}$ sont égaux après avoir enlevé les étiquettes. Étant donné que la propriété ϕ ne concerne que des traces où les agents A_1 et A_2 sont honnêtes et d'accord sur la valeur de la nonce $X^1_{A_1}$, le message reçu par le rôle $\Pi(2)$ doit être sûrement émis par le rôle $\Pi(1)$, parce que l'adversaire ne peut pas connaître la valeur de $X^1_{A_1}$. Par conséquent, le deuxième rôle devrait avoir reçu deux fois le même chiffrement de $X^1_{A_1}$.

La propriété 1 n'est donc pas satisfaite pour la fonction $\overline{\cdot}^{lhs\to hs}$.

Contre-exemple pour la propriété 2

Pendant l'exécution du protocole que nous considérons dans ce contre-exemple, le premier rôle envoie un nonce deux fois chiffré avec sa clef publique en utilisant des étiquettes différentes $\mathsf{ag}(1)$ et $\mathsf{ag}(2)$. Avant d'envoyer les deux chiffrements, il les chiffre encore une fois, cette fois-ci avec la clef publique du deuxième rôle, et signe le tout avec sa clef de signature. Le deuxième rôle attend un message de cette forme et stocke les deux chiffrements du nonce dans des variables $C_1, C_2 \in X.c$. Les deux rôles du protocole $\Pi : [2] \to \mathsf{Roles}^{lhs}$ sont donnés par :

$$\begin{aligned}\Pi(1) &= (\mathsf{init}, [\,\{\{X^1_{A_1}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_1)}, \{X^1_{A_1}\}^{\mathsf{ag}(2)}_{\mathsf{ek}(A_1)}\}^{\mathsf{ag}(3)}_{\mathsf{ek}(A_2)}\,]^{\mathsf{ag}(4)}_{\mathsf{sk}(A_1)})\\ \Pi(2) &= ([\{C_1, C_2\}^{L_1}_{\mathsf{ek}(A_2)}]^{L_2}_{\mathsf{sk}(A_1)}, \mathsf{stop}).\end{aligned}$$

Ce protocole est un peu artificiel, mais il nous permet de montrer que la propriété 2 n'est pas satisfaite pour la fonction $\overline{\cdot}^{lhs\to hs}$. Considérons la formule ϕ donnée par :

$$\phi = \forall \mathcal{LS}_{2,2}.\varsigma\ (NC(\varsigma(A_2)) \wedge NC(\varsigma(A_1))) \to \neg(\varsigma(C_1) = \varsigma(C_2)).$$

Nous considérons toutes les session du deuxième rôle pour lesquelles l'exécution s'est terminée. La propriété dit que, si dans ces sessions l'agent qui joue le deuxième rôle n'est pas corrompu et parle à un agent non corrompu, alors les valeurs affectées aux variables C_1 et C_2 ne sont pas égales. Le fait que l'agent représenté par la variable A_1 n'est pas corrompu assure que le message reçu par le deuxième rôle provient réellement du premier rôle, parce que l'intrus ne connaît pas la clef de signature $\mathsf{sk}(A_1)$. Dans ce cas, les deux chiffrements $\{X^1_{A_1}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_1)}$ et $\{X^1_{A_1}\}^{\mathsf{ag}(2)}_{\mathsf{ek}(A_1)}$ sont en effet différents, car les étiquettes utilisées diffèrent. Nous avons donc $\Pi \models \phi$. Si nous ôtons ces étiquettes avec la fonction $\overline{\cdot}^{lhs\to hs}$, les deux chiffrements deviennent égaux. Nous avons alors $\overline{\Pi}^{lhs\to hs} \not\models \overline{\phi}^{lhs\to hs}$.

Par conséquent, la propriété 2 est fausse pour la fonction $\overline{\cdot}^{lhs\to hs}$.

3.2.3 Le cas de la fonction $hs \to h$

Comme pour la fonction $\overline{\cdot}^{lhs\to hs}$ les propriétés de correction 1 et 2 ne sont pas satisfaites pour la fonction $\overline{\cdot}^{hs\to h}$. Voici les contre-exemples :

Contre-exemple pour la propriété 1

Nous considérons le protocole $\Pi : [2] \to \mathsf{Roles}^{hs}$, spécifié par les deux rôles

$$\begin{aligned}\Pi(1) &= (\mathsf{init}, \langle \mathsf{sk}(A_1), \{X^1_{A_1}\}_{\mathsf{ek}(A_1)}\rangle)\\ \Pi(2) &= (X^2_{A_1}, \mathsf{stop}).\end{aligned}$$

Le premier rôle envoie une paire qui consiste, d'une part, en sa clef de signature et, d'autre part, en un nonce chiffré avec sa propre clef publique. Le deuxième rôle attend un nonce. Notons que nous avons utilisé une autre variable pour dénoter le nonce attendu. Cela nous semble plus clair et facilite la discussion. Il n'y pas d'exécution raisonnable de ce protocole : il a besoin d'un intrus pour s'exécuter. L'intrus doit, par exemple, modifier le message émis par le premier rôle, pour qu'il soit accepté par le deuxième. Nous considérons la propriété suivant :

$$\phi = \exists \mathcal{LS}_{1,2}.\varsigma\ \exists \mathcal{LS}_{2,2}.\varsigma'\ NC(\varsigma(A_1)) \wedge (\varsigma(X^1_{A_1}) = \varsigma'(X^2_{A_1})).$$

Elle dit qu'il existe une session du premier rôle, telle que l'agent qui le joue n'est pas corrompu, et qu'il existe une session du deuxième rôle, telle que l'agent qui joue ce rôle est d'accord avec l'agent du premier rôle sur une valeur de nonce.

Considérons maintenant le protocole $\overline{\Pi}^{hs\to h}$:

$$\begin{aligned}\overline{\Pi(1)}^{hs\to h} &= (\mathsf{init}, \langle \mathsf{dk}(A_1), \{X^1_{A_1}\}_{\mathsf{ek}(A_1)}\rangle)\\ \overline{\Pi(2)}^{hs\to h} &= (X^2_{A_1}, \mathsf{stop}).\end{aligned}$$

La propriété $\overline{\phi}^{hs\to h}$ est intuitivement satisfait par ce protocole, parce qu'un agent qui joue le premier rôle, quoique non corrompu, révèle sa clef privée $\mathsf{dk}(A_1)$. L'intrus peut donc déduire la valeur affectée à la variable $X^1_{A_1}$ et l'envoie à un participant qui joue le deuxième rôle. Nous avons alors $\overline{\Pi}^{hs\to h} \models \overline{\phi}^{hs\to h}$.

D'autre côté, le protocole Π ne satisfait pas la propriété ϕ, parce qu'un agent non corrompu qui joue le premier rôle dans ce protocole ne révèle plus sa clef privée $\mathsf{dk}(A_1)$. Par conséquent, il est impossible pour un intrus de déduire la valeur de la variable $X^1_{A_1}$. Il n'existe donc aucune session pour le deuxième rôle où la variable $X^2_{A_1}$ prend cette valeur. Nous avons donc $\Pi \not\models \phi$.

Nous concluons que la propriété 1 n'est pas satisfaite dans le cas de la fonction $\overline{\cdot}^{hs\to h}$.

Contre-exemple pour la propriété 2

Prenons encore le protocole $\Pi : [2] \to \mathsf{Roles}^{hs}$ du paragraphe précédant :

$$\begin{aligned}\Pi(1) &= (\mathsf{init}, \langle \mathsf{sk}(A_1), \{X^1_{A_1}\}_{\mathsf{ek}(A_1)}\rangle)\\ \Pi(2) &= (X^2_{A_1}, \mathsf{stop}).\end{aligned}$$

Considérons une propriété qui dit que, tant que le participant qui joue le premier rôle n'est pas corrompu, le participant qui joue le deuxième rôle ne connaîtra pas la valeur affectée à la variable $X^1_{A_1}$:

$$\phi = \forall \mathcal{LS}_{1,2}.\varsigma\ \forall \mathcal{LS}_{2,2}.\varsigma'\ NC(\varsigma(A_1)) \to \neg(\varsigma(X^1_{A_1}) = \varsigma'(X^2_{A_1}))$$

Il semble intuitivement clair que le protocole satisfait cette propriété, car l'adversaire ne connaît pas la clef privée $\mathsf{dk}(A_1)$ de l'agent qui joue le premier rôle, parce que nous ne considérons que les session où il est non corrompu. Pourtant cette clef est nécessaire pour déduire la valeur prise par la variable $X^1_{A_1}$. Nous concluons donc que nous avons $\Pi \models \phi$.

Considérons maintenant encore une fois le protocole $\overline{\Pi}^{hs\to h}$. Bien qu'un agent qui joue le premier rôle ne puisse être corrompu, l'intrus peut déduire la clef $\mathsf{dk}(A_1)$ à partir du premier message

émis par ce rôle. Il peut ensuite déduire la valeur prise par la variable $X^1_{A_1}$, ce qui lui permet de l'envoyer à un participant qui joue le deuxième rôle. Dans ce cas, la propriété $\overline{\phi}^{hs\to h}$ n'est pas satisfaite. Par conséquent, nous avons $\overline{\Pi}^{hs\to h} \not\models \overline{\phi}^{hs\to h}$.

Cet exemple montre que la propriété 2 n'est pas vraie pour la fonction $\overline{\cdot}^{hs\to h}$.

3.2.4 Le cas de la fonction $h \to \_$

Contrairement aux autres fonctions, la fonction $\overline{\cdot}^{h\to\_}$ satisfait la propriété 1, si nous respectons quelques conditions. Nous éclaircissons d'abord les conditions nécessaires et faisons les respecter par un changement léger de notre formalisme. Ensuite, nous prouvons que la propriété est satisfaite.

Quant à la propriété 2, elle est néanmoins fausse pour cette fonction et nous donnons un contre-exemple.

Les conditions nécessaires pour prouver la justesse de la propriété 1

Au cours de la preuve de la propriété 1, nous avons besoin de montrer que $[\![\overline{\phi}^{h\to\_}, \overline{tr}^{h\to\_}]\!] = 1$ implique $[\![\phi, tr]\!] = 1$. Cela est le contenu du lemme 2 que nous énonçons ultérieurement. Il y a deux cas, dans lesquels l'implication désirée n'est pas juste. Nous considérons les deux cas pour en trouver la raison :

1. Les formules de la logique $\mathcal{L}_1^h$ peuvent contenir des termes dans lesquels le sous-terme $h \in \mathsf{ID}$ apparaît. Considérons la formule $\phi = (\varsigma(H) = \{a\}_{\mathsf{ek}(h)})$ où $a \in \mathsf{ID}$ est une identité d'agent et $H \in X.h$ une variable du type hachage. Supposons que nous avons $\overline{H}^{h\to\_} = C$ pour une variables $C \in X.c$. Considérons l'interprétation de cette formule où la variable de substitution $\varsigma \in Sub$ prend la valeur $\sigma = \{H \leftarrow hash(a)\}$. Nous interprétons la formule ϕ pour une trace $tr \in \mathsf{SymbTr}^h$:

$$[\![(\varsigma(H) = \{a\}_{\mathsf{ek}(h)})[\varsigma \leftarrow \sigma], tr]\!] = [\![(\sigma(H) = \{a\}_{\mathsf{ek}(h)}), tr]\!] = [\![(hash(a) = \{a\}_{\mathsf{ek}(h)}), tr]\!] = 0.$$

   Considérons d'autre part l'interprétation de la formule $\overline{\phi}^{h\to\_} = (\varsigma(C) = \{a\}_{\mathsf{ek}(h)})$ pour la trace $\overline{tr}^{h\to\_}$:

$$[\![(\varsigma(C) = \{a\}_{\mathsf{ek}(h)})[\varsigma \leftarrow \overline{\sigma}^{h\to\_}], \overline{tr}^{h\to\_}]\!] = [\![(\overline{\sigma}^{h\to\_}(C) = \{a\}_{\mathsf{ek}(h)}), \overline{tr}^{h\to\_}]\!] =$$
$$[\![(\{a\}_{\mathsf{ek}(h)} = \{a\}_{\mathsf{ek}(h)}), \overline{tr}^{h\to\_}]\!] = 1.$$

   Nous voyons que $[\![\overline{\phi}^{h\to\_}, \overline{tr}^{h\to\_}]\!] = 1$ n'implique pas $[\![\phi, tr]\!] = 1$.

2. Une substitution peut affecter à une variable un terme qui contient le sous-terme $h \in \mathsf{ID}$. Nous donnons l'exemple suivant : considérons la formule $\phi = (\varsigma(C) = hash(a))$ où la variable de substitution $\varsigma \in Sub$ prend la valeur $\sigma = \{C \leftarrow \{a\}_{\mathsf{ek}(h)}\}$. Nous interprétons cette formule pour une trace $tr \in \mathsf{SymbTr}^h$:

$$[\![(\varsigma(C) = hash(a))[\varsigma \leftarrow \sigma], tr]\!] = [\![(\sigma(C) = hash(a)), tr]\!] = [\![(\{a\}_{\mathsf{ek}(h)} = hash(a)), tr]\!] = 0.$$

Cependant, si nous interprétons la formule $\overline{\phi}^{h\rightarrow\_} = (\varsigma(C) = \{a\}_{\mathsf{ek}(h)})$ pour la trace $\overline{tr}^{h\rightarrow\_}$, nous obtenons

$$[\![(\varsigma(C) = \{a\}_{\mathsf{ek}(h)})[\varsigma \leftarrow \sigma], \overline{tr}^{h\rightarrow\_}]\!] = [\![(\sigma(C) = \{a\}_{\mathsf{ek}(h)}), \overline{tr}^{h\rightarrow\_}]\!] = [\![(\{a\}_{\mathsf{ek}(h)} = \{a\}_{\mathsf{ek}(h)}), \overline{tr}^{h\rightarrow\_}]\!] = 1.$$

Nous voyons que $[\![\overline{\phi}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$ n'implique pas $[\![\phi, tr]\!] = 1$.

En regardant les exemples ci-dessus, nous constatons que la raison pour laquelle $[\![\overline{\phi}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$ n'implique pas $[\![\phi, tr]\!] = 1$ se trouve dans le fait que $\overline{m_1}^{h\rightarrow\_} = \overline{m_2}^{h\rightarrow\_}$ n'implique pas $m_1 = m_2$, $m_1, m_2 \in T^h_{Sub}$. Cette dernière implication est cependant juste, si nous n'autorisons pas des sous-termes de la forme $h \in \mathsf{ID}$ dans les termes m_1 et m_2, comme nous le montrons dans le lemme 1. Pour empêcher que l'identité d'agent h apparaît dans un terme d'une formule de la logique $\mathcal{L}^h_1$, nous devons, d'une part, interdire son occurrence dans la formule elle-même et, d'autre part, exclure qu'elle puisse apparaître dans une transition d'une trace et de cette façon être affectée, possiblement en tant que sous-terme d'un autre terme, à une variable de la formule. Quant à ce deuxième scénario, deux possibilités peuvent y amener : soit, la spécification d'un protocole prévoit l'émission d'un message qui contient le sous-terme h et ensuite l'adversaire peut déduire un terme qui contient l'identité h pour créer un nouveau message, soit, l'adversaire déduit et envoie un terme qui contient le sous-terme h à l'aide de la règle $\mathcal{RH}^{init}_2$ du système de déduction de la figure 2.4. Pour prouver l'implication souhaitée, à savoir $[\![\overline{\phi}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1 \implies [\![\phi, tr]\!] = 1$, nous devons donc adopter les modifications suivantes :

Définition 22. Nous redéfinissons l'ensemble des identités d'agents par $\mathsf{ID}^- = \mathsf{ID} \setminus \{h\}$. Nous définissons l'algèbre T^{h-} comme le sous-ensemble de l'algèbre T^h dont les termes ne contiennent pas le sous-terme $h \in \mathsf{ID}$. Nous changeons la règle $\mathcal{RH}^{init}_2$ de la figure 2.4 en

$$\mathcal{RH}^{init-}_2 \qquad \frac{}{S \vdash^h a, \mathsf{ek}(a)}\, a \in \mathsf{ID}^-.$$

Nous changeons les transitions **corrupt** et **new** de façon à ce qu'elles soient définies pour l'ensemble d'identités ID^-. Nous notons SymbTr^{h-} l'ensemble des traces d'exécution. Soit Roles^{h-} l'ensemble des rôles qui sont spécifiés par des termes de l'algèbre T^{h-} et soit $\Pi : [k] \rightarrow \mathsf{Roles}^{h-}$ un protocole à k participants. Nous dénotons les traces d'exécution valides du protocole Π par $\mathsf{Exec}^{h-}(\Pi)$.

Nous notons $\mathcal{L}^{h-}_1$ la partie de la logique $\mathcal{L}^h_1$ qui utilise exclusivement les termes de l'algèbre T^{h-}_{Sub} qui désigne le sous-ensemble de l'algèbre T^h_{Sub} dont les termes ne contiennent pas le sous-terme $h \in \mathsf{ID}$. Soit $\phi \in \mathcal{L}^{h-}_1$ une formule. Si nous avons $[\![\phi, tr]\!] = 1$ pour tout trace $tr \in \mathsf{Exec}^{h-}(\Pi)$, alors le protocole Π *satisfait la propriété* ϕ. Nous notons cela $\Pi \models \phi$. Nous définissons aussi l'ensemble $\mathsf{Tr}^{h-}(\phi) = \{tr \in \mathsf{SymbTr}^{h-} \mid [\![\phi, tr]\!] = 1\}$.

Preuve de la propriété 1

En adoptant le modèle légèrement modifié, nous prouvons donc d'abord l'implication du lemme 2, c'est-à-dire que $[\![\overline{\phi}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$ implique $[\![\phi, tr]\!] = 1$, qui, lui-même, est basé sur le fait que $\overline{m_1}^{h\rightarrow\_} = \overline{m_2}^{h\rightarrow\_} \implies m_1 = m_2$, implication prouvée dans le lemme suivant :

Lemme 1. *Soient m_1 et m_2 deux termes de T^{h-}_{Sub}. Alors, $\overline{\overset{h\rightarrow}{m_1}} = \overline{\overset{h\rightarrow}{m_2}}$ implique $m_1 = m_2$.*

Preuve. La preuve se fait par induction sur la structure des termes :

- Cas de base : les constantes et expressions substitutionnelles.
  - Soient m_1 et m_2 de la forme a, $n(a,j,s)$, $k(a,j,s)$, $\varsigma(x)$ pour $a \in \mathsf{ID}^-$, $j,s \in \mathbb{N}$, $x \in X$ et $\varsigma \in Sub$. Nous avons $\overline{\overset{h\rightarrow}{m_1}} = m_1$ et $\overline{\overset{h\rightarrow}{m_2}} = m_2$. Si $\overline{\overset{h\rightarrow}{m_1}} = \overline{\overset{h\rightarrow}{m_2}}$, alors $m_1 = m_2$.
  - Soit $\varsigma(x), \varsigma(x') \in SubExpr(X.h)$ avec $\overline{\overset{h\rightarrow}{x}} = y$ et $\overline{\overset{h\rightarrow}{x'}} = y'$ pour $y, y' \in X.c$. Supposons que $\overline{\overset{h\rightarrow}{\varsigma(x)}} = \overline{\overset{h\rightarrow}{\varsigma(x')}}$. Nous avons donc $\varsigma(y) = \varsigma(y')$, c'est-à-dire $y = y'$ et, par conséquent, $\overline{\overset{h\rightarrow}{x}} = \overline{\overset{h\rightarrow}{x'}}$. Nous avons donc $x = x'$, parce que la fonction $\overline{\overset{h\rightarrow}{\cdot}}$ est injective. Nous concluons que $\varsigma(x) = \varsigma(x')$.
  - Dans le cas où $\varsigma(x) \in SubExpr(X)$ et $\varsigma(x') \in SubExpr(X.h)$ avec $\overline{\overset{h\rightarrow}{x'}} = y' \in X.c$, nous avons toujours $\overline{\overset{h\rightarrow}{\varsigma(x)}} \neq \overline{\overset{h\rightarrow}{\varsigma(x')}}$. Cela peut être montré par contradiction : supposons que $\overline{\overset{h\rightarrow}{\varsigma(x)}} = \overline{\overset{h\rightarrow}{\varsigma(x')}}$. Nous avons donc $\varsigma(x) = \varsigma(y')$ et, par conséquent, $x = y'$. Étant donné que nous avons $\overline{\overset{h\rightarrow}{x}} = x$ nous avons alors $\overline{\overset{h\rightarrow}{x}} = \overline{\overset{h\rightarrow}{x'}}$, donc $x = x'$, parce que la fonction $\overline{\overset{h\rightarrow}{\cdot}}$ est injective. Cela est en contradiction au fait que $x \in X$ et $x' \in X.h$, car $X \cap X.h = \emptyset$.
- L'étape d'induction : les termes composés.
  - Soit m_1, m_2 de la forme $\mathsf{ek}(A), \mathsf{dk}(A)$ ou $\mathsf{symk}(A,B)$ avec $A, B \in X_a^{\mathsf{ID}^-} \cup SubExpr(X.a)$. Dans ce cas, nous avons $\overline{\overset{h\rightarrow}{m_1}} = m_1$ et $\overline{\overset{h\rightarrow}{m_2}} = m_2$. Si $\overline{\overset{h\rightarrow}{m_1}} = \overline{\overset{h\rightarrow}{m_2}}$, alors $m_1 = m_2$.
  - Soit $\overline{\overset{h\rightarrow}{m_1}} = \overline{\overset{h\rightarrow}{\langle m_1', m_1'' \rangle}} = \langle \overline{\overset{h\rightarrow}{m_1'}}, \overline{\overset{h\rightarrow}{m_1''}} \rangle$. Si nous avons $\overline{\overset{h\rightarrow}{m_1}} = \overline{\overset{h\rightarrow}{m_2}}$, alors $\overline{\overset{h\rightarrow}{m_2}}$ doit avoir la forme $\langle \overline{\overset{h\rightarrow}{m_2'}}, \overline{\overset{h\rightarrow}{m_2''}} \rangle$ avec $\overline{\overset{h\rightarrow}{m_1'}} = \overline{\overset{h\rightarrow}{m_2'}}$ et $\overline{\overset{h\rightarrow}{m_1''}} = \overline{\overset{h\rightarrow}{m_2''}}$. Par hypothèse d'induction, $m_1' = m_2'$ et $m_1'' = m_2''$. Nous avons donc $m_1 = \langle m_1', m_1'' \rangle = \langle m_2', m_2'' \rangle = m_2$.
  - Nous avons le même raisonnement pour les symboles fonctionnels $\{\!|m_1|\!\}_{m_2}$, $\{m\}_{\mathsf{ek}(a)}$ où $a \in \mathsf{ID}^-$.
  - Soit $\overline{\overset{h\rightarrow}{m_1}} = \overline{\overset{h\rightarrow}{hash(m_1')}} = \{\overline{\overset{h\rightarrow}{m_1'}}\}_{\overline{\overset{h\rightarrow}{\mathsf{ek}(h)}}}$. Si nous avons $\overline{\overset{h\rightarrow}{m_1}} = \overline{\overset{h\rightarrow}{m_2}}$, alors $\overline{\overset{h\rightarrow}{m_2}}$ doit avoir la forme $\{\overline{\overset{h\rightarrow}{m_2'}}\}_{\overline{\overset{h\rightarrow}{\mathsf{ek}(h)}}}$ avec $\overline{\overset{h\rightarrow}{m_1'}} = \overline{\overset{h\rightarrow}{m_2'}}$. Nous avons donc par hypothèse d'induction $m_1' = m_2'$. Étant donné que $m_2 \in T^{h-}$, le terme m_2 ne contient pas le sous-terme $h \in \mathsf{ID}$. Nous concluons donc que $m_2 = hash(m_2')$ et, par conséquent, nous avons $m_1 = m_2$. □

Lemme 2. *Soit $\phi \in \mathcal{L}_1^{h-}$ une formule. Pour une trace $tr \in \mathsf{SymbTr}^{h-}$, nous avons $[\![\overline{\overset{h\rightarrow}{\phi}}, \overline{\overset{h\rightarrow}{tr}}]\!] = 1$ si et seulement si $[\![\phi, tr]\!] = 1$.*

Preuve. La preuve se fait par induction sur la structure des formules.

- $\phi = NC(a)$. Nous avons $[\![NC(a), tr]\!] = 1$ si et seulement si $a \in \mathsf{ID}^-$ est différent de int et n'apparaît pas dans une transition **corrupt** de la trace tr. Cela signifie que $\overline{\overset{h\rightarrow}{a}} \in \mathsf{ID}^-$ est différent de int et n'apparaît pas dans une transition **corrupt** de la trace $\overline{\overset{h\rightarrow}{tr}}$. Nous avons donc $[\![NC(a), tr]\!] = 1$ si et seulement si $[\![NC(\overline{\overset{h\rightarrow}{a}}), \overline{\overset{h\rightarrow}{tr}}]\!] = [\![\overline{\overset{h\rightarrow}{NC(a)}}, \overline{\overset{h\rightarrow}{tr}}]\!] = 1$.

- $\phi = (m_1 = m_2)$ où m_1 et m_2 sont des termes dans T^{h-}_{Sub}. Supposons que $[\![\overline{\phi}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = [\![\overline{(m_1 = m_2)}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = [\![(\overline{m_1}^{h\rightarrow\_} = \overline{m_2}^{h\rightarrow\_}), \overline{tr}^{h\rightarrow\_}]\!] = 1$. Par le lemme 1, $(\overline{m_1}^{h\rightarrow\_} = \overline{m_2}^{h\rightarrow\_})$ implique $(m_1 = m_2)$. Nous avons $[\![(m_1 = m_2), tr]\!] = 1$. D'autre côté, $(m_1 = m_2)$ implique toujours $(\overline{m_1}^{h\rightarrow\_} = \overline{m_2}^{h\rightarrow\_})$. $[\![(m_1 = m_2), tr]\!] = 1$ implique donc $[\![(\overline{m_1}^{h\rightarrow\_} = \overline{m_2}^{h\rightarrow\_}), \overline{tr}^{h\rightarrow\_}]\!] = 1$.
- $\phi = \neg\phi'$. Supposons que $[\![\overline{\neg\phi'}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = [\![\neg\overline{\phi'}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$. Nous avons $[\![\overline{\phi'}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 0$ et cela, par hypothèse d'induction, si et seulement si $[\![\phi', tr]\!] = 0$ et donc $[\![\neg\phi', tr]\!] = 1$.
- $\phi = \phi_1 \wedge \phi_2$. Supposons que $[\![\overline{\phi_1 \wedge \phi_2}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = [\![\overline{\phi_1}^{h\rightarrow\_} \wedge \overline{\phi_2}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$. Cela signifie que $[\![\overline{\phi_1}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$ et $[\![\overline{\phi_2}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$. Par hypothèse d'induction, si et seulement si $[\![\phi_1, tr]\!] = 1$ et $[\![\phi_2, tr]\!] = 1$, donc $[\![\phi_1 \wedge \phi_2, tr]\!] = 1$.
- $\phi = \phi_1 \vee \phi_2$. Supposons que $[\![\overline{\phi_1 \vee \phi_2}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = [\![\overline{\phi_1}^{h\rightarrow\_} \vee \overline{\phi_2}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$. Cela signifie que $[\![\overline{\phi_1}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$ ou $[\![\overline{\phi_2}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$. Par hypothèse d'induction, si et seulement si $[\![\phi_1, tr]\!] = 1$ ou $[\![\phi_2, tr]\!] = 1$, donc $[\![\phi_1 \vee \phi_2, tr]\!] = 1$.
- $\phi = \forall\mathcal{LS}_{i,p}.\varsigma\ \phi'$. Supposons que $[\![\overline{\forall\mathcal{LS}_{i,p}.\varsigma\ \phi'}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = [\![\forall\mathcal{LS}_{i,p}.\varsigma\ \overline{\phi'}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$. $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$ implique $(\overline{\theta}^{h\rightarrow\_}, p) \in \mathcal{LS}_{i,p}(\overline{tr}^{h\rightarrow\_})$. Pour tout état local $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$ nous avons donc $[\![\overline{\phi'}^{h\rightarrow\_}[\varsigma \leftarrow \overline{\theta}^{h\rightarrow\_}], \overline{tr}^{h\rightarrow\_}]\!] = 1$. Par hypothèse d'induction, nous avons $[\![\overline{\phi'}^{h\rightarrow\_}[\varsigma \leftarrow \overline{\theta}^{h\rightarrow\_}], \overline{tr}^{h\rightarrow\_}]\!] = [\![\overline{\phi'[\varsigma \leftarrow \theta]}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$, si et seulement si $[\![\phi'[\varsigma \leftarrow \theta], tr]\!] = 1$. Nous avons donc $[\![\forall\mathcal{LS}_{i,p}.\varsigma\ \phi', tr]\!] = 1$.
- $\phi = \exists\mathcal{LS}_{i,p}.\varsigma\ \phi'$. Supposons que $[\![\overline{\exists\mathcal{LS}_{i,p}.\varsigma\ \phi'}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = [\![\exists\mathcal{LS}_{i,p}.\varsigma\ \overline{\phi'}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$. $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$ implique $(\overline{\theta}^{h\rightarrow\_}, p) \in \mathcal{LS}_{i,p}(\overline{tr}^{h\rightarrow\_})$. Il existe donc un état local $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$, tel que $[\![\overline{\phi'}^{h\rightarrow\_}[\varsigma \leftarrow \overline{\theta}^{h\rightarrow\_}], \overline{tr}^{h\rightarrow\_}]\!] = 1$. Par hypothèse d'induction, nous avons $[\![\overline{\phi'}^{h\rightarrow\_}[\varsigma \leftarrow \overline{\theta}^{h\rightarrow\_}], \overline{tr}^{h\rightarrow\_}]\!] = [\![\overline{\phi'[\varsigma \leftarrow \theta]}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$ si et seulement si $[\![\phi'[\varsigma \leftarrow \theta], tr]\!] = 1$. Nous concluons que $[\![\exists\mathcal{LS}_{i,p}.\varsigma\ \phi', tr]\!] = 1$. □

La prochaine étape de la preuve de la propriété 1 est la preuve de l'implication $tr \in \mathsf{Exec}^{h-}(\Pi) \implies \overline{tr}^{h\rightarrow\_} \in \mathsf{Exec}(\overline{\Pi}^{h\rightarrow\_})$, énoncée par le lemme 6. Pour prouver ce lemme, nous avons besoin de trois lemmes auxiliaires. Nous prouvons ces trois lemmes pour l'algèbre T^h, car il n'est pas important si les termes contiennent le sous-terme $h \in \mathsf{ID}$.

Lemme 3. *Soit σ une substitution et t un terme dans T^h. Nous avons $\overline{t\sigma}^{h\rightarrow\_} = \overline{t}^{h\rightarrow\_}\overline{\sigma}^{h\rightarrow\_}$.*

Preuve. La preuve se fait par induction sur la structure des termes.

- Les cas de base : les variables et les constantes.
  - Soit t une variable.

    Ou bien $t \in X \setminus X.h$. Alors, nous avons $\overline{t\sigma}^{h\rightarrow\_} = t\overline{\sigma}^{h\rightarrow\_} = \overline{t}^{h\rightarrow\_}\overline{\sigma}^{h\rightarrow\_}$.

    Ou bien $t \in X.h$ est une variable du type hachage. Alors, nous avons $\overline{t}^{h\rightarrow\_} = x$ pour une variable $x \in X.c$. Par conséquent, $\overline{t\sigma}^{h\rightarrow\_} = x\overline{\sigma}^{h\rightarrow\_} = \overline{t}^{h\rightarrow\_}\overline{\sigma}^{h\rightarrow\_}$.
  - Soit t est de la forme a, $n(a, j, s)$, ou $k(a, j, s)$ où et j et s des entiers. Alors, $\overline{t\sigma}^{h\rightarrow\_} = t = \overline{t}^{h\rightarrow\_}\overline{\sigma}^{h\rightarrow\_}$.

- L'étape d'induction : les termes composés.
  - Soit t de la forme $\mathsf{ek}(A)$, $\mathsf{dk}(A)$ ou $\mathsf{symk}(A,B)$ où A et $B \in X_a^{\mathsf{ID}}$ sont des identités d'agents ou des variables de ce type. Nous montrons le lemme pour le terme $t = \mathsf{ek}(A)$. Les autres cas sont similaires. Nous avons $\overline{t\sigma}^{h\to} = \overline{\mathsf{ek}(A)\sigma}^{h\to} = \overline{\mathsf{ek}(A\sigma)}^{h\to} = \mathsf{ek}(\overline{A\sigma}^{h\to}) = \mathsf{ek}(\overline{A}^{h\to}\overline{\sigma}^{h\to}) = \mathsf{ek}(\overline{A}^{h\to})\overline{\sigma}^{h\to} = \overline{\mathsf{ek}(A)}^{h\to}\overline{\sigma}^{h\to} = \overline{t}^{h\to}\overline{\sigma}^{h\to}$.
  - Soit $t = \langle t_1, t_2\rangle$. Nous avons $\overline{t\sigma}^{h\to} = \overline{\langle t_1, t_2\rangle\sigma}^{h\to} = \overline{\langle t_1\sigma, t_2\sigma\rangle}^{h\to} = \langle \overline{t_1\sigma}^{h\to}, \overline{t_2\sigma}^{h\to}\rangle = \langle \overline{t_1}^{h\to}\overline{\sigma}^{h\to}, \overline{t_2}^{h\to}\overline{\sigma}^{h\to}\rangle = \langle \overline{t_1}^{h\to}, \overline{t_2}^{h\to}\rangle\overline{\sigma}^{h\to} = \overline{\langle t_1, t_2\rangle}^{h\to}\overline{\sigma}^{h\to} = \overline{t}^{h\to}\overline{\sigma}^{h\to}$.
  - Soit $t = \{\!|t_1|\!\}_{t_2}$. Nous avons $\overline{t\sigma}^{h\to} = \overline{\{\!|t_1|\!\}_{t_2}\sigma}^{h\to} = \overline{\{\!|t_1\sigma|\!\}_{t_2\sigma}}^{h\to} = \{\!|\overline{t_1\sigma}^{h\to}|\!\}_{\overline{t_2\sigma}^{h\to}} = \{\!|\overline{t_1}^{h\to}\overline{\sigma}^{h\to}|\!\}_{\overline{t_2}^{h\to}\overline{\sigma}^{h\to}} = \{\!|\overline{t_1}^{h\to}|\!\}_{\overline{t_2}^{h\to}}\overline{\sigma}^{h\to} = \overline{\{\!|t_1|\!\}_{t_2}}^{h\to}\overline{\sigma}^{h\to} = \overline{t}^{h\to}\overline{\sigma}^{h\to}$.
  - Soit $t = \{t'\}_{\mathsf{ek}(a)}$. Nous avons $\overline{t\sigma}^{h\to} = \overline{\{t'\}_{\mathsf{ek}(a)}\sigma}^{h\to} = \overline{\{t'\sigma\}_{\mathsf{ek}(a)\sigma}}^{h\to} = \{\overline{t'\sigma}^{h\to}\}_{\overline{\mathsf{ek}(a)\sigma}^{h\to}} = \{\overline{t'}^{h\to}\overline{\sigma}^{h\to}\}_{\overline{\mathsf{ek}(a)}^{h\to}\overline{\sigma}^{h\to}} = \{\overline{t'}^{h\to}\}_{\overline{\mathsf{ek}(a)}^{h\to}}\overline{\sigma}^{h\to} = \overline{\{t'\}_{\mathsf{ek}(a)}}^{h\to}\overline{\sigma}^{h\to} = \overline{t}^{h\to}\overline{\sigma}^{h\to}$.
  - Soit $t = hash(t')$. Nous avons $\overline{t\sigma}^{h\to} = \overline{hash(t')\sigma}^{h\to} = \overline{hash(t'\sigma)}^{h\to} = \{\overline{t'\sigma}^{h\to}\}_{\overline{\mathsf{ek}(h)}^{h\to}} = \{\overline{t'}^{h\to}\overline{\sigma}^{h\to}\}_{\overline{\mathsf{ek}(h)}^{h\to}} = \{\overline{t'}^{h\to}\}_{\overline{\mathsf{ek}(h)}^{h\to}}\overline{\sigma}^{h\to} = \overline{hash(t')}^{h\to}\overline{\sigma}^{h\to} = \overline{t}^{h\to}\overline{\sigma}^{h\to}$. □

Lemme 4. *Soient t et m deux termes dans T^h, tels que $m = t\sigma$. Il existe une substitution σ', telle que $\overline{m}^{h\to} = \overline{t}^{h\to}\sigma'$.*

Preuve. Nous avons $\overline{m}^{h\to} = \overline{t\sigma}^{h\to} = \overline{t}^{h\to}\overline{\sigma}^{h\to}$. On choisit $\sigma' = \overline{\sigma}^{h\to}$. □

Lemme 5. *Soit $S \subseteq T^h$. $S \vdash^h m$ implique $\overline{S}^{h\to} \vdash \overline{m}^{h\to}$.*

Preuve. Nous faisons une induction sur le nombre d'applications de règles de déduction pour déduire m de S. Les systèmes de déduction se trouvent dans les figures 2.4 et 2.5.

- Cas de base : Seulement une règle été nécessaire pour déduire m de S.
  - La règle utilisée était la règle $\mathcal{RH}_1^{init}$. Nous avons donc $m \in S$. Par définition, $\overline{m}^{h\to} \in \overline{S}^{h\to}$, donc $\overline{S}^{h\to} \vdash \overline{m}^{h\to}$ avec la règle $\mathcal{R}_1^{init}$.
  - La règle utilisée était la règle $\mathcal{RH}_2^{init}$. Le terme m est donc de la forme $a \in \mathsf{ID}$ ou $\mathsf{ek}(a)$. Alors, $\overline{S}^{h\to} \vdash \overline{m}^{h\to}$ avec la règle $\mathcal{R}_2^{init}$.
- L'étape d'induction : Nous considérons la dernière règle utilisée.
  - La règle $\mathcal{RH}_{comp}^{pair}$.
    Nous avons $m = \langle m_1, m_2\rangle$ et la dernière règle pour obtenir m était la règle $\mathcal{RH}_{comp}^{pair}$. Les termes m_1, m_2 sont donc déductibles à partir de S. Nous avons $\overline{S}^{h\to} \vdash \overline{m_1}^{h\to}$ et $\overline{S}^{h\to} \vdash \overline{m_2}^{h\to}$ par hypothèse d'induction et donc $\overline{S}^{h\to} \vdash \langle \overline{m_1}^{h\to}, \overline{m_2}^{h\to}\rangle = \overline{\langle m_1, m_2\rangle}^{h\to} = \overline{m}^{h\to}$ avec la règle $\mathcal{R}_{comp}^{pair}$.

- La règle $\mathcal{RH}^{pair}_{dec}$.
  Le terme $m = m_i$, $i \in \{1,2\}$, a pu être déduit à partir d'un terme $\langle m_1, m_2\rangle$ qui est déductible à partir de S en appliquant la règle $\mathcal{RH}^{pair}_{dec}$. Par hypothèse d'induction, nous avons $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{\langle m_1, m_2\rangle}} = \langle \overset{h\rightarrow}{\overline{m_1}}, \overset{h\rightarrow}{\overline{m_2}}\rangle$ et donc $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{m_i}}$ avec la règle $\mathcal{R}^{pair}_{dec}$.
- La règle $\mathcal{RH}^{sym}_{c}$.
  La dernière règle utilisée pour déduire le terme $m = \{\!|m_1|\!\}_{m_2}$ était la règle $\mathcal{RH}^{sym}_{c}$. Nous avons donc $S \vdash^h m_1$ et $S \vdash^h m_2$. Par hypothèse d'induction, nous avons $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{m_1}}$ et $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{m_2}}$, donc $\overset{h\rightarrow}{\overline{S}} \vdash \{\!|\overset{h\rightarrow}{\overline{m_1}}|\!\}_{\overset{h\rightarrow}{\overline{m_2}}} = \overset{h\rightarrow}{\overline{\{\!|m_1|\!\}_{m_2}}} = \overset{h\rightarrow}{\overline{m}}$ avec la règle $\mathcal{R}^{sym}_{c}$
- La règle $\mathcal{RH}^{sym}_{d}$.
  Le terme m_1 a été déduit à partir des terme $\{\!|m_1|\!\}_{m_2}$ et m_2 où la règle $\mathcal{RH}^{sym}_{d}$ est la dernière règle utilisée. Nous avons donc $S \vdash^h \{\!|m_1|\!\}_{m_2}$ et $S \vdash^h m_2$. Par hypothèse d'induction, nous avons $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{\{\!|m_1|\!\}_{m_2}}} = \{\!|\overset{h\rightarrow}{\overline{m_1}}|\!\}_{\overset{h\rightarrow}{\overline{m_2}}}$ et $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{m_2}}$, donc $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{m_1}}$ avec la règle $\mathcal{R}^{sym}_{d}$.
- La règle $\mathcal{RH}^{asym}_{c}$.
  La règle $\mathcal{RH}^{asym}_{c}$ est la dernière règle utilisé pour déduire le terme $m = \{m'\}_{\mathsf{ek}(a)}$. Nous avons donc $S \vdash^h \mathsf{ek}(a)$ et $S \vdash^h m'$. Par hypothèse d'induction, nous avons $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{\mathsf{ek}(a)}} = \mathsf{ek}(a)$ et $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{m'}}$ et donc $\overset{h\rightarrow}{\overline{S}} \vdash \{\overset{h\rightarrow}{\overline{m'}}\}_{\mathsf{ek}(a)} = \{\overset{h\rightarrow}{\overline{m'}}\}_{\overset{h\rightarrow}{\overline{\mathsf{ek}(a)}}} = \overset{h\rightarrow}{\overline{\{m'\}_{\mathsf{ek}(a)}}} = \overset{h\rightarrow}{\overline{m}}$ avec la règle $\mathcal{R}^{asym}_{c}$.
- La règle $\mathcal{RH}^{asym}_{d}$.
  La règle $\mathcal{RH}^{asym}_{d}$ a été utilisée comme dernière règle pour déduire le terme m à partir des termes $\{m\}_{\mathsf{ek}(a)}$ et $\mathsf{dk}(a)$. Nous avons donc $S \vdash^h \{m\}_{\mathsf{ek}(a)}$ et $S \vdash^h \mathsf{dk}(a)$. Par hypothèse d'induction, nous avons $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{\{m\}_{\mathsf{ek}(a)}}} = \{\overset{h\rightarrow}{\overline{m}}\}_{\overset{h\rightarrow}{\overline{\mathsf{ek}(a)}}} = \{\overset{h\rightarrow}{\overline{m}}\}_{\mathsf{ek}(a)}$ et $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{\mathsf{dk}(a)}} = \mathsf{dk}(a)$, donc $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{m}}$ avec la règle $\mathcal{R}^{asym}_{d}$.
- Les règles $\mathcal{RH}^{asym}_{cinv}$ et $\mathcal{RH}^{asym}_{dinv}$. Ces cas sont analogues aux cas des règles $\mathcal{RH}^{asym}_{c}$ et $\mathcal{RH}^{asym}_{d}$.
- La règle $\mathcal{RH}^{hash}$.
  La dernière règle pour déduire le terme $m = hash(m')$ était la règle $\mathcal{RH}^{hash}$. Nous avons donc $S \vdash^h m'$. Par hypothèse d'induction, $\overset{h\rightarrow}{\overline{S}} \vdash \overset{h\rightarrow}{\overline{m'}}$. Nous avons toujours $\overset{h\rightarrow}{\overline{S}} \vdash \mathsf{ek}(h)$ avec la règle $\mathcal{R}^{init}_{2}$. Nous appliquons la règle de chiffrement asymétrique $\mathcal{R}^{asym}_{c}$ et obtenons $\overset{h\rightarrow}{\overline{S}} \vdash \{\overset{h\rightarrow}{\overline{m'}}\}_{\mathsf{ek}(h)} = \{\overset{h\rightarrow}{\overline{m'}}\}_{\overset{h\rightarrow}{\overline{\mathsf{ek}(h)}}} = \overset{h\rightarrow}{\overline{hash(m')}} = \overset{h\rightarrow}{\overline{m}}$. □

Nous prouvons maintenant le lemme 6. Pour ce faire, nous avons besoin de la notion de la longueur d'une trace.

Définition 23. La *longueur d'une trace* est le nombre d'états globaux dans cette trace.

Lemme 6. *Soit* $\Pi : [k] \rightarrow \mathsf{Roles}^h$ *un protocole.* $tr \in \mathsf{Exec}^{h-}(\Pi)$ *implique* $\overset{h\rightarrow}{\overline{tr}} \in \mathsf{Exec}(\overset{h\rightarrow}{\overline{\Pi}})$.

Preuve. La preuve se fait par induction sur la longueur de trace.

– Cas de base : nous considérons les traces de longueur 1.
Soit $tr = (\mathsf{SId}_1, f_1, H_1) \in \mathsf{Exec}^{h-}(\Pi)$ une trace d'exécution valide de longueur 1. Nous avons $\mathsf{SId}_1 = \emptyset$ et $H_1 = \mathbf{kn}^h(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}$. Comme l'ensemble SId_1 est vide, la fonction f_1 n'est définie nulle part. Considérons maintenant la trace $\overline{\overline{tr}^{h\rightarrow\_}} = (\mathsf{SId}_1, \overline{\overline{f_1}^{h\rightarrow\_}}, \overline{\overline{H_1}^{h\rightarrow\_}})$. La fonction $\overline{\overline{f_1}^{h\rightarrow\_}}$ n'est définie nulle part. Quant à l'ensemble $\overline{\overline{H_1}^{h\rightarrow\_}}$, nous avons $\overline{\overline{H_1}^{h\rightarrow\_}} = \mathbf{kn}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}$. Nous concluons que $\overline{\overline{tr}^{h\rightarrow\_}} \in \mathsf{Exec}(\overline{\overline{\Pi}^{h\rightarrow\_}})$ est une trace valide.
– L'étape d'induction : nous considérons les traces de longueur $n+1$.
Soit $tr = (\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\leq i\leq n+1} \in \mathsf{Exec}^{h-}(\Pi)$ une trace d'exécution valide de longueur $n+1$. Par hypothèse d'induction, nous pouvons appliquer le lemme à la trace

$$(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\leq i\leq n} \in \mathsf{Exec}^{h-}(\Pi)$$

et, par conséquent,

$$\frac{\overline{\quad\quad}^{h\rightarrow\_}}{((\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\leq i\leq n})} \in \mathsf{Exec}(\overline{\overline{\Pi}^{h\rightarrow\_}})$$

est une trace d'exécution valide. Il y a trois cas possibles pour la n-ième transition de la trace tr :

1. $\mathbf{trans}_n = \mathbf{corrupt}(a_1, \ldots, a_l)$.

   Cela signifie que $tr = (\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}_2, f_2, H_2) \in \mathsf{Exec}^{h-}(\Pi)$, parce que la transition **corrupt** est seulement permise pour la première transition. Nous avons $\mathsf{SId}_1 = \mathsf{SId}_2 = \emptyset$, $f_1 = f_2$ et $H_2 = H_1 \cup \bigcup_{1\leq i\leq l} \mathbf{kn}^h(a_i)$.

   Considérons maintenant la trace $tr = \frac{\overline{\quad\quad}^{h\rightarrow\_}}{(\mathsf{SId}_1, f_1, H_1)} \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}_2, \overline{\overline{f_2}^{h\rightarrow\_}}, \overline{\overline{H_2}^{h\rightarrow\_}})$.

   Nous avons $\mathsf{SId}_2 = \mathsf{SId}_1 = \emptyset$. La fonction $\overline{\overline{f_2}^{h\rightarrow\_}}$ n'est définie nulle part et nous avons donc $\overline{\overline{f_2}^{h\rightarrow\_}} = \overline{\overline{f_1}^{h\rightarrow\_}}$. Quant à l'ensemble $\overline{\overline{H_2}^{h\rightarrow\_}}$, nous avons $\overline{\overline{H_2}^{h\rightarrow\_}} = \frac{\overline{\quad\quad}^{h\rightarrow\_}}{H_1 \cup \bigcup_{1\leq i\leq l} \mathbf{kn}^h(a_i)} = \overline{\overline{H_1}^{h\rightarrow\_}} \cup \bigcup_{1\leq i\leq l} \mathbf{kn}(a_i)$. Nous concluons que $\overline{\overline{tr}^{h\rightarrow\_}} \in \mathsf{Exec}(\overline{\overline{\Pi}^{h\rightarrow\_}})$.

2. $\mathbf{trans}_n = \mathbf{new}(i, a_1, \ldots, a_k)$.

   Nous avons donc $tr = \ldots(\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})$. Pour $\mathsf{sid} = (|\mathsf{SId}_n|+1, i, (a_1, \ldots, a_k))$, nous avons $\mathsf{SId}_{n+1} = \mathsf{SId}_n \cup \{\mathsf{sid}\}$ et $f_{n+1}(\mathsf{sid}) = (\sigma, 1)$. Pour tous les $s \in \mathsf{SId}_n$, nous avons $f_{n+1}(s) = f_n(s)$. Finalement, $H_{n+1} = H_n$.

   Nous considérons la trace $\overline{\overline{tr}^{h\rightarrow\_}} = \frac{\overline{\quad\quad}^{h\rightarrow\_}}{(\ldots(\mathsf{SId}_n, f_n, H_n))} \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}_{n+1}, \overline{\overline{f_{n+1}}^{h\rightarrow\_}}, \overline{\overline{H_{n+1}}^{h\rightarrow\_}})$.

   Pour $s \in \mathsf{SId}_n$, nous avons $\overline{\overline{f_{n+1}}^{h\rightarrow\_}}(s) = \overline{\overline{f_n}^{h\rightarrow\_}}(s)$. Considérons l'état local $(\sigma', 1)$ qui est affecté au nouvel identifiant de session sid. Nous avons $\sigma' = \sigma$, parce que les domaines des deux substitutions contiennent les mêmes variables. Ces variables ont le type agent, nonce et clef de chiffrement symétrique à court terme. Étant donné que l'identifiant de session sid ne change pas, les mêmes termes sont affecté aux variables des domaines de deux substitutions. D'autre côté, nous avons pour un terme ou une variable m des types mentionnés $\overline{\overline{m}^{h\rightarrow\_}} = m$. Par conséquent, $\overline{\overline{\sigma}^{h\rightarrow\_}} = \sigma$ et

donc $(\sigma', 1) = (\overline{\overline{\sigma}^{h\rightarrow}}^{\_}, 1) = \overline{\overline{f_{n+1}}^{h\rightarrow}}^{\_}(\mathsf{sid})$. Finalement, $\overline{\overline{H_{n+1}}^{h\rightarrow}}^{\_} = \overline{\overline{H_n}^{h\rightarrow}}^{\_}$. Nous concluons que $\overline{\overline{tr}^{h\rightarrow}}^{\_} \in \mathsf{Exec}(\overline{\overline{\Pi}^{h\rightarrow}}^{\_})$.

3. $\mathbf{trans}_n = \mathbf{send}(\mathsf{sid}, m)$.

   Nous avons donc $tr = \ldots(\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})$. Nous avons $\mathsf{SId}_{n+1} = \mathsf{SId}_n$ et $f_{n+1}(s) = f_n(s)$ pour toute session $s \neq \mathsf{sid}$ dans SId_n.
   Soit $\overline{\overline{tr}^{h\rightarrow}}^{\_} = \overline{(\ldots(\mathsf{SId}_n, f_n, H_n))}^{h\rightarrow}{}^{\_} \xrightarrow{\mathbf{send}(\mathsf{sid},m')} (\mathsf{SId}_{n+1}, \overline{\overline{f_{n+1}}^{h\rightarrow}}^{\_}, \overline{\overline{H_{n+1}}^{h\rightarrow}}^{\_})$ où le terme m' a été choisi selon la définition de la fonction $\overline{\cdot}^{h\rightarrow}{}^{\_}$ pour les traces. Nous avons $\overline{\overline{f_{n+1}}^{h\rightarrow}}^{\_}(s) = \overline{\overline{f_n}^{h\rightarrow}}^{\_}(s)$ pour tout identifiant de session $s \neq \mathsf{sid}$.
   Dans la suite, nous supposons que nous avons $\mathsf{sid} = (x, j, (a_1, \ldots, a_k))$ et le rôle joué dans cette session est donné par $\Pi(j) = (l_i, r_i)_{1\leq i\leq l}$. Soit $f_n(\mathsf{sid}) = (\sigma, p)$ l'état de cette session avant la transition. Nous distinguons deux cas :
   - Soit, nous avons $p \leq l$ et il existe une substitution θ, telle que $m = l_p\sigma\theta$. Nous avons $f_{n+1}(\mathsf{sid}) = (\sigma \cup \theta, p+1)$ et $H_{n+1} = H_n \cup \{r_p\sigma\theta\}$.
     Dans ce cas, nous avons $m' = \overline{\overline{m}^{h\rightarrow}}^{\_}$. Le lemme 5 assure que nous avons $\overline{\overline{H_n}^{h\rightarrow}}^{\_} \vdash \overline{\overline{m}^{h\rightarrow}}^{\_}$, car m est déductible à partir de H_n. Nous avons $\overline{\overline{f_n}^{h\rightarrow}}^{\_}(\mathsf{sid}) = (\overline{\overline{\sigma}^{h\rightarrow}}^{\_}, p)$ et les lemmes 3 et 4 assurent que $\overline{\overline{m}^{h\rightarrow}}^{\_}$ et $\overline{\overline{l_p}^{h\rightarrow}}^{\overline{\sigma}^{h\rightarrow}}$ sont unifiable avec $\overline{\overline{\theta}^{h\rightarrow}}^{\_}$ comme unificateur. Nous avons donc $\overline{\overline{f_{n+1}}^{h\rightarrow}}^{\_}(\mathsf{sid}) = (\overline{\overline{\sigma}^{h\rightarrow}}^{\_} \cup \overline{\overline{\theta}^{h\rightarrow}}^{\_}, p+1)$. Nous avons $\overline{\overline{H_{n+1}}^{h\rightarrow}}^{\_} = \overline{\overline{H_n}^{h\rightarrow}}^{\_} \cup \{\overline{\overline{r_p}^{h\rightarrow}}^{\overline{\sigma}^{h\rightarrow}}\overline{\theta}^{h\rightarrow}\}^{\_} = \overline{H_n \cup \{r_p\sigma\theta\}}^{h\rightarrow}{}^{\_}$. La trace $\overline{\overline{tr}^{h\rightarrow}}^{\_}$ est valide et nous avons donc $\overline{\overline{tr}^{h\rightarrow}}^{\_} \in \mathsf{Exec}(\overline{\overline{\Pi}^{h\rightarrow}}^{\_})$.
   - Soit, nous avons $f_{n+1}(\mathsf{sid}) = f_n(\mathsf{sid})$ et $H_{n+1} = H_n$. Cela peut avoir deux raisons :
     - Soit, $p > l$. Nous avons $m' = a$ pour une identité de session $a \in \mathsf{ID}$. Le système de déduction nous assure que nous avons $\overline{\overline{H_n}^{h\rightarrow}}^{\_} \vdash a$. De plus, $\overline{\overline{f_{n+1}}^{h\rightarrow}}^{\_}(\mathsf{sid}) = \overline{\overline{f_n}^{h\rightarrow}}^{\_}(\mathsf{sid})$ et $\overline{\overline{H_{n+1}}^{h\rightarrow}}^{\_} = \overline{\overline{H_n}^{h\rightarrow}}^{\_}$. Nous avons donc $\overline{\overline{tr}^{h\rightarrow}}^{\_} \in \mathsf{Exec}(\overline{\overline{\Pi}^{h\rightarrow}}^{\_})$.
     - Soit, $p \leq l$ et les termes m et $l_p\sigma$ ne sont pas unifiables. Nous distinguons encore deux cas :
       - Soit, les termes $\overline{\overline{m}^{h\rightarrow}}^{\_}$ et $\overline{\overline{l_p\sigma}^{h\rightarrow}}^{\_}$ ne sont pas unifiables. Nous avons $m' = \overline{\overline{m}^{h\rightarrow}}^{\_}$. Le lemme 5 assure que $\overline{\overline{H_n}^{h\rightarrow}}^{\_} \vdash m'$, parce que nous avons $H_n \vdash^h m$. Nous avons donc $\overline{\overline{f_{n+1}}^{h\rightarrow}}^{\_}(\mathsf{sid}) = \overline{\overline{f_n}^{h\rightarrow}}^{\_}(\mathsf{sid})$ et $\overline{\overline{H_{n+1}}^{h\rightarrow}}^{\_} = \overline{\overline{H_n}^{h\rightarrow}}^{\_}$. La trace $\overline{\overline{tr}^{h\rightarrow}}^{\_}$ est donc valide, c'est-à-dire $\overline{\overline{tr}^{h\rightarrow}}^{\_} \in \mathsf{Exec}(\overline{\overline{\Pi}^{h\rightarrow}}^{\_})$.
       - Soit, les termes $\overline{\overline{m}^{h\rightarrow}}^{\_}$ et $\overline{\overline{l_p\sigma}^{h\rightarrow}}^{\_}$ sont unifiables. Nous avons donc $m' = a$ pour une identité d'agent $a \in \mathsf{ID}$. Nous avons $\overline{\overline{H_n}^{h\rightarrow}}^{\_} \vdash a$. Les termes m' et $\overline{\overline{l_p\sigma}^{h\rightarrow}}^{\_}$ ne sont pas unifiables, parce que le terme $\overline{\overline{l_p\sigma}^{h\rightarrow}}^{\_}$ contient un sous-terme de la forme $\{t\}_{\mathsf{ek}(h)}$. Cela peut être montré par contradiction : supposons que $\overline{\overline{m}^{h\rightarrow}}^{\_}$ ne contient pas un sous-terme de la forme $\{t\}_{\mathsf{ek}(h)}$. Nous avons donc $m = \overline{\overline{m}^{h\rightarrow}}^{\_}$. Il existe une substitution θ, telle que $\overline{\overline{m}^{h\rightarrow}}^{\_} = \overline{\overline{l_p\sigma}^{h\rightarrow}}^{\_}\theta$. Étant donné que $\overline{\overline{m}^{h\rightarrow}}^{\_}$ ne contient pas un sous-terme de la forme $\{t\}_{\mathsf{ek}(h)}$, $\overline{\overline{l_p\sigma}^{h\rightarrow}}^{\_}$ ne contient également pas

un sous-terme de cette forme. Nous avons donc $l_p\sigma = \overline{l_p\sigma}^{h\rightarrow\_}$. Nous avons alors que $\overline{m}^{h\rightarrow\_} = \overline{l_p\sigma}^{h\rightarrow\_}\theta$ implique $m = l_p\sigma\theta$, ce qui est une contradiction, parce que les termes $l_p\sigma$ et m ne sont pas unifiables. On conclut que $\overline{l_p\sigma}^{h\rightarrow\_}$ contient un sous-terme de la forme $\{t\}_{\mathsf{ek}(h)}$ et, par conséquent, les termes $m' = a$ et $\overline{l_p\sigma}^{h\rightarrow\_}$ ne sont pas unifiables. Nous avons donc $\overline{f_{n+1}}^{h\rightarrow\_}(\mathsf{sid}) = \overline{f_n}^{h\rightarrow\_}(\mathsf{sid})$ et $\overline{H_{n+1}}^{h\rightarrow\_} = \overline{H_n}^{h\rightarrow\_}$. Nous concluons que $\overline{tr}^{h\rightarrow\_} \in \mathsf{Exec}(\overline{\Pi}^{h\rightarrow\_})$.

□

Finalement, les lemmes 2 et 6 nous permettent de prouvé le théorème désiré.

Théorème 1. *Soit* $\Pi : [k] \rightarrow \mathsf{Roles}^{h-}$ *un protocole à* k *participants et soit* $\phi \in \mathcal{L}_1^{h-}$ *une formule. Nous avons* $\overline{\Pi}^{h\rightarrow\_} \models \overline{\phi}^{h\rightarrow\_} \implies \Pi \models \phi$.

Preuve. Nous montrons que $\mathsf{Exec}(\overline{\Pi}^{h\rightarrow\_}) \subseteq \mathsf{Tr}(\overline{\phi}^{h\rightarrow\_})$ implique $\mathsf{Exec}^{h-}(\Pi) \subseteq \mathsf{Tr}^{h-}(\phi)$. Soit $tr \in \mathsf{Exec}^{h-}(\Pi)$. Nous obtenons alors $\overline{tr}^{h\rightarrow\_} \in \mathsf{Exec}(\overline{\Pi}^{h\rightarrow\_})$ par le lemme 6 et donc $\overline{tr}^{h\rightarrow\_} \in \mathsf{Tr}(\overline{\phi}^{h\rightarrow\_})$. Nous avons donc $[\![\overline{\phi}^{h\rightarrow\_}, \overline{tr}^{h\rightarrow\_}]\!] = 1$, ce qui implique par le lemme 2 que $[\![\phi, tr]\!] = 1$, donc $tr \in \mathsf{Tr}^{h-}(\phi)$. □

Contre-exemple pour la propriété 2

Nous nous situons maintenant dans notre modèle habituel, c'est-à-dire que l'identité d'agent h n'est plus traitée spécialement. Considérons le protocole Π, donné par

$$\begin{aligned}\Pi(1) &= (\mathsf{init}, hash(X^1_{A_1}))\\ \Pi(2) &= (C, \mathsf{stop}).\end{aligned}$$

où $C \in X.c$ est une variable du type chiffrement. Le premier rôle émet la valeur hachée d'un nonce. Le deuxième rôle attend un message chiffré. Il n'y a pas d'exécution raisonnable de ce protocole. Considérons la propriété

$$\phi = \forall\mathcal{LS}_{1,2}.\varsigma\ \forall\mathcal{LS}_{2,2}.\varsigma'\ \neg(\varsigma'(C) = \{\varsigma(X^1_{A_1})\}_{\mathsf{ek}(h)}).$$

Elle exprime qu'il n'y a pas d'états locaux du premier, respectivement deuxième rôle, tels que la variable C prend la valeur $\{X^1_{A_1}\}_{\mathsf{ek}(h)}$. Intuitivement, cette propriété est satisfaite, parce que la valeur du nonce $X^1_{A_1}$ n'est pas déductible par l'intrus. Notons que cela est indépendant du fait que l'agent qui joue le premier rôle est corrompu ou non. L'intrus ne peut donc pas chiffrer la valeur $X^1_{A_1}$ avec la clef publique $\mathsf{ek}(h)$. Nous avons $\Pi \models \phi$.

Considérons maintenant le protocole $\overline{\Pi}^{h\rightarrow\_}$:

$$\begin{aligned}\overline{\Pi(1)}^{h\rightarrow\_} &= (\mathsf{init}, \{X^1_{A_1}\}_{\mathsf{ek}(h)})\\ \overline{\Pi(2)}^{h\rightarrow\_} &= (C, \mathsf{stop}).\end{aligned}$$

et la propriété $\overline{\phi}^{h\rightarrow\_}$, donnée par

$$\overline{\phi}^{h\rightarrow\_} = \forall\mathcal{LS}_{1,2}.\varsigma\ \forall\mathcal{LS}_{2,2}.\varsigma'\ \neg(\varsigma'(C) = \{\varsigma(X^1_{A_1})\}_{\mathsf{ek}(h)}).$$

Fonction	Théorème 1 : $f^\star(\Pi) \models f^\star(\phi)$ implique $\Pi \models \phi$	Théorème 2 : $\Pi \models \phi$ implique $f^\star(\Pi) \models f^\star(\phi)$
$f^\star = \overline{\cdot}^{lhs \to hs}$	−	−
$f^\star = \overline{\cdot}^{hs \to h}$	−	−
$f^\star = \overline{\cdot}^{h \to}$	+	−

FIG. 3.5 – Tableau récapitulatif sur la logique $\mathcal{L}_1^\star$

En effet, la fonction $\overline{\cdot}^{h \to}$ ne change rien de cette propriété. Il est cependant clair, que même une exécution normale du protocole $\overline{\Pi}^{h \to}$ ne satisfait pas la propriété. Nous avons $\overline{\Pi}^{h \to} \not\models \overline{\phi}^{h \to}$. La propriété 2 est donc fausse pour la fonction $\overline{\cdot}^{h \to}$.

3.3 Conclusion sur la logique $\mathcal{L}_1^\star$

Nous avons résumé les résultats que nous avons obtenus sur la logique $\mathcal{L}_1^\star$ dans le tableau de la figure 3.5. Nous constatons que la classe des propriétés qui peuvent être exprimées dans la logique $\mathcal{L}_1^\star$ est trop vaste, pour que les propriétés soient préservées d'un modèle à l'autre. Nous avons marqué ces cas avec un minus (−). Nous avons donné un contre-exemple pour chacun de ces cas.

L'exception à la règle constitue la fonction $\overline{\cdot}^{h \to}$ pour la propriété 1. Nous étions en effet en mesure de montrer le théorème 1 pour cette fonction sous la condition que l'identité d'agent qui est utilisée pour modéliser la fonction de hachage par un chiffrement à clef publique n'apparaisse ni dans une formule, ni dans un protocole ou une trace d'exécution en tant que agent. Nous avons forcé ces restrictions en modifiant le modèle de façon à ce que l'identité d'agent en question y ait été éliminée.

3.4 Amélioration des logiques $\mathcal{L}_1^\star$: les logiques $\mathcal{L}_2^\star$

Nous avons vu dans le paragraphe 3.2 que les logiques $\mathcal{L}_1^\star$ décrivent en général de trop grandes classes de propriétés des protocoles pour permettre le transfert de celles-ci d'un modèle à un autre modèle. L'exception à la règle était la fonction $\overline{\cdot}^{h \to}$ pour la propriété 1. Dans ce paragraphe, nous identifions des parties des logiques $\mathcal{L}_1^\star$, appelées les logiques $\mathcal{L}_2^\star$, telles qu'elles satisfaisons la propriété 1, donnée par :

Propriété 3. *Soit* $\Pi : [k] \to \mathsf{Roles}^\star$ *un protocole à* k *participants et soit* $\phi \in \mathcal{L}_2^\star$ *un propriété de trace. Nous avons* $f^\star(\Pi) \models f^\star(\phi) \implies \Pi \models \phi$.

La propriété complémentaire est donnée par :

Propriété 4. *Soit* $\Pi : [k] \to \mathsf{Roles}^\star$ *un protocole à* k *participants et soit* $\phi \in \mathcal{L}_2^\star$ *un propriété de trace. Nous avons* $\Pi \models \phi \implies f^\star(\Pi) \models f^\star(\phi)$.

Dans ces propriétés, la fonction $f^\star$ est une des fonctions $\overline{\cdot}^{lhs \to hs}$, $\overline{\cdot}^{hs \to h}$ ou $\overline{\cdot}^{h \to}$, selon le modèle dans lequel le protocole Π est spécifié.

3.4.1 Réflexion sur les propriétés des termes utilisés dans les logiques $\mathcal{L}_1^\star$

Comme nous l'avons vu dans le paragraphe 3.2.4, la propriété clef des termes de T_{Sub}^{h-} qui nous a permis de prouver le théorème 1 est le fait que $\overline{\overline{m_1}}^{h\rightarrow\_} = \overline{\overline{m_2}}^{h\rightarrow\_}$ implique $m_1 = m_2$ pour deux termes m_1 et m_2 de cette algèbre. Deux exemples montrent que cette implication n'est pas satisfaite pour les algèbres T_{Sub}^{lhs} et T_{Sub}^{hs}.

Exemple 14. Soient $m_1 = \{b\}_{\mathsf{ek}(a)}^{\mathsf{ag}(1)}$ et $m_2 = \{b\}_{\mathsf{ek}(a)}^{\mathsf{ag}(2)}$ deux termes de l'algèbre T_{Sub}^{lhs} où $a, b \in \mathsf{ID}$. Nous avons $\overline{\overline{m_1}}^{lhs\rightarrow hs} = \{b\}_{\mathsf{ek}(a)} = \overline{\overline{m_2}}^{lhs\rightarrow hs}$. Pourtant, nous avons $m_1 \neq m_2$ à cause des étiquettes $\mathsf{ag}(1)$ et $\mathsf{ag}(2)$.

Exemple 15. Soient $m_1 = [b]_{\mathsf{sk}(a)}$ et $m_2 = \{b\}_{\mathsf{dk}(a)}$ deux termes de l'algèbre T_{Sub}^{hs} avec $a, b \in \mathsf{ID}$. Nous avons $\overline{\overline{m_1}}^{hs\rightarrow h} = \{\overline{b}^{hs\rightarrow h}\}_{\overline{\mathsf{sk}(a)}^{h\rightarrow\_}} = \{b\}_{\mathsf{dk}(a)} = \overline{\overline{m_2}}^{h\rightarrow\_}$. Pourtant, comme dans l'exemple précédent, nous avons $m_1 \neq m_2$.

Étant donné que l'implication $f(m_1) = f(m_2) \implies m_1 = m_2$ est indispensable, ainsi que la seule condition qui manque pour prouver la propriété 1 pour la fonction $f \in \{\overline{\cdot}^{lhs\rightarrow hs}, \overline{\cdot}^{hs\rightarrow h}\}$, nous réfléchissons aux modifications possibles pour l'assurer. En effet, deux possibilités se présentent : soit, nous changeons la définition des fonctions, ce qui serait dommage, parce que nous les avons définies selon les habitudes de codages de vérificateurs. Par conséquent, un tel changement rendrait nos résultats moins intéressants. Soit, nous changeons les définitions des logiques $\mathcal{L}_1^\star$ de façon à ce que nous n'autorisions que des tests d'égalité entre les termes qui satisfont cette implication. Bien que l'expressivité des logiques en soit diminuée, nous optons clairement pour la deuxième possibilité, parce que nos résultats seront plus significatifs en pratique.

La définition suivante présente la notion de terme fixe. Un terme fixe est un terme qui n'est pas transformé en un autre terme par l'application d'une fonction. Après avoir donné un exemple, nous énonçons un lemme simple qui affirme, que les termes fixes satisfont l'implication en question. Soit comme avant $f^\star \in \{\overline{\cdot}^{lhs\rightarrow hs}, \overline{\cdot}^{hs\rightarrow h}, \overline{\cdot}^{h\rightarrow\_}\}$ la fonction correspondante au modèle.

Définition 24. Un terme $u \in T_{Sub}^\star$ est appelé *fixe*, si $u = f^\star(u)$.

Exemple 16. Considérons la fonction $\overline{\cdot}^{lhs\rightarrow hs}$. Soient a et $b \in \mathsf{ID}$ des identités d'agents et j et s des entiers. Les termes a, $n(a, j, s)$, $k(a, j, s)$ sont des termes fixes, ainsi que les clefs $\mathsf{ek}(a)$, $\mathsf{dk}(a)$, $\mathsf{sk}(a)$, $\mathsf{vk}(a)$ et $\mathsf{symk}(a, b)$. Soient m_1 et $m_2 \in T_{Sub}^{lhs}$ des termes fixes. Alors les termes $\langle m_1, m_2\rangle$ et $hash(m_1)$ sont des termes fixes.

Lemme 7. *Soient u_1 et $u_2 \in T_{Sub}^\star$ deux termes fixes. $f^\star(u_1) = f^\star(u_2)$ implique $u_1 = u_2$.*

Preuve. Nous avons $u_1 = f(u_1) = f(u_2) = u_2$. □

3.4.2 Les logiques $\mathcal{L}_2^\star$

Les formules

Nous définissons maintenant les logiques $\mathcal{L}_2^\star$, à savoir $\mathcal{L}_2^{lhs}$, $\mathcal{L}_2^{hs}$, $\mathcal{L}_2^h$ et $\mathcal{L}_2$. La différence principale de nouvelles logiques par rapport aux logiques $\mathcal{L}_1^\star$ est le fait que nous n'autorisons que des tests d'égalité entre des termes fixes. Pour implanter ce changement dans les logiques $\mathcal{L}_2^\star$ nous procédons comme suit :

1. Nous n'autorisons que des tests d'égalité entre des termes fixes.

2. Étant donné que nous voulons garder les tests d'inégalité entre des termes arbitraires m_1 et m_2, jusqu'alors exprimés par des formules comme $\neg(m_1 = m_2)$, mais que la formule $(m_1 = m_2)$ n'est plus autorisée, nous devons introduire un nouveau prédicat $\neq$.
3. Par conséquent, une formule comme $\neg(m_1 \neq m_2)$ reviendrait à un test d'égalité entre deux termes arbitraires m_1 et m_2. Nous supprimons donc la négation des formules arbitraires.
4. Comme nous voulons quand même garder le pouvoir d'exprimer qu'un agent est corrompu, nous introduisons un nouveau prédicat $\neg NC$.

Nous condensons ces changements dans la définition des formules des logiques $\mathcal{L}_2^\star$.

Définition 25. Les *formules* des logiques $\mathcal{L}_2^\star$ sont inductivement définies par

$$\begin{aligned} F \quad ::= \quad & NC(a) \mid \neg NC(a) \mid (u_1 = u_2) \mid (m_1 \neq m_2) \mid F \wedge F \mid F \vee F \\ & \mid \forall \mathcal{LS}_{i,p}.\varsigma\ F \mid \exists \mathcal{LS}_{i,p}.\varsigma\ F \end{aligned}$$

où u_1 et $u_2 \in T^\star_{Sub}$ sont des termes fixes, a, m_1 et $m_2 \in T^\star_{Sub}$ sont des termes arbitraires, i et $p \in \mathbb{N}$ sont des entiers et $\varsigma \in Sub$ est une variable de substitution. Pour deux formules ϕ_1 et $\phi_2 \in \mathcal{L}_2^\star$, nous autorisons aussi la notation $\phi_1 \rightarrow \phi_2$ en tant qu'abréviation pour $\neg\phi_1 \vee \phi_2$.

Nous étendons la fonction $f \in \{ \overset{lhs \rightarrow hs}{\cdot}, \overset{hs \rightarrow h}{\cdot}, \overset{h \rightarrow}{\cdot} \}$ d'une manière naturelle sur les nouvelles formules : notamment, nous posons $f(\neg NC(a)) = \neg NC(f(a))$ et $f((m_1 \neq m_2)) = (f(m_1) \neq f(m_2))$.

L'interprétation

Nous gardons la notion des formules closes, donnée par la définition 19. La définition suivante donne l'interprétation d'une formule close des logiques $\mathcal{L}_2^\star$ pour une trace dans $\mathsf{SymbTr}^\star$.

Définition 26. L'*interprétation* d'une formule close pour une trace est l'application $[\![\_, \_]\!] : \mathcal{L}_2^\star \times \mathsf{SymbTr}^\star \rightarrow \{0, 1\}$, donnée par

1. Le prédicat NC.

$$[\![NC(a), tr]\!] = \begin{cases} 1 & \text{si } a \in \mathsf{ID} \text{ et } a \neq \mathsf{int} \text{ n'apparaît pas dans une transition } \mathbf{corrupt} : \\ & \text{pour } tr = e_1 e_2 \ldots e_n \text{ et } a_1, ..., a_k, \\ & \text{tels que } e_1 \xrightarrow{\mathbf{corrupt}(a_1,...,a_k)} e_2 \text{ nous avons } a \neq a_i \text{ et } a \neq \mathsf{int} \\ 0 & \text{sinon} \end{cases}$$

2. La négation du prédicat NC.

$$[\![\neg NC(a), tr]\!] = \begin{cases} 1 & \text{si } [\![NC(a), tr]\!] = 0 \\ 0 & \text{si } [\![NC(a), tr]\!] = 1 \end{cases}$$

3. Égalité syntaxique entre termes simples.

$$[\![(u_1 = u_2), tr]\!] = \begin{cases} 1 & \text{si } u_1 = u_2 \\ 0 & \text{sinon} \end{cases}$$

4. Inégalité syntaxique.

$$[\![(m_1 \neq m_2), tr]\!] = \begin{cases} 1 & \text{si } m_1 \neq m_2 \\ 0 & \text{sinon} \end{cases}$$

5. La conjonction et la disjonction.

$$\begin{aligned}[\![F_1 \wedge F_2, tr]\!] &= [\![F_1, tr]\!] \wedge [\![F_2, tr]\!] \\ [\![F_1 \vee F_2, tr]\!] &= [\![F_1, tr]\!] \vee [\![F_2, tr]\!]\end{aligned}$$

6. La quantification universelle.

$$[\![\forall \mathcal{LS}_{i,p}.\varsigma\ F, tr]\!] = \begin{cases} 1 & \text{si } \forall(\theta, p) \in \mathcal{LS}_{i,p}(tr), \text{ nous avons } [\![F[\varsigma \leftarrow \theta], tr]\!] = 1, \\ 0 & \text{sinon.} \end{cases}$$

7. La quantification existentielle.

$$[\![\exists \mathcal{LS}_{i,p}.\varsigma\ F, tr]\!] = \begin{cases} 1 & \text{si } \exists(\theta, p) \in \mathcal{LS}_{i,p}(tr), \text{ tel que } [\![F[\varsigma \leftarrow \theta], tr]\!] = 1, \\ 0 & \text{sinon.} \end{cases}$$

Rappelons que nous devons éventuellement appliquer des substitutions avant d'interpréter une formule.

Exemples des propriétés exprimées par des formules des logiques $\mathcal{L}_2^\star$

Les propriétés données dans le paragraphe 2.5.2 sont aussi contenues dans la logique $\mathcal{L}_2^\star$. Nous voyons que les restrictions que nous avons faites par rapport aux logiques $\mathcal{L}_1^\star$ ne sont pas trop contraignantes.

3.5 Les théorèmes de correction pour les logiques $\mathcal{L}_2^\star$

Ce paragraphe prouve la propriété 3 pour les fonctions $\overline{\cdot}^{lhs \to hs}$ et $\overline{\cdot}^{hs \to h}$. Nous rediscutons aussi la fonction $\overline{\cdot}^{h \to}$. Les preuves sont faites à l'instar de la preuve donné au paragraphe 3.2.4. Quant à la propriété 4, nous donnons des contre-exemples qui montrent qu'elle n'est pas satisfaite pour les formules des logiques $\mathcal{L}_2^\star$ non plus.

3.5.1 La fonction $lhs \to hs$ réexaminée

Preuve de la propriété 3

Comme nous avons la notion de terme fixe, nous pouvons procéder toute de suite, sans passer par un lemme auxiliaire comme avant, à la preuve de l'implication énoncée par le lemme suivant :

Lemme 8. *Soit $\phi \in \mathcal{L}_2^{lhs}$ une formule et $tr \in \mathsf{SymbTr}^{lhs}$ une trace. Alors, $[\![\overline{\phi}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$ implique $[\![\phi, tr]\!] = 1$.*

Preuve. La preuve se fait par induction sur la structure des formules.

- $\phi = NC(a)$. Nous avons $[\![NC(a), tr]\!] = 1$ si et seulement si $a \in \mathsf{ID}$ est différent de int et n'apparaît pas dans une transition **corrupt** de la trace tr. Cela signifie que $\overline{a}^{lhs \to hs} \in \mathsf{ID}$ est différent de int et n'apparaît pas dans une transition **corrupt** de la trace $\overline{tr}^{lhs \to hs}$. Nous avons donc $[\![NC(a), tr]\!] = 1$ si et seulement si $[\![\overline{NC(a)}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$. Par conséquent, $[\![NC(a), tr]\!] = 0$ si et seulement si $[\![\overline{NC(a)}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 0$ et donc $[\![\neg NC(a), tr]\!] = 1$ si et seulement si $[\![\overline{\neg NC(a)}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$. Nous avons donc traité en même temps le cas où $\phi = \neg NC(a)$.

- $\phi = (u_1 = u_2)$ où u_1 et u_2 sont des termes fixes. Supposons que $[\![\overline{(u_1 = u_2)}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = [\![(\overline{u_1}^{lhs \to hs} = \overline{u_2}^{lhs \to hs}), \overline{tr}^{lhs \to hs}]\!] = 1$. Cela est le cas, si et seulement si $\overline{u_1}^{lhs \to hs} = \overline{u_2}^{lhs \to hs}$. Étant donné que u_1 et u_2 sont des termes fixes, $\overline{u_1}^{lhs \to hs} = \overline{u_2}^{lhs \to hs}$ implique $u_1 = u_2$. Par conséquence $[\![(u_1 = u_2), tr]\!] = 1$.
- $\phi = (m_1 \neq m_2)$ où m_1 et $m_2 \in T^{lhs}$. Supposons $[\![\overline{(m_1 \neq m_2)}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = [\![(\overline{m_1}^{lhs \to hs} \neq \overline{m_2}^{lhs \to hs}), \overline{tr}^{lhs \to hs}]\!] = 1$. Nous avons $\overline{m_1}^{lhs \to hs} \neq \overline{m_2}^{lhs \to hs}$. Supposons que $m_1 = m_2$. Dans ce cas $\overline{m_1}^{lhs \to hs} = \overline{m_2}^{lhs \to hs}$, ce qui est une contradiction. Par conséquence, nous avons $m_1 \neq m_2$ et donc $[\![(m_1 \neq m_2), tr]\!] = 1$.
- $\phi = \phi_1 \wedge \phi_2$. Supposons $[\![\overline{\phi_1 \wedge \phi_2}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = [\![\overline{\phi_1}^{lhs \to hs} \wedge \overline{\phi_2}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$. Nous avons $[\![\overline{\phi_1}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$ et $[\![\overline{\phi_2}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$. Par hypothèse d'induction, $[\![\phi_1, tr]\!] = 1$ et $[\![\phi_2, tr]\!] = 1$, donc $[\![\phi_1 \wedge \phi_2, tr]\!] = 1$.
- $\phi = \phi_1 \vee \phi_2$. Supposons que $[\![\overline{\phi_1 \vee \phi_2}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = [\![\overline{\phi_1}^{lhs \to hs} \vee \overline{\phi_2}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$. Nous avons $[\![\overline{\phi_1}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$ ou $[\![\overline{\phi_2}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$. Par hypothèse d'induction, $[\![\phi_1, tr]\!] = 1$ ou $[\![\phi_2, tr]\!] = 1$, donc $[\![\phi_1 \vee \phi_2, tr]\!] = 1$.
- $\phi = \forall \mathcal{LS}_{i,p}.\varsigma\ \phi'$. Supposons que $[\![\overline{\forall \mathcal{LS}_{i,p}.\varsigma\ \phi'}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = [\![\forall \mathcal{LS}_{i,p}.\varsigma\ \overline{\phi'}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$. $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$ implique $(\overline{\theta}^{lhs \to hs}, p) \in \mathcal{LS}_{i,p}(\overline{tr}^{lhs \to hs})$. Pour tout état local $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$ nous avons donc $[\![\overline{\phi'}^{lhs \to hs}[\varsigma \leftarrow \overline{\theta}^{lhs \to hs}], \overline{tr}^{lhs \to hs}]\!] = 1$. Par hypothèse d'induction, $[\![\overline{\phi'}^{lhs \to hs}[\varsigma \leftarrow \overline{\theta}^{lhs \to hs}], \overline{tr}^{lhs \to hs}]\!] = [\![\overline{\phi'[\varsigma \leftarrow \theta]}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$ implique $[\![\phi'[\varsigma \leftarrow \theta], tr]\!] = 1$. Nous avons donc $[\![\forall \mathcal{LS}_{i,p}.\varsigma\ \phi', tr]\!] = 1$.
- $\phi = \exists \mathcal{LS}_{i,p}.\varsigma\ \phi'$. Supposons que $[\![\overline{\exists \mathcal{LS}_{i,p}.\varsigma\ \phi'}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = [\![\exists \mathcal{LS}_{i,p}.\varsigma\ \overline{\phi'}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$. $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$ implique $(\overline{\theta}^{lhs \to hs}, p) \in \mathcal{LS}_{i,p}(\overline{tr}^{lhs \to hs})$. Il existe donc un état local $(\theta, p) \in \mathcal{LS}_{i,p}(tr)$, tel que $[\![\overline{\phi'}^{lhs \to hs}[\varsigma \leftarrow \overline{\theta}^{lhs \to hs}], \overline{tr}^{lhs \to hs}]\!] = 1$. Par hypothèse d'induction, $[\![\overline{\phi'}^{lhs \to hs}[\varsigma \leftarrow \overline{\theta}^{lhs \to hs}], \overline{tr}^{lhs \to hs}]\!] = [\![\overline{\phi'[\varsigma \leftarrow \theta]}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$ implique $[\![\phi'[\varsigma \leftarrow \theta], tr]\!] = 1$. Nous concluons que $[\![\exists \mathcal{LS}_{i,p}.\varsigma\ \phi', tr]\!] = 1$. □

Remarquons que nous ne pouvons pas prouver l'implication inverse, à savoir $[\![\phi, tr]\!] = 1 \implies [\![\overline{\phi}^{lhs \to hs}, \overline{tr}^{lhs \to hs}]\!] = 1$, comme l'exemple du lemme 2 pourrait faire croire. La raison se trouve dans la preuve pour la formule $\phi = (m_1 \neq m_2)$ où m_1 et m_2 sont des termes arbitraires. Supposons que nous avons $[\![\overline{m_1}^{lhs \to hs} \neq \overline{m_2}^{lhs \to hs}, tr]\!] = 0$. Nous avons donc $\overline{m_1}^{lhs \to hs} = \overline{m_2}^{lhs \to hs}$. Étant donné que m_1 et m_2 ne sont pas des termes fixes, nous ne pouvons pas conclure que $m_1 = m_2$. Cela peut être vrai ou non. Néanmoins, la direction de l'implication démontrée suffit pour prouve la propriété 3 correcte pour la fonction $\overline{\cdot}^{lhs \to hs}$.

Soit $\Pi : [k] \to \mathsf{Roles}^{lhs}$ un protocole à k participants. Nous procédons maintenant à la preuve que $tr \in \mathsf{Exec}(\Pi)$ implique $\overline{tr}^{lhs \to hs} \in \mathsf{Exec}(\overline{\Pi}^{lhs \to hs})$. Nous avons d'abord besoin de trois lemmes auxiliaires.

Lemme 9. *Soit σ une substitution et $m \in T^{lhs}$ un terme. Nous avons $\overline{m\sigma}^{lhs \to hs} = \overline{m}^{lhs \to hs}\overline{\sigma}^{lhs \to hs}$.*

Preuve. La preuve se fait par induction sur la structure des termes. Nous pouvons adapté la preuve du lemme 3 où nous remplaçons la fonction $\overline{\cdot}^{h \to}$ par la fonction $\overline{\cdot}^{lhs \to hs}$. Nous rajoutons les cas manquants :

- Les cas de base.
  - Soit $m \in X^{hs}$ une variable. Alors, nous avons $\overline{m\sigma}^{lhs\to hs} = m\overline{\sigma}^{lhs\to hs} = \overline{m}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}$.
- L'étape d'induction.
  - Soit maintenant $m = \mathsf{sk}(A)$ ou $m = \mathsf{vk}(A)$ où $A \in X_a^{\mathsf{ID}}$ est une identité d'agent ou une variable de ce type. Nous avons $\overline{\mathsf{sk}(A)\sigma}^{lhs\to hs} = \overline{\mathsf{sk}(A\sigma)}^{lhs\to hs} = \mathsf{sk}(\overline{A\sigma}^{lhs\to hs}) = \mathsf{sk}(\overline{A}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}) = \mathsf{sk}(\overline{A}^{lhs\to hs})\overline{\sigma}^{lhs\to hs} = \overline{\mathsf{sk}(A)}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}$. Nous avons le même raisonnement pour $\mathsf{vk}(A)$.
  - Nous ajoutons le cas de la signature. Soit $m = [m']^l_{\mathsf{sk}(a)}$. $\overline{m\sigma}^{lhs\to hs} = \overline{[m']^l_{\mathsf{sk}(a)}\sigma}^{lhs\to hs} = \overline{[m'\sigma]^l_{\mathsf{sk}(a)\sigma}}^{lhs\to hs} = [\overline{m'\sigma}^{lhs\to hs}]_{\overline{\mathsf{sk}(a)\sigma}^{lhs\to hs}} = [\overline{m'}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}]_{\overline{\mathsf{sk}(a)}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}} = [\overline{m'}^{lhs\to hs}]_{\overline{\mathsf{sk}(a)}^{lhs\to hs}}\overline{\sigma}^{lhs\to hs} = \overline{[m']^l_{\mathsf{sk}(a)}}^{lhs\to hs}\overline{\sigma}^{lhs\to hs} = \overline{m}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}$.
  - Nous regardons à nouveau le cas de la fonction de hachage. Soit $m = hash(m')$. Nous avons $\overline{m\sigma}^{lhs\to hs} = \overline{hash(m')\sigma}^{lhs\to hs} = \overline{hash(m'\sigma)}^{lhs\to hs} = hash(\overline{m'\sigma}^{lhs\to hs}) = hash(\overline{m'}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}) = hash(\overline{m'}^{lhs\to hs})\overline{\sigma}^{lhs\to hs} = \overline{hash(m')}^{lhs\to hs}\overline{\sigma}^{lhs\to hs} = \overline{m}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}$. □

Lemme 10. *Soient t et m des termes dans T^{lhs}, tels que $m = t\sigma$. Il existe une substitution σ', telle que $\overline{m}^{lhs\to hs} = \overline{t}^{lhs\to hs}\sigma'$.*

Preuve. Nous avons $\overline{m}^{lhs\to hs} = \overline{t\sigma}^{lhs\to hs} = \overline{t}^{lhs\to hs}\overline{\sigma}^{lhs\to hs}$. Nous choisissons $\sigma' = \overline{\sigma}^{lhs\to hs}$. □

Lemme 11. *Soit $S \subseteq T^{lhs}$. $S \vdash^{lhs} m$ implique $\overline{S}^{lhs\to hs} \vdash^{hs} \overline{m}^{lhs\to hs}$.*

Preuve. La preuve se fait sur le nombre d'applications des règles pour déduire m à partir de S. Les systèmes de déduction se trouvent dans les figures 2.2 et 2.3. Nous pouvons réutiliser une grande partie de la preuve du lemme 5 où nous remplaçons respectivement la fonction $\overline{\cdot}^{h\to}$ par la fonction $\overline{\cdot}^{lhs\to hs}$ et les relations $\vdash^h$ et $\vdash$ par les relations $\vdash^{lhs}$ et $\vdash^{hs}$.

Nous ajoutons les cas de la signature et modifions le cas de la fonction de hachage. Le cas des fonctions des chiffrements sont similaires au cas de la signature.

- La règle $\mathcal{RLHS}^{sign}_{sig}$.
  Le terme $m = [m']^l_{\mathsf{sk}(a)}$ a été déduit en appliquant la règle $\mathcal{RLHS}^{sign}_{sig}$ comme dernière règle. Nous avons donc $S \vdash^{lhs} \mathsf{sk}(a)$ et $S \vdash^{lhs} m'$. Par hypothèse d'induction, nous avons $\overline{S}^{lhs\to hs} \vdash^{hs} \overline{\mathsf{sk}(a)}^{lhs\to hs} = \mathsf{sk}(a)$ et $\overline{S}^{lhs\to hs} \vdash^{hs} \overline{m'}^{lhs\to hs}$. Nous appliquons la règle $\mathcal{RHS}^{sign}_{sig}$ et obtenons $\overline{S}^{lhs\to hs} \vdash^{hs} [\overline{m'}^{lhs\to hs}]_{\mathsf{sk}(a)} = [\overline{m'}^{lhs\to hs}]_{\overline{\mathsf{sk}(a)}^{lhs\to hs}} = \overline{[m']^l_{\mathsf{sk}(a)}}^{lhs\to hs} = \overline{m}^{lhs\to hs}$.
- La règle $\mathcal{RLHS}^{sign}_{ded}$.
  Le terme m a été déduit à partir d'un terme $[m]^l_{\mathsf{sk}(a)}$ en appliquant la règle $\mathcal{RLHS}^{sign}_{ded}$ comme dernière règle. Nous avons donc $S \vdash^{lhs} [m]^l_{\mathsf{sk}(a)}$ et par hypothèse d'induction $\overline{S}^{lhs\to hs} \vdash^{lhs} \overline{[m]^l_{\mathsf{sk}(a)}}^{lhs\to hs} = [\overline{m}^{lhs\to hs}]_{\overline{\mathsf{sk}(a)}^{lhs\to hs}} = [\overline{m}^{lhs\to hs}]_{\mathsf{sk}(a)}$. Nous utilisons la règle $\mathcal{RHS}^{sign}_{ded}$ et obtenons $\overline{S}^{lhs\to hs} \vdash^{hs} \overline{m}^{lhs\to hs}$.
- La règle $\mathcal{RLHS}^{hash}$.

Soit $m = hash(m')$ et la dernière règle utilisée pour déduire m est la règle $\mathcal{RLHS}^{hash}$. Nous avons donc $S \vdash^{lhs} m'$. Par hypothèse, $\frac{S}{S}^{lhs\to hs} \vdash^{hs} \frac{m'}{m'}^{lhs\to hs}$ et alors $\frac{S}{S}^{lhs\to hs} \vdash^{hs} hash(\frac{m'}{m'}^{lhs\to hs}) = \frac{hash(m')}{hash(m')}^{lhs\to hs} = \frac{m}{m}^{lhs\to hs}$ avec la règle $\mathcal{RHS}^{hash}$. □

Lemme 12. *Soit* $\Pi : [k] \to \mathsf{Roles}^{lhs}$ *un protocole.* $tr \in \mathsf{Exec}^{lhs}(\Pi)$ *implique* $\frac{tr}{tr}^{lhs\to hs} \in \mathsf{Exec}^{hs}(\frac{\Pi}{\Pi}^{lhs\to hs})$.

Preuve. La preuve se fait par induction sur la longueur de trace.

- Cas de base : nous considérons les traces de longueur 1.
  Soit $(\mathsf{SId}_1, f_1, H_1) \in \mathsf{Exec}^{lhs}(\Pi)$ une trace d'exécution valide de longueur 1. Nous avons $\mathsf{SId}_1 = \emptyset$, la fonction f_1 n'est définie nulle part et $H_1 = \mathbf{kn}^{lhs}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}$. Considérons la trace $(\mathsf{SId}_1, \frac{f_1}{f_1}^{lhs\to hs}, \frac{H_1}{H_1}^{lhs\to hs})$. La fonction $\frac{f_1}{f_1}^{lhs\to hs}$ n'est définie nulle part. En outre, nous avons $\frac{H_1}{H_1}^{lhs\to hs} = \frac{\mathbf{kn}^{lhs}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}}{\mathbf{kn}^{lhs}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}}^{lhs\to hs} = \mathbf{kn}^{hs}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}$. Nous concluons que $\frac{tr}{tr}^{lhs\to hs} \in \mathsf{Exec}(\frac{\Pi}{\Pi}^{lhs\to hs})$.
- L'étape d'induction : nous considérons les traces de longueur $n+1$.
  Soit $tr = (\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n+1} \in \mathsf{Exec}^{lhs}(\Pi)$ une trace d'exécution valide de longueur $n+1$. Nous avons par hypothèse d'induction que

  $$(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n} \in \mathsf{Exec}^{lhs}(\Pi)$$

  implique

  $$\frac{(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n}}{(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n}}^{lhs\to hs} \in \mathsf{Exec}^{hs}(\frac{\Pi}{\Pi}^{lhs\to hs}).$$

  Trois cas sont possibles pour la n-ième transition de la trace tr :

  1. $\mathbf{trans}_n = \mathbf{corrupt}(a_1, \ldots, a_l)$.
     Étant donné que cette type de transition est seulement autorisé pour la première transition, nous avons $tr = (\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}_2, f_2, H_2)$, où $\mathsf{SId}_2 = \mathsf{SId}_1$, $f_2 = f_1$ et $H_2 = H_1 \cup \bigcup_{1\le i\le l} \mathbf{kn}^{lhs}(a_i)$.
     Considérons la trace $\frac{tr}{tr}^{lhs\to hs} = \frac{(\mathsf{SId}_1, f_1, H_1)}{(\mathsf{SId}_1, f_1, H_1)}^{lhs\to hs} \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}_2, \frac{f_2}{f_2}^{lhs\to hs}, \frac{H_2}{H_2}^{lhs\to hs})$.
     Nous avons $\mathsf{SId}_1 = \mathsf{SId}_2$, $\frac{f_2}{f_2}^{lhs\to hs} = \frac{f_1}{f_1}^{lhs\to hs}$ et $\frac{H_1}{H_1}^{lhs\to hs} \cup \bigcup_{1\le i\le l} \mathbf{kn}^{hs}(a_i) = \frac{H_1 \cup \bigcup_{1\le i\le l} \mathbf{kn}^{lhs}(a_i)}{H_1 \cup \bigcup_{1\le i\le l} \mathbf{kn}^{lhs}(a_i)}^{lhs\to hs} = \frac{H_2}{H_2}^{lhs\to hs}$. Nous concluons que $\frac{tr}{tr}^{lhs\to hs} \in \mathsf{Exec}(\frac{\Pi}{\Pi}^{lhs\to hs})$
  2. $\mathbf{trans}_n = \mathbf{new}(i, a_1, \ldots, a_k)$.
     Nous avons donc $tr = \ldots(\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}_n \cup \{\mathsf{sid}\}, f_{n+1}, H_{n+1})$ où $\mathsf{sid} = (|\mathsf{SId}_n| + 1, i, (a_1, \ldots, a_k))$. Nous avons $f_{n+1}(\mathsf{sid}) = (\sigma, 1)$ et $f_{n+1}(s) = f_n(s)$ pour tout identifiant de session $s \in \mathsf{SId}_n$. Nous avons $H_{n+1} = H_n$.
     Considérons la trace $\frac{tr}{tr}^{lhs\to hs} = \frac{\ldots(\mathsf{SId}_n, f_n, H_n)}{\ldots(\mathsf{SId}_n, f_n, H_n)}^{lhs\to hs} \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}_n \cup \{\mathsf{sid}\}, \frac{f_{n+1}}{f_{n+1}}^{lhs\to hs}, \frac{H_{n+1}}{H_{n+1}}^{lhs\to hs})$. Pour tout identifiant de session $s \in \mathsf{SId}_{n-1}$, nous avons $\frac{f_n}{f_n}^{lhs\to hs}(s) = \frac{f_{n-1}}{f_{n-1}}^{lhs\to hs}(s)$. Nous considérons l'état local $(\sigma', 1)$ qui est affecté à l'identifiant de session sid. Le domaine de la substitution σ' ne contient que des variables des types agent, nonce et clef de chiffrement à court terme. Étant donné que l'identifiant de cette session est identique à l'identifiant qui apparaît dans la trace tr, les domaines des substitutions σ' et σ sont identiques et aussi les termes qui sont affectes aux variables de ce domaine. Nous avons donc $\sigma' = \sigma$. Vu les types des variables nous avons

aussi $\sigma = \overline{\sigma}^{lhs\to hs}$ et donc $\sigma' = \overline{\sigma}^{lhs\to hs}$. Par conséquent, $(\sigma', p) = (\overline{\sigma}^{lhs\to hs}, p) = \overline{f_{n+1}}^{lhs\to hs}(\mathsf{sid})$. Finalement, nous avons $\overline{H_{n+1}}^{lhs\to hs} = \overline{H_n}^{lhs\to hs}$ et donc $\overline{tr}^{lhs\to hs} \in \mathsf{Exec}(\overline{\Pi}^{lhs\to hs})$.

3. $\mathbf{trans}_n = \mathbf{send}(\mathsf{sid}, m)$.

   Nous avons donc $tr = \ldots(\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})$. Nous avons $\mathsf{SId}_{n+1} = \mathsf{SId}_n$ et $f_{n+1}(s) = f_n(s)$ pour tout identifiant de session $s \in \mathsf{SId}_n$, tel que $s \neq \mathsf{sid}$.

   Considérons la trace $\overline{tr}^{lhs\to hs} = \overline{\ldots(\mathsf{SId}_n, f_n, H_n)}^{lhs\to hs} \xrightarrow{\mathbf{send}(\mathsf{sid},m')} (\mathsf{SId}_{n+1}, \overline{f_{n+1}}^{lhs\to hs}, \overline{H_{n+1}}^{lhs\to hs})$ où le terme m' est donné dans la définition de la fonction $\overline{\cdot}^{lhs\to hs}$. Nous avons $\overline{f_{n+1}}^{lhs\to hs}(s) = \overline{f_n}^{lhs\to hs}(s)$ pour tout identifiant de session $s \neq \mathsf{sid}$ de SId_n.

   Nous supposons que $\mathsf{sid} = (s, j, (a_1, \ldots, a_k))$ et que $\Pi(j) = (l_i, r_i)_{1\leq i\leq l}$ est le rôle joué dans cette session. L'état de cette session avant la transition est donné par $\overline{f_n}^{lhs\to hs}(\mathsf{sid}) = (\overline{\sigma}^{lhs\to hs}, p)$.

   Nous devons distinguer deux cas :

   - Nous avons $p \leq l$ et il existe un unificateur θ tel que $m = l_p\sigma\theta$. Nous avons $f_{n+1}(\mathsf{sid}) = (\sigma \cup \theta, p+1)$ et $H_{n+1} = H_n \cup \{r_p\sigma\theta\}$.

     Dans ce cas, nous avons $m' = \overline{m}^{lhs\to hs}$, parce que le terme l_p est défini et m et $l_p\sigma$ sont unifiables. Le lemme 11 assure que $\overline{m}^{lhs\to hs}$ est déductible à partir de $\overline{H_n}^{lhs\to hs}$, car m était déductible à partir de H_n. Le lemme 10 dit que les termes $\overline{m}^{lhs\to hs}$ et $\overline{l_p\sigma}^{lhs\to hs}$ sont unifiables avec $\overline{\theta}^{lhs\to hs}$ comme unificateur. Le nouveau état local affecté à sid est donc $(\overline{\sigma}^{lhs\to hs} \cup \overline{\theta}^{lhs\to hs}, p+1) = \overline{f_{n+1}}^{lhs\to hs}(\mathsf{sid})$ et, par conséquent, $\overline{H_n}^{lhs\to hs} \cup \{\overline{r_p\sigma\theta}^{lhs\to hs}\} = \overline{H_{n+1}}^{lhs\to hs}$. Nous concluons que $\overline{tr}^{lhs\to hs} \in \mathsf{Exec}^{hs}(\overline{\Pi}^{lhs\to hs})$.
   - Soit nous avons $f_{n+1} = f_n$ et $H_{n+1} = H_n$. Deux cas sont possibles :
     - Soit, $p > l$. Le terme l_p n'est pas défini et nous avons $m' = a$ pour une identité d'agent $a \in \mathsf{ID}$. Ce terme est déductible à partir de $\overline{H_n}^{lhs\to hs}$ selon le système de déduction. Nous avons $\overline{f_{n+1}}^{lhs\to hs}(\mathsf{sid}) = \overline{f_n}^{lhs\to hs}(\mathsf{sid})$ et $\overline{H_{n+1}}^{lhs\to hs} = \overline{H_n}^{lhs\to hs}$. Nous concluons que $\overline{tr}^{lhs\to hs} \in \mathsf{Exec}^{hs}(\overline{\Pi}^{lhs\to hs})$.
     - Soit, $p \leq l$ et les termes m et $l_p\sigma$ ne sont pas unifiables. Nous devons encore distinguer deux cas :
       - Si les termes $\overline{m}^{lhs\to hs}$ et $\overline{l_p\sigma}^{lhs\to hs}$ ne sont pas unifiables, nous avons $m' = \overline{m}^{lhs\to hs}$. Le lemme 11 assure que le terme m' est déductible à partir de l'ensemble $\overline{H_n}^{lhs\to hs}$. Nous avons $\overline{f_{n+1}}^{lhs\to hs}(\mathsf{sid}) = \overline{f_n}^{lhs\to hs}(\mathsf{sid})$ et $\overline{H_{n+1}}^{lhs\to hs} = \overline{H_n}^{lhs\to hs}$ et donc $\overline{tr}^{lhs\to hs} \in \mathsf{Exec}(\overline{\Pi}^{lhs\to hs})$.
       - Si les termes $\overline{m}^{lhs\to hs}$ et $\overline{l_p\sigma}^{lhs\to hs}$ sont unifiables, nous avons $m' = a$ pour une identité d'agent $a \in \mathsf{ID}$, ce qui est déductible à partir de $\overline{H_n}^{lhs\to hs}$. Dans ce cas, le terme m' ne s'unifie pas avec le terme $\overline{l_p\sigma}^{lhs\to hs}$, parce que ce dernier contient un sous-terme du type chiffrement symétrique ou asymétrique ou signature. Nous prouvons cette affirmation par contradiction : supposons que $\overline{l_p\sigma}^{lhs\to hs}$ ne contient pas de sous-terme du type chiffrement symétrique ou asymétrique ou

signature. Dans ce cas, le terme $l_p\sigma$ ne contient pas d'étiquette. Il est donc fixe et nous avons $l_p\sigma = \overline{l_p\sigma}^{lhs\to hs}$. Étant donné qu'il existe un unificateur θ, tel que $\overline{m}^{lhs\to hs} = \overline{l_p\sigma}^{lhs\to hs}\theta$, le terme $\overline{m}^{lhs\to hs}$ ne contient pas de sous-terme du type chiffrement symétrique ou asymétrique ou signature non plus. Le terme m est donc fixe aussi et nous avons $m = \overline{m}^{lhs\to hs}$. Dans ce cas, nous avons $m = l_p\sigma\theta$. Il existe donc un unificateur pour les termes m et $l_p\sigma$, ce qui est une contradiction. Par conséquent, le terme $\overline{l_p\sigma}^{lhs\to hs}$ contient un sous-terme du type chiffrement symétrique ou asymétrique ou signature et ne s'unifie donc pas avec le terme $m' = a$. Nous avons donc $\overline{f_{n+1}}^{lhs\to hs}(\mathsf{sid}) = \overline{f_n}^{lhs\to hs}(\mathsf{sid})$ et $\overline{H_{n+1}}^{lhs\to hs} = \overline{H_n}^{lhs\to hs}$. Il s'ensuit que $\overline{tr}^{lhs\to hs} \in \mathsf{Exec}^{hs}(\overline{\Pi}^{lhs\to hs})$. □

Les lemmes 8 et 12 nous permettent de prouver la propriété 3 pour la fonction $\overline{\cdot}^{lhs\to hs}$.

Théorème 2. *Soit* $\Pi : [k] \to \mathsf{Roles}^{lhs}$ *un protocole à k participants et soit* $\phi \in \mathcal{L}_2^{lhs}$ *une formule. Nous avons* $\overline{\Pi}^{lhs\to hs} \models \overline{\phi}^{lhs\to hs} \implies \Pi \models \phi$.

Preuve. Nous prouvons que $\mathsf{Exec}^{hs}(\overline{\Pi}^{lhs\to hs}) \subseteq \mathsf{Tr}^{hs}(\overline{\phi}^{lhs\to hs})$ implique $\mathsf{Exec}^{lhs}(\Pi) \subseteq \mathsf{Tr}^{lhs}(\phi)$. Soit $tr \in \mathsf{Exec}^{lhs}(\Pi)$ une trace d'exécution. Par le lemme 12, nous avons $\overline{tr}^{lhs\to hs} \in \mathsf{Exec}^{hs}(\overline{\Pi}^{lhs\to hs})$, donc $\overline{tr}^{lhs\to hs} \in \mathsf{Tr}^{hs}(\overline{\phi}^{lhs\to hs})$. Cela signifie que $[\![\overline{\phi}^{lhs\to hs}, \overline{tr}^{lhs\to hs}]\!] = 1$. En appliquant le lemme 8, nous obtenons $[\![\phi, tr]\!] = 1$ et donc $tr \in \mathsf{Tr}^{lhs}(\phi)$. □

Contre-exemple pour la propriété 4

Considérons le protocole $\Pi : [2] \to \mathsf{Roles}^{lhs}$:

$$\begin{aligned}\Pi(1) &= (\mathsf{init}, \{X^1_{A_1}\}^{\mathsf{ag}(1)}_{\mathsf{ek}(A_2)})\\ \Pi(2) &= (\{X^1_{A_1}\}^{\mathsf{ag}(2)}_{\mathsf{ek}(A_2)}, \mathsf{stop}).\end{aligned}$$

Le premier rôle émet un nonce chiffré avec la clef publique de l'agent qui joue le deuxième rôle. Le deuxième rôle attend un message de cette forme. Ce protocole est quelque peu artificiel, parce que le nonce attendu par le deuxième rôle doit être chiffré avec une étiquette fixée, à savoir $\mathsf{ag}(2)$. Bien qu'il ne soit pas très réaliste d'attendre un message chiffré avec un aléa fixé, notre formalisme autorise néanmoins ce scénario. Nous considérons la propriété suivante :

$$\phi = \forall \mathcal{LS}_{1,2}.\varsigma\ \forall \mathcal{LS}_{2,2}.\varsigma'\ (\varsigma(X^1_{A_1}) \neq \varsigma'(X^1_{A_1})).$$

Cette formule dit que le nonce chiffré, obtenu par le deuxième rôle est, toujours différent du nonce chiffré, émis par le premier rôle. Intuitivement, cette formule est vraie, parce que le premier rôle chiffre avec l'étiquette $\mathsf{ag}(1)$, tandis que le deuxième rôle attend un nonce chiffré avec l'étiquette $\mathsf{ag}(2)$. Nous avons donc $\Pi \models \phi$. D'autre part, si nous passons au modèle sans étiquettes, il est clair que même une exécution normale rend la formule ϕ fausse. Nous concluons que la propriété 4 n'est pas satisfaite par la fonction $\overline{\cdot}^{lhs\to hs}$.

3.5.2 La fonction $hs \to h$ réexaminée

Nous prouvons la propriété 3 et donnons des contre-exemples pour la propriété 4.

Preuve de la propriété 3

Lemme 13. *Soit $\phi \in \mathcal{L}_2^{hs}$ une formule et $tr \in \mathsf{SymbTr}^{hs}$ une trace. Alors, $[\![\overline{\phi}^{hs\to h}, \overline{tr}^{hs\to h}]\!] = 1$ implique $[\![\phi, tr]\!] = 1$.*

Preuve. La preuve est analogue à la preuve du lemme 13. □

Lemme 14. *Soit σ une substitution et $m \in T^{hs}$ un terme. Nous avons $\overline{m\sigma}^{hs\to h} = \overline{m}^{hs\to h}\overline{\sigma}^{hs\to h}$.*

Preuve. La preuve se fait par induction sur la structure des termes. La plus grande partie de la preuve est analogue à la preuve du lemme 9 où nous remplaçons la fonction $\overline{\cdot}^{lhs\to hs}$ par la fonction $\overline{\cdot}^{hs\to h}$. Les cas modifiés se reportent à la signature. Nous ne traitons que ces cas :

- Les cas de base.
  - Soit $m \in X^h$ une variable. Alors, nous avons $\overline{m\sigma}^{hs\to h} = m\overline{\sigma}^{hs\to h} = \overline{m}^{hs\to h}\overline{\sigma}^{hs\to h}$.
  - Soit $m \in X.s$ une variable de signature. Supposons que $\overline{m}^{hs\to h} = x$ pour une variable $x \in X.c$. Nous avons $\overline{m\sigma}^{hs\to h} = x\overline{\sigma}^{hs\to h} = \overline{m}^{hs\to h}\overline{\sigma}^{hs\to h}$.
- L'étape d'induction.
  - Soit $m = [m']_{\mathsf{sk}(a)}$. Nous avons $\overline{m\sigma}^{hs\to h} = \overline{[m'\sigma]_{\mathsf{sk}(a)\sigma}}^{hs\to h} = \{\overline{m'\sigma}^{hs\to h}\}_{\overline{\mathsf{sk}(a)\sigma}^{hs\to h}} = \{\overline{m'}^{hs\to h}\overline{\sigma}^{hs\to h}\}_{\overline{\mathsf{sk}(a)}^{hs\to h}\overline{\sigma}^{hs\to h}} = \{\overline{m'}^{hs\to h}\}_{\overline{\mathsf{sk}(a)}^{hs\to h}}\overline{\sigma}^{hs\to h} = \overline{[m']_{\mathsf{sk}(a)}}^{hs\to h}\overline{\sigma}^{hs\to h} = \overline{m}^{hs\to h}\overline{\sigma}^{hs\to h}$. □

Lemme 15. *Soient t et m des termes dans T^{hs}, tels que $m = t\sigma$. Il existe une substitution σ', telle que $\overline{m}^{hs\to h} = \overline{t}^{hs\to h}\sigma'$.*

Preuve. Nous avons $\overline{m}^{hs\to h} = \overline{t\sigma}^{hs\to h} = \overline{t}^{hs\to h}\overline{\sigma}^{hs\to h}$. Nous choisissons $\sigma' = \overline{\sigma}^{hs\to h}$. □

Lemme 16. *Soit $S \subseteq T^{hs}$. $S \vdash^{hs} m$ implique $\overline{S}^{hs\to h} \vdash^h \overline{m}^{hs\to h}$.*

Preuve. La preuve se fait sur le nombre d'applications des règles pour déduire m à partir de S. Les systèmes de déduction se trouvent dans les figures 2.3 et 2.4. Nous pouvons réutiliser une grande partie de la preuve du lemme 11 où nous remplaçons respectivement la fonction $\overline{\cdot}^{lhs\to hs}$ par la fonction $\overline{\cdot}^{hs\to h}$ et les relations $\vdash^{lhs}$ et $\vdash^{hs}$ par les relations $\vdash^{hs}$ et $\vdash^h$. Nous modifions seulement le cas de la signature, concernant l'application des règles $\mathcal{RHS}_{sig}^{sign}$ et $\mathcal{RHS}_{ded}^{sign}$ comme dernières règles.

- La règle $\mathcal{RHS}_{sig}^{sign}$.
  Le terme $m = [m']_{\mathsf{sk}(a)}$ a été déduit en appliquant $\mathcal{RHS}_{sig}^{sign}$ comme dernière règle. Nous avons $S \vdash^{hs} \mathsf{sk}(a)$ et $S \vdash^{hs} m'$. Par hypothèse d'induction, nous avons $\overline{S}^{hs\to h} \vdash^h \overline{\mathsf{sk}(a)}^{hs\to h} = \mathsf{dk}(a)$ et $\overline{S}^{hs\to h} \vdash^h \overline{m'}^{hs\to h}$. En utilisant la règle $\mathcal{RH}_{cinv}^{asym}$, nous obtenons $\overline{S}^{hs\to h} \vdash^h \{\overline{m'}^{hs\to h}\}_{\mathsf{dk}(a)} = \{\overline{m'}^{hs\to h}\}_{\overline{\mathsf{sk}(a)}^{h\to\_}} = \overline{[m']_{\mathsf{sk}(a)}}^{hs\to h} = \overline{m}^{hs\to h}$.
- La règle $\mathcal{RHS}_{ded}^{sign}$.
  Le terme m a été déduit à partir d'un terme $[m]_{\mathsf{sk}(a)}$ en appliquant la règle $\mathcal{RHS}_{ded}^{sign}$ comme dernière règle. Par hypothèse d'induction, nous avons $\overline{S}^{hs\to h} \vdash^h \overline{[m]_{\mathsf{sk}(a)}}^{hs\to h} = \{\overline{m}^{hs\to h}\}_{\overline{\mathsf{sk}(a)}^{hs\to h}} = \{\overline{m}^{hs\to h}\}_{\mathsf{dk}(a)}$. Comme nous avons toujours $\overline{S}^{hs\to h} \vdash^h \mathsf{ek}(a)$ par la règle $\mathcal{R}_2^{init}$, nous pouvons utiliser la règle $\mathcal{R}_{dinv}^{asym}$. Nous obtenons alors $\overline{S}^{hs\to h} \vdash^h \overline{m}^{hs\to h}$. □

Lemme 17. *Soit* $\Pi : [k] \to \mathsf{Roles}^{hs}$ *un protocole.* $tr \in \mathsf{Exec}^{hs}(\Pi)$ *implique* $\overline{tr}^{hs\to h} \in \mathsf{Exec}^{h}(\overline{\Pi}^{hs\to h})$.

Preuve. La preuve se fait par induction sur la longueur de trace.

- Cas de base : nous considérons les traces de longueur 1.
  Soit $(\mathsf{SId}_1, f_1, H_1) \in \mathsf{Exec}^{hs}(\Pi)$ une trace d'exécution valide. Nous avons $\mathsf{SId}_1 = \emptyset$, la fonction f_1 n'est définie nulle part et $H_1 = \mathbf{kn}^{hs}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}$. Considérons la trace $(\mathsf{SId}_1, \overline{f_1}^{hs\to h}, \overline{H_1}^{hs\to h})$. La fonction $\overline{f_1}^{hs\to h}$ n'est définie nulle part. En outre, nous avons $\overline{H_1}^{hs\to h} = \overline{\mathbf{kn}^{hs}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}}^{hs\to h} = \mathbf{kn}^{h}(\mathsf{int}) \cup \mathsf{Nonce}(\mathsf{int}) \cup \mathsf{SKShort}(\mathsf{int}) \cup \{\mathsf{init}\}$. Nous concluons que $\overline{tr}^{hs\to h} \in \mathsf{Exec}^{h}(\overline{\Pi}^{hs\to h})$.
- L'étape d'induction : nous considérons les traces de longueur $n+1$.
  Soit $tr = (\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n+1} \in \mathsf{Exec}^{hs}(\Pi)$ une trace d'exécution valide de longueur $n+1$. Nous avons par hypothèse d'induction que

  $$(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n} \in \mathsf{Exec}^{hs}(\Pi)$$

  implique

  $$\overline{(\mathsf{SId}_1, f_1, H_1)(\xrightarrow{\mathbf{trans}_{i-1}} (\mathsf{SId}_i, f_i, H_i))_{2\le i\le n}}^{hs\to h} \in \mathsf{Exec}^{h}(\overline{\Pi}^{hs\to h}).$$

  Trois cas sont possibles pour la n-ième transition de la trace tr :

  1. $\mathbf{trans}_n = \mathbf{corrupt}(a_1, \ldots, a_l)$.
     Étant donné que cette type de transition est seulement autorisé pour la première transition, nous avons $tr = (\mathsf{SId}_1, f_1, H_1) \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}_2, f_2, H_2)$, où $\mathsf{SId}_2 = \mathsf{SId}_1$, $f_2 = f_1$ et $H_2 = H_1 \cup \bigcup_{1\le i\le l} \mathbf{kn}^{hs}(a_i)$.
     Considérons la trace $\overline{tr}^{hs\to h} = \overline{(\mathsf{SId}_1, f_1, H_1)}^{hs\to h} \xrightarrow{\mathbf{corrupt}(a_1,\ldots,a_l)} (\mathsf{SId}_2, \overline{f_2}^{hs\to h}, \overline{H_2}^{hs\to h})$. Nous avons $\mathsf{SId}_1 = \mathsf{SId}_2$, $\overline{f_2}^{hs\to h} = \overline{f_1}^{hs\to h}$ et $\overline{H_1}^{hs\to h} \cup \bigcup_{1\le i\le l} \mathbf{kn}^{h}(a_i) = \overline{H_1 \cup \bigcup_{1\le i\le l} \mathbf{kn}^{hs}(a_i)}^{hs\to h} = \overline{H_2}^{hs\to h}$. Nous concluons que $\overline{tr}^{hs\to h} \in \mathsf{Exec}^{h}(\overline{\Pi}^{hs\to h})$.
  2. $\mathbf{trans}_n = \mathbf{new}(i, a_1, \ldots, a_k)$.
     Nous avons $tr = \ldots(\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}_n \cup \{\mathsf{sid}\}, f_{n+1}, H_{n+1})$ où $\mathsf{sid} = (|\mathsf{SId}_n|+1, i, (a_1, \ldots, a_k))$. Nous avons $f_{n+1}(\mathsf{sid}) = (\sigma, 1)$ et $f_{n+1}(s) = f_n(s)$ pour tout identifiant de session $s \in \mathsf{SId}_n$ et $H_{n+1} = H_n$.
     Considérons la trace $\overline{tr}^{hs\to h} = \overline{\ldots(\mathsf{SId}_n, f_n, H_n)}^{hs\to h} \xrightarrow{\mathbf{new}(i,a_1,\ldots,a_k)} (\mathsf{SId}_n \cup \{\mathsf{sid}\}, \overline{f_{n+1}}^{hs\to h}, \overline{H_{n+1}}^{hs\to h})$. Pour tout identifiant de session $s \in \mathsf{SId}_n$, nous avons $\overline{f_{n+1}}^{hs\to h}(s) = \overline{f_n}^{hs\to h}(s)$. Nous considérons l'état local $(\sigma', 1)$ qui est affecté à l'identifiant de session sid. Le domaine de la substitution σ' ne contient que des variables des types agent, nonce et clef de chiffrement à court terme. Étant donné que l'identifiant de cette session est identique à l'identifiant qui apparaît dans la trace tr, les domaines des substitutions σ' et σ sont identiques et aussi les termes qui sont affectes aux variables de ce domaine. Nous avons donc $\sigma' = \sigma$. Vu les types des variables nous avons aussi $\sigma = \overline{\sigma}^{hs\to h}$ et donc $\sigma' = \overline{\sigma}^{hs\to h}$. Par conséquent, $(\sigma', p) = (\overline{\sigma}^{hs\to h}, p) = \overline{f_{n+1}}^{hs\to h}(\mathsf{sid})$. Finalement, nous avons $\overline{H_{n+1}}^{hs\to h} = \overline{H_n}^{hs\to h}$ et donc $\overline{tr}^{hs\to h} \in \mathsf{Exec}^{h}(\overline{\Pi}^{hs\to h})$.

3. $\mathbf{trans}_n = \mathbf{send}(\mathsf{sid}, m)$.

   Nous avons $tr = \ldots(\mathsf{SId}_n, f_n, H_n) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}_{n+1}, f_{n+1}, H_{n+1})$. Nous avons $\mathsf{SId}_{n+1} = \mathsf{SId}_n$ et $f_{n+1}(s) = f_n(s)$ pour tout identifiant de session $s \in \mathsf{SId}_n$, tel que $s \neq \mathsf{sid}$.
   Considérons la trace $\overline{tr}^{hs\to h} = \overline{\ldots(\mathsf{SId}_n, f_n, H_n)}^{hs\to h} \xrightarrow{\mathbf{send}(\mathsf{sid},m')} (\mathsf{SId}_{n+1}, \overline{f_{n+1}}^{hs\to h}, \overline{H_{n+1}}^{hs\to h})$ où le terme m' est donné dans la définition de la fonction $\overline{\cdot}^{hs\to h}$. Nous avons $\overline{f_{n+1}}^{hs\to h}(s) = \overline{f_n}^{hs\to h}(s)$ pour tout identifiant de session $s \neq \mathsf{sid}$ de SId_n.
   Nous supposons que $\mathsf{sid} = (s, j, (a_1, \ldots, a_k))$ et que $\Pi(j) = (l_i, r_i)_{1\leq i\leq l}$ est le rôle joué dans cette session. L'état de cette session avant la transition est donné par $\overline{f_n}^{hs\to h}(\mathsf{sid}) = (\overline{\sigma}^{hs\to h}, p)$. Nous devons distinguer deux cas :
   - Nous avons $p \leq l$ et il existe un unificateur θ tel que $m = l_p\sigma\theta$. Nous avons $f_{n+1}(\mathsf{sid}) = (\sigma \cup \theta, p+1)$ et $H_{n+1} = H_n \cup \{r_p\sigma\theta\}$.
     Dans ce cas, nous avons $m' = \overline{m}^{hs\to h}$, parce que le terme l_p est défini et m et $l_p\sigma$ sont unifiables. Le lemme 16 assure que $\overline{m}^{hs\to h}$ est déductible à partir de $\overline{H_n}^{hs\to h}$, car m était déductible à partir de H_n. Le lemme 15 dit que les termes $\overline{m}^{hs\to h}$ et $\overline{l_p\sigma}^{hs\to h}$ sont unifiables avec $\overline{\theta}^{hs\to h}$ comme unificateur. Le nouveau état local affecté à sid est donc $(\overline{\sigma}^{hs\to h} \cup \overline{\theta}^{hs\to h}, p+1) = \overline{f_{n+1}}^{hs\to h}(\mathsf{sid})$ et, par conséquent, $\overline{H_n}^{hs\to h} \cup \{\overline{r_p\sigma\theta}^{hs\to h}\} = \overline{H_{n+1}}^{hs\to h}$.
     Nous concluons que $\overline{tr}^{hs\to h} \in \mathsf{Exec}^h(\overline{\Pi}^{hs\to h})$.
   - Soit nous avons $f_{n+1} = f_n$ et $H_{n+1} = H_n$. Deux cas sont possibles :
     - Soit, $p > l$. Le terme l_p n'est pas défini et nous avons $m' = a$ pour une identité d'agent $a \in \mathsf{ID}$. Ce terme est déductible à partir de $\overline{H_n}^{hs\to h}$ selon le système de déduction. Nous avons $\overline{f_{n+1}}^{hs\to h}(\mathsf{sid}) = \overline{f_n}^{hs\to h}(\mathsf{sid})$ et $\overline{H_{n+1}}^{hs\to h} = \overline{H_n}^{hs\to h}$. Nous concluons que $\overline{tr}^{hs\to h} \in \mathsf{Exec}^h(\overline{\Pi}^{hs\to h})$.
     - Soit, $p \leq l$ et les termes m et $l_p\sigma$ ne sont pas unifiables. Nous devons encore distinguer deux cas :
       - Si les termes $\overline{m}^{hs\to h}$ et $\overline{l_p\sigma}^{hs\to h}$ ne sont pas unifiables, nous avons $m' = \overline{m}^{hs\to h}$. Le lemme 16 assure que le terme m' est déductible à partir de l'ensemble $\overline{H_n}^{hs\to h}$. Nous avons $\overline{f_{n+1}}^{hs\to h}(\mathsf{sid}) = \overline{f_n}^{hs\to h}(\mathsf{sid})$ et $\overline{H_{n+1}}^{hs\to h} = \overline{H_n}^{hs\to h}$ et donc $\overline{tr}^{hs\to h} \in \mathsf{Exec}^h(\overline{\Pi}^{hs\to h})$.
       - Si les termes $\overline{m}^{hs\to h}$ et $\overline{l_p\sigma}^{hs\to h}$ sont unifiables, nous avons $m' = a$ pour une identité d'agent $a \in \mathsf{ID}$, ce qui est déductible à partir de $\overline{H_n}^{hs\to h}$. Dans ce cas, le terme m' ne s'unifie pas avec le terme $\overline{l_p\sigma}^{hs\to h}$, parce que ce dernier contient un sous-terme de la forme $\{t\}_{\mathsf{dk}(a)}$. Nous démontrons cela par contradiction : supposons que le terme $\overline{l_p\sigma}^{hs\to h}$ ne contient pas de sous-terme de la forme $\{t\}_{\mathsf{dk}(a)}$. Le terme $l_p\sigma$ est donc un terme fixe. Nous avons $l_p\sigma = \overline{l_p\sigma}^{hs\to h}$. Il existe un unificateur θ, tel que $\overline{m}^{hs\to h} = \overline{l_p\sigma}^{hs\to h}\theta$. Le terme $\overline{m}^{hs\to h}$ ne contient donc pas un sous-terme de la forme $\{t\}_{\mathsf{dk}(a)}$ non plus. Par conséquent, le terme m est fixe aussi et nous avons $m = \overline{m}^{hs\to h}$. Alors, $\overline{m}^{hs\to h} = \overline{l_p\sigma}^{hs\to h}\theta$ implique $m = l_p\sigma\theta$, ce qui est en

contradiction avec le fait que m et $l_p\sigma$ ne sont pas de termes unifiables. Nous concluons que le terme $\overline{l_p\sigma}^{hs\to h}$ contient un sous-terme de la forme $\{t\}_{\mathsf{dk}(a)}$ et n'est donc pas unifiable avec le terme $m' = a$. Nous avons donc $\overline{f_{n+1}}^{hs\to h}(\mathsf{sid}) = \overline{f_n}^{hs\to h}(\mathsf{sid})$ et $\overline{H_{n+1}}^{hs\to h} = \overline{H_n}^{hs\to h}$. Il découle que $\overline{tr}^{hs\to h} \in \mathsf{Exec}^h(\overline{\Pi}^{hs\to h})$. □

Théorème 3. *Soit* $\Pi : [k] \to \mathsf{Roles}^{hs}$ *un protocole à* k *participants et soit* $\phi \in \mathcal{L}_2^{hs}$ *une formule. Nous avons* $\overline{\Pi}^{hs\to h} \models \overline{\phi}^{hs\to h} \implies \Pi \models \phi$.

Preuve. Nous prouvons que $\mathsf{Exec}^h(\overline{\Pi}^{hs\to h}) \subseteq \mathsf{Tr}^h(\overline{\phi}^{hs\to h})$ implique $\mathsf{Exec}^{hs}(\Pi) \subseteq \mathsf{Tr}^{hs}(\phi)$. Soit $tr \in \mathsf{Exec}^{hs}(\Pi)$ une trace d'exécution. Par le lemme 17, nous avons $\overline{tr}^{hs\to h} \in \mathsf{Exec}^h(\overline{\Pi}^{hs\to h})$, donc $\overline{tr}^{hs\to h} \in \mathsf{Tr}^h(\overline{\phi}^{hs\to h})$. Cela signifie que $[\![\overline{\phi}^{hs\to h}, \overline{tr}^{hs\to h}]\!] = 1$. En appliquant le lemme 13, nous obtenons $[\![\phi, tr]\!] = 1$ et donc $tr \in \mathsf{Tr}^{hs}(\phi)$. □

Contre-exemple pour la propriété 4

Nous considérons le protocole $\Pi : [2] \to \mathsf{Roles}^{hs}$ suivant :

$$\begin{aligned}\Pi(1) &= (\mathsf{init}, [X^1_{A_1}]_{\mathsf{sk}(A_1)})\\ \Pi(2) &= (\{X^1_{A_1}\}_{\mathsf{dk}(A_1)}, \mathsf{stop}).\end{aligned}$$

Le premier rôle envoie le nonce $X^1_{A_1}$ signé avec sa clef de signature. Le deuxième rôle attend ce nonce, mais chiffré avec la clef privée de chiffrement du premier rôle. Considérons une propriété qui dit que, tant que l'agent qui joue le premier rôle n'est pas corrompu et le deuxième rôle attend un message de cet agent, le nonce chiffré attendu par le deuxième rôle n'est pas égal au nonce utilisé dans le message émis par le premier rôle :

$$\phi = \forall \mathcal{LS}_{1,2}.\varsigma\ \forall \mathcal{LS}_{2,2}.\varsigma'\ NC(\varsigma(A_1)) \wedge (\sigma'(A_1) = \sigma(A_1)) \to (\varsigma(X^1_{A_1}) \neq \varsigma'(X^1_{A_1})).$$

Intuitivement, le protocole Π satisfait la formule ϕ, parce que, étant donné que l'agent A_1 qui joue le premier rôle n'est pas corrompu, l'adversaire ne peut pas chiffrer le nonce $X^1_{A_1}$ avec la clef $\mathsf{dk}(A_1)$, bien qu'il puisse déduire ce nonce. Le message reçu par l'agent qui joue le deuxième rôle sera donc toujours différent de $\{X^1_{A_1}\}_{\mathsf{dk}(A_1)}$. Nous avons $\Pi \models \phi$. Considérons maintenant le protocole $\overline{\Pi}^{hs\to h}$, donné par :

$$\begin{aligned}\overline{\Pi(1)}^{hs\to h} &= (\mathsf{init}, \{X^1_{A_1}\}_{\mathsf{dk}(A_1)})\\ \overline{\Pi(2)}^{hs\to h} &= (\{X^1_{A_1}\}_{\mathsf{dk}(A_1)}, \mathsf{stop}).\end{aligned}$$

Il est clair que même une exécution normale de ce protocole ne satisfait pas la formule ϕ. Nous avons donc $\overline{\Pi}^{hs\to h} \not\models \overline{\phi}^{hs\to h}$. Nous concluons que la propriété 4 n'est pas satisfaite pour la fonction $\overline{\cdot}^{hs\to h}$.

3.5.3 La fonction $h \to \_$ réexaminée

Dans ce paragraphe, nous n'examinons plus la fonction $\overline{\cdot}^{h\to}$, parce que nous l'avons déjà fait dans le paragraphe 3.2.4. Étant donné que l'expressivité de la logique $\mathcal{L}_2^h$ est moins forte que celle de la logique $\mathcal{L}_1^{h-}$, le théorème 1 est plus fort que celui auquel nous pourrions nous attendre dans

le cas de la logique $\mathcal{L}_2^h$. Le fait que nous n'avons pas autorisé l'utilisation de l'identité d'agent h dans la logique $\mathcal{L}_1^{h-}$ n'y change rien, parce que nous devrions changer le modèle d'une manière similaire pour obtenir un résultat pour une modification de la logique $\mathcal{L}_2^h$. Nous nous contentons donc avec le résultat déjà obtenu.

3.6 Conclusion

Nous avons prouvé toutes les correspondances indiquées dans la figure 3.1. En ce qui concerne la preuve de la propriété de correction 1 pour la fonction $\overset{h\rightarrow}{\cdot}\overset{-}{}$, nous étions obligés de modifier légèrement le modèle. Cette modification ne peux guerre porter atteinte à notre résultat, car il s'agit simplement d'utiliser l'identité d'agent h en tant que agent. En contrepartie, nous étions en mesure de prendre en compte une plus grande classe de propriétés, à savoir le propriétés qui peuvent s'écrire comme formules de la logique $\mathcal{L}_1^{h-}$, que pour les fonctions $\overset{lhs\rightarrow hs}{\cdot}$ et $\overset{hs\rightarrow h}{\cdot}$. Pour prouver la propriété de correction 3 pour ces deux dernières fonctions, nous étions obligés de restreindre l'ensemble des formules aux formules dans $\mathcal{L}_2^\star$. Quant aux propriétés 2 et 4, nous avons donné des contre-exemples. Comme nous l'avons déjà mentionné dans l'introduction de ce chapitre, cela est dû au fait que nous n'avons pas l'implication $m_1 \neq m_2 \implies f(m_1) \neq f(m_2)$. Cette implication est peut-être valide dans le modèle modifié que nous avons utilisé pour prouve la propriété de correction 1 pour la fonction $\overset{h\rightarrow}{\cdot}\overset{-}{}$. Cela reste à vérifier.

En somme, nous pouvons conclure que nous pouvons transférer des propriétés prouver dans un modèle restreint vers un modèle plus riches en fonctionnalités, si nous pouvons exprimer cette propriété dans la logique $\mathcal{L}_2^\star$, respectivement $\mathcal{L}_1^{h-}$ pour la fonction $\overset{h\rightarrow}{\cdot}\overset{-}{}$. Une application pratique des résultats obtenus se trouve dans le chapitre 4.

Chapitre 4

AVISPA

4.1 Introduction

4.1.1 Les modèles symboliques et les modèles calculatoires

Depuis les années quatre-vingts, deux approches ont été développées afin d'analyser des protocoles cryptographiques. L'une d'entre elles repose sur un modèle calculatoire qui permet de raisonner sur des questions de complexité et de probabilité. Cette approche dont le cadre a été développé à commencer avec [BM82, Yao82, GM84] prend en compte toutes les attaques en temps polynomial qui puissent exister contre un protocole. L'autre approche est basée sur un modèle symbolique. Les primitives cryptographiques y sont traitées comme des boîtes noires. Avec les travaux de Dolev et Yao [DY81], la communauté scientifique a reconnu que cette deuxième approche permet d'écrire des preuves simplifiées et souvent automatisées (voir par exemple le paragraphe 1.5.2). L'outil AVISPA a été conçu pour la vérification des protocoles cryptographiques dans un modèle symbolique, mais jusqu'alors aucune des garanties donnée par l'outil ne peut être automatiquement transférée vers un modèle calculatoire.

Des résultats récents [AR00, MW04, CW05, BP05, Her05, CKKW06] élucident les rapports entre ces deux approches. Les auteurs de [CW05] et [CKKW06] ont par exemple prouvé que des traces calculatoires d'un protocole peuvent être projetées sur des traces symboliques. On peut ensuite se servir de cette projection pour transférer des propriétés des protocoles, prouvées dans le modèle symbolique, vers un modèle calculatoire. Pour établir ce résultat, les auteurs ont utilisé un modèle symbolique qui permet d'exprimer le caractère probabiliste des primitives cryptographiques, telles que le chiffrement et la signature.

4.1.2 Motivation

Étant donné qu'à ce jour aucun outil automatique de vérification n'utilise un modèle basé sur des symboles fonctionnels dans leurs versions probabilistes, les résultats de [CW05, CKKW06] ne sont pas directement utilisables en pratique. Pour les rendre accessibles aux vérificateurs, deux possibilités sont envisageables : soit, on développe de nouveaux outils de vérification qui intègrent un modèle symbolique permettant d'exprimer le caractère probabiliste des primitives cryptographiques ou, au moins, on modifie les outils existants. Cette méthode est pénible et susceptible d'être erronée, car les développements et modifications doivent être validés à leurs tours. Soit, on utilise les résultats de correspondance obtenus au cours du chapitre précédent. En effet, nous avons vu que les propriétés d'un protocole qui appartiennent à une certaine classe peuvent être transférées vers d'autres modèles, sans qu'une nouvelle vérification soit nécessaire.

Fonctionnalités autorisées par	chiffrement symétrique	chiffrement asymétrique	signature	fonctions de hachage
l'outil AVISPA	+	+	−	+
l'article [CW05]	−	+	+	−
l'article [CKKW06]	−	+	−	+
le modèle avec sign. et hachage	+	+	+	+

FIG. 4.1 – Comparaison des fonctionnalités autorisées par la plate-forme AVISPA, ainsi que par les articles [CW05, CKKW06] respectifs et le modèle avec signature et hachage. Peut être exprimé syntaxiquement (+), ne peut pas être exprimé syntaxiquement (−). Notons que les articles mettent à disposition la version probabiliste des fonctionnalités.

Nos résultats de correspondance nous permettent de conclure automatiquement que la propriété est satisfaite malgré le changement du modèle. Si nous vérifions donc un protocole dans un modèle symbolique où les primitives sont déterministes, alors nous pouvons transférer le résultat vers un modèle symbolique où elles sont probabilistes et, ensuite, utiliser les résultats de [CW05, CKKW06] pour obtenir des garanties dans un modèle calculatoire.

4.1.3 Résultats

Basés sur l'article [CKKW06], nous avons développé un module pour la plate-forme AVISPA qui permet de donner des garanties dans un modèle calculatoire. Les propriétés pour lesquelles l'outil peut donner des garanties calculatoires sont l'*authentification faible* et l'*authentification forte*. Nous avons ensuite essayé de vérifier tous les protocoles de la bibliothèque d'AVISPA. Parmi les 84 protocoles pris en compte, nous avons pu donné des garanties calculatoires pour 9 protocoles. Nous avons identifié les causes pour ce taux faible qui consistent principalement en l'utilisation du chiffrement symétrique et le chiffrement asymétrique à clef privée, modes de chiffrement qui ne sont pas pris en compte par l'article [CKKW06]. Néanmoins, la plate-forme AVISPA est maintenant la première qui peut donner des garanties dans un modèle calculatoire. Nous avons aussi réalisé une documentation du nouveau module dans le cadre du projet AVISPA. Une première version des résultats présentés dans ce chapitre a déjà été publié dans [CHW06a] et [CHW06b].

4.1.4 La structure de ce chapitre

La section 4.2.1 explique le fonctionnement du nouveau module, ainsi que les restrictions qui doivent être respectées, si on veut transférer les propriétés vérifiées vers le modèle calculatoire. Elle explique aussi la signification des propriétés par rapport au modèle calculatoire. Ensuite, la section 4.2.2 explique la syntaxe que nous avons utilisée pour afficher des formules de la logique $\mathcal{L}_2^{hs}$ sur l'écran. La section 4.2.3 explique les propriétés prises en compte à travers un exemple concret d'un protocole. Nous donnons une description d'utilisation du nouveau module dans la section 4.2.4. La section 4.2.5 décrit les résultats de la vérifications des protocoles de la bibliothèque d'AVISPA. Nous finissons ce chapitre avec une discussion dans la section 4.2.6.

4.2 Un nouveau module pour AVISPA

Nous motivons d'abord pourquoi nous avons préféré l'article [CKKW06] à l'article [CW05] pour le développement du module. La figure 4.1 résume ces raisons. Une première constatation est que notre modèle symbolique qui comprend les différents modes de chiffrement, la signature et le hachage permet d'exprimer syntaxiquement toutes les fonctionnalités qu'on peut trouver dans la plate-forme AVISPA qui, elle, ne permet pas la signature. Si nous comparons les moyens syntaxique d'AVISPA avec ceux de l'article [CW05], il nous ne reste comme fonctionnalités communes que le chiffrement asymétrique, tandis que, en comparaison avec l'article [CKKW06], nous avons le chiffrement asymétrique, ainsi que les fonctions de hachage. Pour cette raison, nous avons espéré d'obtenir un meilleur taux de protocoles vérifiables en préférant les résultats de l'article [CKKW06]. Notons que cela n'exclut pas le développement d'un module similaire en se basant sur l'autre article.

4.2.1 Le nouveau module

La convention de l'interprétation des clefs

Étant donné que la plate-forme AVISPA ne fait pas de distinction syntaxique entre des clefs publiques et privées, mais seulement entre des clefs et leurs inverses, nous avons décidé de suivre la convention qu'une *clef* (ou *clef non inverse*) est une clef publique et qu'une *clef inverse* est une clef privée. C'est aussi l'interprétation adoptée par la plupart de vérificateurs.

Le fonctionnement du module

Le modèle symbolique utilisé dans [CKKW06] se sert d'étiquettes pour exprimer le caractère probabiliste des primitives cryptographiques. Il est donc proche du premier modèle défini dans le chapitre 2. Nous notons l'algèbre qui est utilisé dans le modèle symbolique de cette article par $T^{[\mathrm{CKKW06}]}$. Nous la considérons comme un sous-ensemble de l'algèbre T^{lhs}, donc $T^{[\mathrm{CKKW06}]} \subset T^{lhs}$. Nous notons $T^{[\mathrm{CKKW06}]}_{lhs \to hs}$ l'algèbre $T^{[\mathrm{CKKW06}]}$ à laquelle nous avons ôté les étiquettes. Nous considérons cela comme une application de la fonction $\overline{\cdot}^{lhs \to hs}$, définie dans le chapitre 3, à l'ensemble de termes $T^{[\mathrm{CKKW06}]}$.

D'autre part, les moyens syntaxiques de la plate-forme AVISPA ne permettant pas d'exprimer leur caractère probabiliste, le modèle utilisé dans cet outil est proche du deuxième modèle défini dans le chapitre 2, c'est-à-dire du modèle qui est basé sur l'algèbre T^{hs}. Nous notons l'algèbre utilisée dans cet outil par T^{AVISPA}. Étant donné qu'AVISPA n'autorise par exemple pas la signature, nous considérons cette algèbre comme un sous-ensemble de l'algèbre T^{hs}, donc $T^{\mathrm{AVISPA}} \subset T^{hs}$. L'algèbre T^{AVISPA} autorise cependant plus que l'algèbre $T^{[\mathrm{CKKW06}]}_{lhs \to hs}$. AVISPA permet par exemple le chiffrement symétrique. Nous avons donc $T^{[\mathrm{CKKW06}]}_{lhs \to hs} \subset T^{\mathrm{AVISPA}}$.

La première vérification que notre module doit donc effectuer est de tester, si un protocole est spécifier en utilisant exclusivement les fonctionnalités de l'algèbre $T^{[\mathrm{CKKW06}]}_{lhs \to hs}$. Nous devons notamment tester que

- le protocole n'utilise pas le chiffrement symétrique. Néanmoins, des clefs de chiffrement symétrique peuvent être déclarées et envoyées en position de plaintexte.
- le protocole n'utilise pas de chiffrement asymétrique à clef inverse.

Une deuxième restriction que notre module doit prendre en compte provient du modèle calculatoire utilisé dans l'article [CKKW06]. Ce modèle ne permet pas la modélisation de tout comportement qu'un agent peut avoir. Notamment, les agents doivent observer quelques règles

concernant le maniement des clefs privées. Ces règles sont aussi à respecter dans la modélisation symbolique d'un protocole. Notre module vérifie donc l'observation de certaines règles concernant les clefs inverses, à savoir que

- les clefs inverses ne sont pas publiées ni
- envoyées chiffrées pendant l'exécution du protocole.

La dernière restriction que notre module doit vérifier concerne la logique $\mathcal{L}_2^{hs}$ dans laquelle les tests d'égalité sont seulement autorisés entre des termes fixes. Or, l'authentification faible dans AVISPA demande de tels tests entre deux termes. Le module vérifie donc, si cette propriété peut être exprimée dans cette logique :

- L'*authentification faible* ne peut être vérifiée que pour des messages du type agent et nonce.

Notons que cette restriction est un peu trop forte, car la logique $\mathcal{L}_2^{hs}$ permet des tests d'égalité entre termes fixes. Or, les termes du type agent et nonce ne sont qu'un sous-ensemble des termes fixes. Cette restriction pourrait donc être relaxée.

Après avoir effectué ces tests syntaxiques, un des moteurs de vérification de la plate-forme AVISPA est lancé. Si la vérification se termine positivement, alors nous pouvons appliquer le théorème 2. Le protocole satisfait donc les propriétés en question dans le modèle basé sur l'algèbre T^{lhs}. Utilisant le résultat de correspondance de l'article [CKKW06], la plate-forme AVISPA peut automatiquement traduire les propriétés dans le modèle calculatoire où elles sont aussi satisfaites.

Le propriétés garanties par le module

Le transfert des propriétés d'authentification d'un modèle symbolique vers un modèle calculatoire et plutôt facile [BJ03, MW04, CKKW06] en comparaisons avec la propriété du *secret* [CKKW06]. La raisons se trouve dans le fait que, dans un modèle symbolique, le déchiffrement d'un message chiffré nécessite la connaissance de la clef de déchiffrement. Dans ce cas, un intrus peut apprendre tout sur le contenu du message, sinon il n'apprend rien. La notion d'information partielle n'existe donc pas, contrairement aux modèles calculatoires où un intrus peut éventuellement apprendre une partie du message chiffré sans pourtant connaître la clef. Pour obtenir quand même un résultat de correspondance, la définition de la propriété du secret dans [CKKW06] est plus restrictive que celle utilisée par la plate-forme AVISPA. Celle-ci ne peut donc pas être transférée automatiquement. Par conséquent, notre module se restreint aux propriétés de l'*authentification faible* et *forte*. Les propriétés sont garanties sous la condition que

- le schéma de chiffrement est sûr dans le sens de IND-CCA et [NY90,BDPR98] et que
- les fonctions de hachage sont modélisées par un oracle randomisé [FS87,BR93,CGH04].

4.2.2 La syntaxe de la logique $\mathcal{L}_2^{hs}$ dans AVISPA

La logique utilisée dans AVISPA pour afficher des formules sur l'écran est une variante de la logique $\mathcal{L}_2^{hs}$, définie dans le paragraphe 3.4.2. Nous donnons une brève description de sa syntaxe. Les termes sont donnés par

$$T ::= s(X) \mid a \mid \mathit{intruder} \mid k \mid \mathit{inv}(k) \mid T.T \mid \{T\}_k,$$

où X est une variable d'un type quelconque spécifiée par le protocole ou la variable *Session* dont nous expliquons la signification ultérieurement. La constante a représente une identité d'agent et la constante *intruder* est utilisée pour représenter l'intrus. Les termes k et $\mathit{inv}(k)$ représentent les clefs de chiffrement asymétrique. La paire est représentée par $T.T$. Nous utilisons la variable s comme caractère générique pour indiquer dans nos formules l'emplacement des substitutions qui

apparaissent dans les états locaux. En simplifiant, on pourrait dire que $s(X)$ est l'instanciation de la variable X dans l'état local s. Les deux quantificateur sont écrits comme suit :

- `FOR ALL SESSIONS` s `OF ROLE` r `AT STEP` p
- `THERE EXISTS A SESSION` s `OF ROLE` r `AT STEP` p

L'interprétation des ces quantificateurs suit la définition que nous avons donnée dans le chapitre 3. Les formules de notre logique sont donné par

$$F ::= F \wedge F \mid F \vee F \mid (u_1 = u_2) \mid \sim (t_1 = t_2)$$
$$\mid \texttt{FOR ALL SESSIONS } s \texttt{ OF ROLE } r \texttt{ AT STEP } p\ F$$
$$\mid \texttt{THERE EXISTS A SESSION } s \texttt{ OF ROLE } r \texttt{ AT STEP } p\ F$$

où u_1 et u_2 sont des termes du type agent ou nonce et t_1, ainsi que t_2 sont des termes quelconques. Le caractère $\sim$ est utilisé pour représenter la négation. Comme dans la logique $\mathcal{L}_2^{hs}$, nous n'autorisons pas la négation des formules arbitraires. Notons que nous n'avons plus besoin du prédicat NC, parce que nous exprimons le fait que, par exemple, l'agent a est non corrompu dans à l'état s par la formule $\sim (s(a) = \mathit{intruder})$, ce qui signifie intuitivement que, dans l'état s, l'agent a n'est pas joué par l'intrus.

4.2.3 Les propriétés expliquées à l'exemple

Pour mieux expliquer la signification des formules affichées par l'outil AVISPA, nous considérons le protocole suivant comme exemple :

$$A \rightarrow B \quad : \quad N_A$$

Ce protocole contient deux rôles A et B. Le rôle A envoie le nonce N_A au protocole B. L'exécution du rôle A est terminée à l'étape 1. Le rôle B s'arrête également à l'étape 1 après avoir reçu le nonce. Nous donnons la spécification HLPSL de ce protocole :

```
role aa (A,B : agent, Snd, Rcv: channel(dy))
played_by A
def=
  local State  : nat,
        Na     : text

  const sec_na : protocol_id

  init  State := 0

  transition
   1. State = 0 /\ Rcv(start) =|>
      State' := 1 /\ Na' := new() /\ Snd(Na) /\ witness(A, B, na, Na)
      /\ secret(Na, sec_na, {A, B})
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role bb (B,A : agent, Snd, Rcv: channel(dy))
played_by B
```

```
def=
  local State  : nat,
        Na     : text

  init  State := 0

  transition
   1. State = 0 /\ Rcv(Na') =|>
      State' := 1 /\ request(B, A, na, Na)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role session(A,B: agent)
def=

  local SA, SB, RA, RB: channel(dy)

  composition
           aa(A, B, SA, RA)
       /\  bb(B, A, SB, RB)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

role environment()
def=
  const a, b   : agent,
        na     : protocol_id

  intruder_knowledge = {a, b}
  composition session(a, b)
end role

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

goal
  secrecy_of sec_na
  authentication_on na
end goal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

environment()
```

Dans les paragraphes suivants, nous donnons les formules des propriétés que notre module afficherait avec ce protocole comme entrée. Ces propriétés seraient satisfaites dans un modèle calculatoire. Notons que notre protocole d'exemple ne satisfait pas ces propriétés et la plate-

forme AVISPA ne affiche donc pas de formules. Celles-ci doivent plutôt être considérées à titre d'exemple. Dans les formules, les rôles A et B sont respectivement appelés `aa` et `bb`.

L'authentification faible

L'authentification faible, exprimée dans la syntaxe de la logique $\mathcal{L}_2^{hs}$ dans AVISPA, est donnée pour le protocole d'exemple par la formule :

```
FOR ALL SESSIONS s OF ROLE bb AT STEP 1
    THERE EXISTS A SESSION s' OF ROLE aa AT STEP 1
    (s'(Na) = s(Na)) /\ (s'(A) = s(A)) /\ (s'(B) = s(B))
    /\ ~(s(A) = intruder)
```

Nous considérons toutes les session terminées du rôle B, c'est-à-dire les sessions où le rôle a reçu le nonce N_A. Pour toutes ces sessions s, il doit y avoir une session s' du rôle A, tel que les rôles A et B sont d'accord sur la valeur du nonce N_A et leurs identités respectives et le rôle A n'est pas joué par l'intrus. Notons que le protocole de l'exemple ne satisfait pas cette propriété, parce que l'intrus peut intercepter le nonce N_A et envoyer un autre au rôle B.

Nous garantissons l'authentification faible seulement pour des termes du type agent ou nonce, car la logique $\mathcal{L}_2^{hs}$ n'autorise pas de tests d'égalité entre des termes arbitraires. Soit par exemple t un terme arbitraire, nous aurions alors un test $(s'(t) = s(t))$, ce qui ne fait pas partie de la logique. Notons que, d'après les résultats du chapitre 3, nous pourrions relaxer cette restriction et autoriser des tests d'égalité entre des termes fixes.

La protection contre le rejeu

La propriété de la protection contre le rejeu garantit que l'intrus ne peut pas rejouer un message déjà envoyé. Elle est décrite pour le protocole d'exemple dans la syntaxe de la logique $\mathcal{L}_2^{hs}$ dans AVISPA par la formule :

```
FOR ALL SESSIONS s OF ROLE bb AT STEP 1
    FOR ALL SESSIONS s' OF ROLE bb AT STEP 1
    ~(s(Na) = s'(Na)) \/ ~(s(B) = s'(B)) \/ ~(s(A) = s'(A))
    \/ (s(Session) = s'(Session)) \/ (s(A) = intruder)
```

Dans cette formule, les termes $s(Session)$ et $s'(Session)$ réfèrent aux identifiants de session des sessions s et s'. La propriété dit que, pour deux sessions s et s' du rôle B, le fait que les variables N_A, A et B sont instanciées par les mêmes valeurs implique que les sessions s et s' sont identiques ou que le rôle A est joué par un agent corrompu, à savoir l'intrus. Si le rôle A est joué par l'intrus, alors le nonce N_A peut être rejoué autant de fois que l'intrus le veut.

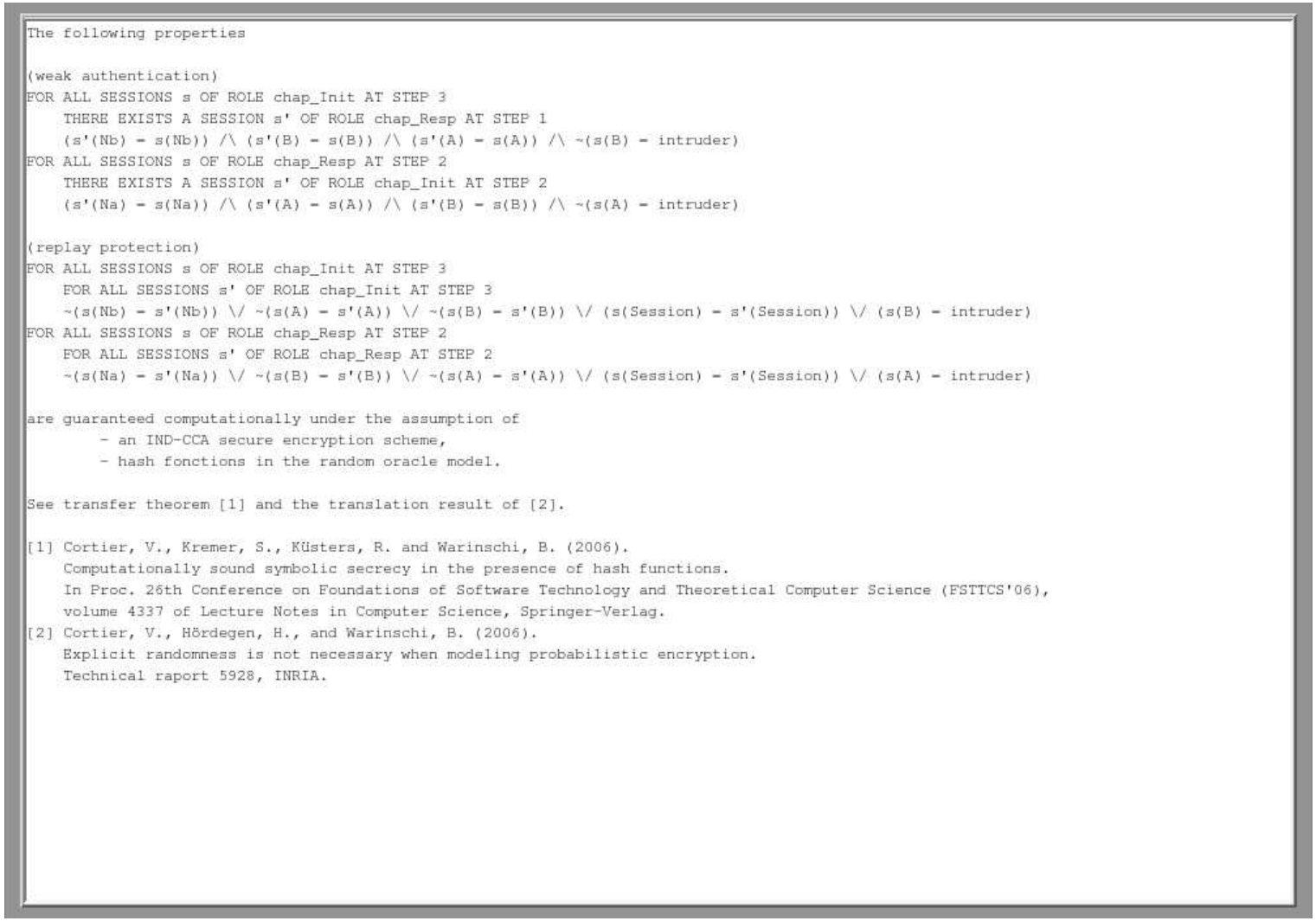
L'authentification forte

La propriété de l'authentification forte dans AVISPA est satisfaite, si les propriétés de l'authentification faible et de la protection contre le rejeu sont satisfaites.

4.2.4 Bref aperçu sur la plate-forme AVISPA

Le projet AVISPA [ABB+05] a développé une plate-forme de vérification automatique de protocoles cryptographiques, appelée également AVISPA. Elle est accessible sur le réseau à l'adresse `www.avispa-project.org`. Elle met à disposition le langage HLPSL qui est utilisé

FIG. 4.2 – La fenêtre d'AVISPA en mode *Expert*.



```
The following properties

(weak authentication)
FOR ALL SESSIONS s OF ROLE chap_Init AT STEP 3
    THERE EXISTS A SESSION s' OF ROLE chap_Resp AT STEP 1
    (s'(Nb) = s(Nb)) /\ (s'(B) = s(B)) /\ (s'(A) = s(A)) /\ ~(s(B) = intruder)
FOR ALL SESSIONS s OF ROLE chap_Resp AT STEP 2
    THERE EXISTS A SESSION s' OF ROLE chap_Init AT STEP 2
    (s'(Na) = s(Na)) /\ (s'(A) = s(A)) /\ (s'(B) = s(B)) /\ ~(s(A) = intruder)

(replay protection)
FOR ALL SESSIONS s OF ROLE chap_Init AT STEP 3
    FOR ALL SESSIONS s' OF ROLE chap_Init AT STEP 3
    ~(s(Nb) = s'(Nb)) \/ ~(s(A) = s'(A)) \/ ~(s(B) = s'(B)) \/ (s(Session) = s'(Session)) \/ (s(B) = intruder)
FOR ALL SESSIONS s OF ROLE chap_Resp AT STEP 2
    FOR ALL SESSIONS s' OF ROLE chap_Resp AT STEP 2
    ~(s(Na) = s'(Na)) \/ ~(s(B) = s'(B)) \/ ~(s(A) = s'(A)) \/ (s(Session) = s'(Session)) \/ (s(A) = intruder)

are guaranteed computationally under the assumption of
        - an IND-CCA secure encryption scheme,
        - hash fonctions in the random oracle model.

See transfer theorem [1] and the translation result of [2].

[1] Cortier, V., Kremer, S., Küsters, R. and Warinschi, B. (2006).
    Computationally sound symbolic secrecy in the presence of hash functions.
    In Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06),
    volume 4337 of Lecture Notes in Computer Science, Springer-Verlag.
[2] Cortier, V., Hördegen, H., and Warinschi, B. (2006).
    Explicit randomness is not necessary when modeling probabilistic encryption.
    Technical raport 5928, INRIA.
```

FIG. 4.3 – La fenêtre pour l'affichage des formules de $\mathcal{L}_2^{hs}$.

Nombre de protocoles dans la bibliothèque	84
exclusivement chiffrement asymétrique à clef non inverse et fonctions de hachage	12
dont symboliquement (et donc calculatoirement) prouvés sûrs	9

FIG. 4.4 – Résumé des résultats de l'analyse des protocoles de la bibliothèque d'AVISPA

pour spécifier à la fois les protocoles et les propriétés à vérifier. Dans ce langage, deux symboles sont définies pour le chiffrement symétrique et asymétrique. Il permet aussi la déclaration des fonctions de hachage. Il existe également un opérateur de concaténation. Par contre, il n'y pas d'opérateur pour la signature. Le langage HLPSL permet de décrire l'échange des messages entre les participants pendant l'exécution d'un protocole. La plate-forme permet de vérifier trois propriétés : le secret, l'authentification faible et l'authentification forte. Cette dernière propriété est une combinaisons entre l'authentification faible et la protection contre le rejeu. Après avoir spécifié le protocole dans HLPSL, celui-ci est traduit dans le langage IF (*intermediate format*) par le compilateur HLPSL2IF. Quant au langage IF, il s'agit d'un langage intermédiaire qui à été développé en vue de faciliter la déduction automatique. Le code IF peut ensuite être utilisé dans des moteurs d'analyse différents qui effectuent la vérification. Jusqu'alors, quatre outils de vérification ont été intégrés dans la plate-forme : OFMC, ATSE, SATMC et TA4SP. Une description du langage HLPSL se trouve dans [YCCC+04]. Les techniques de vérification utilisées dans les moteurs sont décrites dans [BMV05, CV02, AC04, BHKO04].

Pour utiliser le nouveau module, nous devons d'abord choisir le mode *Expert*. Ensuite, nous pouvons charger, éditer et sauvegarder la spécification d'un protocole. Nous pouvons maintenant choisir un parmi les quatre moteurs d'analyse et lancer la vérification. Pendant la traduction du code HLPSL dans le langage IF, le compilateur HLPSL2IF teste automatiquement, si le protocole peut être traduit dans notre formalisme, de façon à ce que toutes les conditions d'application du nouveau module soient satisfaites. Le compilateur teste aussi, si les propriétés à vérifier peuvent être exprimées dans la logique $\mathcal{L}_2^{hs}$, notamment si l'authentification faible ne se fait que sur des termes de type agent ou nonce. Si toutes les conditions sont satisfaites et le protocole a été prouvé sûr, alors le bouton *CryptoSec* en bas de la zone de texte est actif. La figure 4.2 montre cette situation. En cliquant sur le bouton, nous pouvons afficher les formules qui sont satisfaites par le protocole dans le modèle calculatoire. La figure 4.3 montre à titre d'exemple, comment les formules sont typiquement représentées.

4.2.5 La bibliothèque d'AVISPA analysée

Nous avons analysé les 84 protocoles de la bibliothèque d'AVISPA à l'aide de l'outil ATSE. Le résumé de cette analyse se trouve dans le tableau de la figure 4.4. Étant donné que nous ne pouvons transférer des propriétés que pour les protocoles qui utilisent uniquement le chiffrement asymétrique à clef non inverse et des fonctions de hachage, mais ne pas le chiffrement symétrique ou le chiffrement asymétrique à clef inverse, nous nous sommes concentrés sur les 12 protocoles de cette classe. Dans le modèle symbolique utilisé par AVISPA, 9 parmi ces 12 protocoles peuvent être prouvés sûrs. Ces 9 protocoles satisfont donc les propriétés d'authentification dans le modèle calculatoire sous la condition que l'authentification faible ne se fasse que pour des agents ou des nonces.

Le tableau suivant donne les résultats détaillés. La première colonne comporte le nom de chaque protocole de la bibliothèque. La deuxième colonne indique, si le protocole a pu être vérifié

au niveau symbolique par le moteur d'analyse ATSE (+) ou non (−). Un point d'interrogation (?) indique que la vérification avec ATSE n'a pas donné de résultat dans un temps raisonnable. Cela est seulement le cas pour deux protocoles qui, de plus, utilisent le chiffrement symétrique. Pour ces raisons, nous n'aurions pas pu espérer obtenir des garanties pour le modèle calculatoire, même si on les avait pu vérifier dans le modèle symbolique. La troisième colonne nous dit, si le protocole utilise le chiffrement symétrique (+) ou non (−). La quatrième colonne indique, si le protocole utilisé des clefs inverses de façon à ce que nous ne pouvons pas le traiter avec le nouveau module. Notons qu'un plus (+) peut indiquer plusieurs situations possibles : une clef inverse ne doit pas être publiée, ni envoyée, ni utilisée pour le chiffrement asymétrique. Si aucune de ces situations est donnée, nous l'indiquons par un moins (−). La cinquième colonne indique, si le protocole utilise des fonctions de hachage (+) ou non (−). Finalement, la dernière colonne nous dit, si le résultat de la vérification peut être transféré vers un modèle calculatoire. Dans ce cas, nous l'indiquons avec un plus (+), sinon avec un moins (−).

Nom du protocle	symb. vérifié	chiffrement symétrique	clef inverse	fonction de hachage	CryptoSec
2pRSA	+	−	+	+	−
8021x_Radius	+	+	−	+	−
AAAMobileIP	+	+	−	−	−
apop	+	−	−	+	+
ASW-abort	−	−	+	+	−
ASW	+	−	+	+	−
CHAPv2	+	−	−	+	+
CRAM-MD5	+	−	−	+	+
CTP-non_predictive-fix	+	+	−	+	−
DHCP-delayed-auth	+	−	−	+	+
DNSSEC	−	+	+	+	−
EAP_AKA	+	−	−	+	+
EAP_Archie	+	+	−	+	−
EAP_IKEv2	+	−	+	+	−
EAP_SIM	+	−	−	+	+
EAP_TLS	+	−	+	+	−
EAP_TTLS_CHAP	+	−	+	+	−
EKE2	+	+	−	+	−
EKE	−	+	−	−	−
FairZG	−	+	+	+	−
FairZG-LucaC	+	+	+	+	−
geopriv	+	+	−	+	−
h.530-fix	?	+	−	+	−
h.530	−	+	−	+	−
h.530-mim	−	+	−	+	−
hip	+	−	+	+	−
IKEv2-CHILD	+	+	−	+	−
IKEv2-DS	−	−	+	+	−
IKEv2-DSx	+	−	+	+	−
IKEv2-EAP-Archie	+	+	+	+	−
IKEv2-MAC	+	+	−	+	−

Nom du protocle	symb. vérifié	chiffrement symétrique	clef inverse	fonction de hachage	CryptoSec
IKEv2-MACx	+	+	−	+	−
ISO1	−	−	+	−	−
ISO2	+	−	+	−	−
ISO3	−	−	+	−	−
ISO4	+	−	+	−	−
ISO-9798-fixed	+	−	+	−	−
ISO-9798	−	−	+	−	−
Kerb-basic	+	+	−	−	−
Kerb-Cross-Realm	+	+	−	−	−
Kerb-Forwardable	+	+	−	−	−
Kerb-PKINIT	+	+	+	+	−
Kerb-preauth	+	+	−	−	−
Kerb-Ticket-Cache	+	+	−	−	−
LAP-LECP	−	+	+	+	−
lipkey-spkm-known-initiator	+	−	+	+	−
lipkey-spkm-unknown-initiator	+	−	+	+	−
LPD-IMSR	+	+	+	−	−
LPD-MSR	−	+	−	−	−
LPD-MSR-sec-initial	+	+	−	−	−
NSPK-fix	+	−	−	−	+
NSPK	−	−	−	−	−
NSPK-KS-evaluation	−	−	+	−	−
NSPK-KS-fix	+	−	+	−	−
NSPK-KS	−	−	+	−	−
NSPKxor	−	−	−	−	−
NSSK	+	+	−	+	−
password	?	+	+	+	−
PBK-fix	−	−	+	−	−
PBK-fix-weak-auth	+	−	+	−	−
PBK	−	−	+	−	−
PEAP	+	+	+	−	−
pervasive	+	+	+	−	−
QoS-NSLP	+	+	+	−	−
RADIUS-RFC2865	+	+	−	+	−
self-signatures	−	−	+	+	−
SET-purchase.executable	−	+	+	+	−
SET-purchase	−	+	+	+	−
SET-purchase-honest-payment-gateway.executable	+	+	+	+	−
SET-purchase-honest-payment-gateway	+	+	+	+	−
SET-simplif-purchase.executable	−	+	+	+	−
SHARE	−	−	−	−	−

Nom du protocle	symb. vérifié	chiffrement symétrique	clef inverse	fonction de hachage	CryptoSec
Simple	+	+	−	+	−
sip	+	−	−	+	+
skey	+	−	−	+	+
SPEKE	+	+	−	−	−
SRP	+	+	−	+	−
ssh-transport	+	+	+	+	−
tesla	+	+	+	+	−
TLS	+	+	−	+	−
TSIG	+	+	−	+	−
tsp	+	−	+	+	−
two_pseudonyms	−	+	−	+	−
UMTS_AKA	+	+	−	+	−

4.2.6 Conclusion

Nous avons montré que le théorème 2, en combinaison avec l'article [CKKW06] nous permet de transférer les propriétés d'un protocole, vérifiées par la plate-forme AVISPA, vers un modèle calculatoire. Les 9 protocoles que nous avons pu vérifier en sont la preuve. Bien que la méthode que nous avons proposée soit viable, le taux faible des protocoles que nous pouvons traiter est décevant. En effet, il ne s'agit que d'environ 11% de protocoles de la bibliothèque d'AVISPA. Dans la suite nous en discutons les raisons et des solutions alternatives.

Une première constatation porte sur le fait que l'article [CKKW06] ne prend pas en compte le chiffrement symétrique. Cependant, comme la liste des protocoles examinés nous le montre, un nombre total de 47 protocoles utilise ce mode de chiffrement. Nous sommes donc obligés de déjà rayer ces protocoles, ce que nous laisse 37 parmi les 84 protocoles. Un deuxième coup contre cette méthode provient du fait que l'article en question n'autorise pas l'utilisation des clefs privées sauf pour le déchiffrement des messages, mais, étant donné que la plate-forme AVISPA ne met pas à disposition la signature, beaucoup de protocoles, ayant besoin de cette fonction, ont été spécifiés en la remplaçant par un chiffrement à clef inverse. Tous ces protocoles ne peuvent pas être traités par notre module. Un nombre total de 43 protocoles utilise des clefs inverses d'une manière non autorisés par l'article [CKKW06]. Parmi ces protocoles, 25 n'utilisent pas le chiffrement symétrique. Nous devons donc déduire ces 25 des 37 protocoles restants, ce qui nous laisse 12 protocoles. Parmi ces 12 protocoles, 9 peuvent être prouvés sûrs par la plate-forme AVISPA à l'aide du moteur d'analyse ATSE. Pour ces 9 protocoles, nous pouvons donc transférer les propriétés d'authentification prouvées vers le modèle calculatoire, à condition que l'authentification faible ne se fasse que pour des agents ou nonces. Le mélange des restrictions que nous avons dues faire à cause de l'article sur lequel notre méthode est basé, ainsi qu'à cause des fonctionnalités limitées proposées par AVISPA est donc responsable pour le taux faible de protocoles vérifiés calculatoirement.

Alternativement, nous aurions pu basé notre travail sur [CW05], article qui autorise l'utilisation du chiffrement asymétrique et de la signature, mais pas de fonctions de hachage et de chiffrement symétrique. La figure 4.1 résume la situation. Quant aux clefs privées, les mêmes restrictions doivent être faites comme avant. Un grand avantage de cet article est le fait qu'il permet aussi le transfert de la propriété du secret pour les nonces vers un modèle calculatoire.

Cependant, parmi les 9 protocoles prouvés sûrs au niveau symbolique, seulement 1 protocole n'utilise pas de fonctions de hachage. C'est le protocole NSPK-fix. Basé sur cet article, nous ne pouvons donc transférer les propriétés que de ce seul protocole.

Il existe une autre possibilité d'obtenir un meilleur taux de protocoles vérifiés dont nous pouvons transférer les propriétés vers un modèle calculatoire : bien que la plate-forme AVISPA n'admette pas de signature, nous pouvons traiter les protocoles qui l'utilisent à l'aide du théorème 3. Ce théorème nous dit que, si nous remplaçons la signature par un chiffrement à clef privée, alors les propriétés satisfaites par le protocole modifié sont aussi satisfaites par le protocole original. Ensuite, nous pouvons continuer à transférer ces propriétés vers un modèle calculatoire à l'aide du théorème 2, ainsi que des résultats de [CW05] en respectant les restrictions respectives. La difficulté de cette façon de procéder repose dans le fait que la plate-forme AVISPA ne distingue pas entre la clef publique et la clef privée au niveau syntaxique. La seule distinction qu'on puisse faire est celle d'entre une clef et son inverse. On ne peut pas décider au niveau syntaxique, si un protocole est spécifié en utilisant la clef comme clef publique et la clef inverse comme clef privée ou à l'inverse. Par conséquent, nous ne pouvons pas savoir dans quels cas il s'agit d'un chiffrement asymétrique à clef inverse et dans quels cas d'un codage de la signature, ce qui rend impossible l'application du théorème 3 d'une manière automatique. Cependant, si un vérificateur s'impose d'utiliser les clefs seulement comme clefs publiques et les clefs inverses seulement comme clefs privées, alors il peut utiliser notre résultat pour obtenir des garanties dans un modèle calculatoire pour plus de protocoles. Cependant, cela doit être partiellement fait à la main pour les raisons citées.

Un comptage montre que, parmi les 84 protocoles de la bibliothèque d'AVISPA, 16 n'utilisent ni le chiffrement symétrique ni des fonctions de hachage. On peut donc espérer transférer les propriétés de ces 10 protocoles à l'aide des théorèmes 3 et 2, ainsi que de l'article [CW05] sous la condition que l'utilisation des clefs inverses est vraiment un codage de la signature. Cependant, seulement 6 parmi ces 16 protocoles peuvent être vérifiés symboliquement, dont un (NSPK-fix) qui n'utilise pas de clef inverse. Ce dernier a donc déjà été traité à l'aide de [CKKW06], ce qui nous laisse encore potentiellement 5 protocoles de plus (ISO2, ISO4 et ISO-9798-fixed, NSPK-KS-fix et PBK-weak-auth) dont nous pouvons espérer des garanties dans le modèle calculatoire.

Deuxième partie

Étude de protocoles récursifs

Chapitre 5

Présentation générale des protocoles récursifs

5.1 Introduction

Les protocoles qui ont été considérés jusqu'alors dans le domaine de la vérification formelle ont deux points communs : premièrement, il s'agit des protocoles dont le nombre des messages échangés pendant une session est limité et, deuxièmement, la forme de chaque message est fixe et finie. La limitation du nombre des messages échangés pendant une session et la prévisibilité de la forme des messages sont deux caractéristiques qui facilitent l'analyse et la vérification d'un protocole. Avec les travaux de Dolev et Yao [DY81], un cadre formel bien fondé de vérification a été développé. Les résultats obtenus au cours des années ont aussi permis la conception d'outils automatiques de vérification.

Cependant, les intérêts récents se portent vers des protocoles plus complexes [Mea01]. Ces protocoles permettent la transmission de messages à taille non bornée, assurent la communication entre des participants dont le nombre est a priori inconnu ou assurent une communication pendant laquelle le nombre de messages échangés est illimité. Toutes ces variations sur la structure et le nombre de messages échangés dans une session impliquent un mode de fonctionnement qui peut être décrit d'une manière récursive. Pour cette raison, nous appelons ce type de protocole les *protocoles récursifs*. Les domaines d'application des protocoles récursifs sont par exemple la transmission des données de taille inconnue dont on veut assurer l'authenticité, la distribution des clefs de groupe dont le nombre de participants n'est pas connu, etc.

L'enjeu étant réel, les propositions des protocoles récursifs ne manquent pas. Cependant, vu leur nature, les vérificateurs doivent faire face à de nouveaux défis d'analyse, d'autant plus, que les inventeurs des protocoles récursifs revendiquent souvent pour leurs protocoles des propriétés qui n'ont pas été considérées jusqu'à maintenant. Dans la suite de ce chapitre introductif, nous éclaircissons les différentes formes de récursivité qu'on peut trouver dans un protocole. Nous illustrons chacune de ces formes avec des exemples. Ces nombreux exemples justifient aussi l'intérêt que le monde scientifique porte vers les protocoles récursifs. Pour conclure ce chapitre, nous donnons un bref aperçu sur les travaux et résultats existants.

Nous utilisons le chapitre suivant pour proposer un modèle formel qui nous permet de décrire des protocoles récursifs. Le dernier chapitre de cette partie contient une étude de cas où nous montrons que nous pouvons appliquer notre modèle formel avec profit pour vérifier certaines propriétés.

5.2 Présentation générale des protocoles récursifs

5.2.1 Types de récursivité

Plusieurs types de récursivité existent dans les protocoles. Sans vouloir être exhaustif, nous dressons une liste :

- Les structures de données transmises sur le réseau sont de taille non bornée. Nous pouvons par exemple mentionner les listes, les arbres, les piles, etc.
- Le nombre de participants d'un protocole est non borné. Dans ce cas, un participant reçoit un message, fait un calcul et transmet le message à un autre participant. L'exécution du protocole continue jusqu'à ce que le critère d'arrêt soit satisfait.
- Le nombre de participants d'un protocole est borné, mais le nombre d'échanges de messages entre les participants est non borné.

Nous donnons des exemples pour chacun des cas. Ces exemples sont simplistes pour mieux illustrer la dimension de récursivité dont il est question, mais nous justifions leurs prises en compte en citant de vrais protocoles.

Structures de données de taille non bornée

Exemple 17. Dans cette classe de protocoles récursifs, on peut considérer un serveur S qui reçoit une requête d'un client C et qui renvoie la réponse. La requête elle-même contient une structure de données non bornée. Considérons un serveur qui reçoit une liste d'entiers et qui renvoie la somme de ces entiers :

$$\begin{aligned} C \to S &: [n_1, \ldots, n_k] \\ S \to C &: \sum_{i=1}^{k} n_i \end{aligned}$$

où $k \in \mathbb{N}$ et $n_i \in \mathbb{N}$ pour $1 \leq i \leq k$. Pour calculer la somme des entiers transmis dans une liste dont la longueur est a priori inconnue, le serveur S doit effectuer un calcul récursif.

Les protocoles conformes au standard ISAKMP [MSST98] utilisent cette forme de récursivité. Ce standard demande qu'un protocole d'échange de clefs soit divisé en deux étapes. La première étape sert aux participants de se mettre d'accord sur le protocole d'échange de clefs utilisé. Pour ce faire, l'émetteur propose des *associations de sécurité* (SA). Une SA consiste en un algorithme de chiffrement, une fonctions de hachage, une méthodes d'authentification, etc. Le receveur choisit une des propositions qui correspond à ses exigences de sécurité ou les refuse toutes, lorsqu'aucune ne le satisfait. Après s'être mis d'accord sur le protocole, l'échange des clefs est effectué. Cela correspond à la deuxième étape. La récursivité apparaît dans la première étape où le receveur est obligé de parcourir la liste des SA proposées par l'émetteur. Un exemple de protocole qui est conforme à ce standard est le Internet Key Exchange (IKE) protocole [HC98].

Nombre de participants non borné

Exemple 18. Considérons un client C qui cherche un produit pour un prix moins cher que $p \in \mathbb{N}$. Pour ce faire, il connaît un grand nombre de sites $S_1, S_2, \ldots$ où ce produit est vendu. Il contacte tous les sites l'un après l'autre jusqu'à ce qu'il ait trouvé le produit pour un prix moins cher que p :

$$\begin{array}{lll} \text{Étape} \quad \text{i} & C \to S_i & : \langle \textit{requête} \rangle \\ & S_i \to C & : \langle \textit{prix } p_i \rangle \end{array}$$

À l'étape i, le client envoie sa requête au site S_i et, après avoir reçu le prix p_i, il le compare avec son objectif p. Si $p_i \leq p$, alors il s'en contente et l'exécution du protocole se termine. Sinon, le client passe à l'étape $i+1$. Étant donné qu'on ne connaît pas le site S_n, tel que $p_n \leq p$, le nombre des sites impliqués dans l'exécution du protocole ne peut pas être borné.

Une famille de protocoles utilisant cette forme de récursivité est présentée dans [STW98] et développée dans [AST00]. Prenons par exemple le protocole GDH.2. Il a été conçu pour permettre à des participants $M_1, M_2, \ldots$ d'établir une clef de groupe. L'objectif du protocole est évidemment que cette clef ne soit connue que par les membres du groupe. Pour ce faire, une variante d'échange des clefs selon le schéma Diffie-Hellman est utilisé [DH76, STW96]. Ce schéma travaille sur un groupe limité par un nombre premier q. Chaque participant M_i possède un exposant secret N_i. Le participant M_1 commence l'exécution du protocole en envoyant le message $\langle\{g\}, g^{N_1}\rangle$ au participant M_2. Dans cette paire, la première composante $\{g\}$ est un ensemble contenant un générateur $g < q$ du groupe utilisé. La deuxième composante g^{N_1} est appelée la *valeur cardinale*. Le participant M_2 calcule g à la puissance N_2 et ajoute la valeur cardinale dans la liste. Finalement, il calcule une nouvelle valeur cardinale $g^{N_1 N_2}$ et envoie le message $\langle\{g^{N_2}, g^{N_1}\}, g^{N_1 N_2}\rangle$ au participant suivant. D'une manière plus générale, le participant M_i envoie le message $\langle\{g^{\frac{N_1 \ldots N_i}{N_k}} \mid k \in [i]\}, g^{N_1 \ldots N_i}\rangle$ au participant M_{i+1}. Lorsque le dernier participant M_n reçoit le message, il distribue toutes les valeurs de l'ensemble $\{g^{\frac{N_1 \ldots N_n}{N_k}} \mid k \in [n]\}$, de façon à ce que chaque participant M_i obtienne la valeur $g^{\frac{N_1 \ldots N_n}{N_i}}$. En calculant la N_i-ème puissance de cette valeur, le participant M_i obtient la clef commune du groupe. Le nombre n des participants n'est pas borné. Notons que la gestion de la structure de donnée pour stocker les valeurs $g^{\frac{N_1 \ldots N_i}{N_k}}$, $k \in [i]$, se fait probablement aussi d'une manière récursive. Cela est un exemple pour la première forme de récursivité que nous venons de discuter ci-dessus.

Nombre de participants borné, mais nombre des messages non borné

Exemple 19. Considérons le téléchargement d'un fichier. Un client C se connecte à un serveur S et demande la transmission d'un fichier. La taille du fichier est arbitraire. Le serveur coupe le fichier en morceaux d'une taille fixe et les transmet un par un. De l'autre côté de la liaison, le client reconstitue le fichier à partir des morceaux reçus. À la fin de la transmission, le serveur notifie le client en envoyant une constante pour signaler que tous les morceaux ont été émis :

Étape	0	$C \to S$	:	$\langle$*nom du fichier*$\rangle$
Étape	i	$S \to C$	:	$\langle$*i-ième morceaux du fichier*$\rangle$
Étape	n	$S \to C$	:	tous_les_morceaux_transmis

où $1 \leq i \leq n-1$. Comme la taille du fichier n'est a priori pas connue, mais la taille des morceaux est fixe, le nombre des étapes i à répéter ne peut pas être borné. Nous avons donc besoin d'un traitement récursif pour le bon fonctionnement de ce protocole. Dans cet exemple, la récursivité apparaît sur les deux côtés de la communication : le serveur transmet des messages, tant que la fin du fichier n'est pas atteinte et le client accepte des messages tant qu'il n'a pas reçu la notification. Pour compléter ce scénario, on peut penser à la transmission du fichier à travers d'un canal bruité. Dans ce cas, on étend le protocole de façon à ce que le client puisse tester l'intégrité des morceaux transmis. À chaque fois, qu'il trouve une altération, il a le droit de redemander au serveur le morceau concerné.

Des protocoles qui utilisent cette forme de récursivité peuvent être trouvés dans [PCTS00] et dans le paragraphe 4.3 de [WC04]. Ils ont été développés pour transmettre un flot de données

en assurant l'authenticité de celles-ci. Pour ce faire, ces protocoles utilisent une chaîne de clefs $k_1, k_2, \ldots$ telle que $k_i = f(k_{i+1})$. La fonction f est ici une fonction pseudo-aléatoire, telle que, étant donnée une valeur y, on ne peut trouver une valeur x qui satisfait l'égalité $f(x) = y$ qu'avec une probabilité négligeable. Le protocole du deuxième schéma de [PCTS00] fonctionne comme suit : l'émetteur signe d'abord la clef k_0 et l'envoie au receveur. Puis, il envoie les messages $m_i = (n_i, k_{i-1}, \mathit{hash}(\langle n_i, k_i \rangle))$, $i \geq 1$, où n_i est la donnée à transmettre. Le receveur, après avoir accepté la clef k_0 comme authentique, reçoit ces messages. Lorsqu'il reçoit le message m_j $j \geq 2$, il vérifie l'authenticité du message $m_{j-1} = (n_{j-1}, k_{j-2}, \mathit{hash}(\langle n_{j-1}, k_{j-1} \rangle))$ qu'il a reçu auparavant à l'aide de la clef k_{j-1}. Il s'assure d'abord de l'authenticité de la clef k_{j-1} en vérifiant $k_{j-2} = f(k_{j-1})$, puis il calcule la valeur $\mathit{hash}(\langle n_{j-1}, k_{j-1} \rangle)$ qu'il compare avec la troisième valeur contenue dans le message m_{j-1}. Si les deux valeurs sont égales, alors il accepte la donnée n_{j-1} comme authentique.

5.2.2 Aperçu sur les travaux et résultats existants

À notre connaissance, le premier protocole récursif qui a été vérifié est décrit dans [BO97]. Dans ce protocole, le nombre de participants est non borné. Le protocole permet d'établir des clefs de façon à ce que deux participants puissent s'authentifier mutuellement. Le protocole à été vérifié par [Pau97a] et [BS97] en utilisant des assistants de preuve. Les auteurs de [RS98] ont trouvé une attaque contre une implémentation concrète de ce protocole et ils proposent une correction dans ce même article. Néanmoins, ces approches restent très ad hoc, car elles ne visent que sur la vérification de ce seul protocole. Un intérêt particulier a aussi été porté sur les protocoles proposé par le projet CLIQUES [STW98]. Ces protocoles ont été analysés, d'une part à la main [PQ01], d'autre part à l'aide d'un assistant de preuve [Mea00].

Une approche plus universelle d'analyser des protocoles récursifs est développée dans [KW03] et [KW04]. Dans ces articles, les auteurs modélisent le comportement récursif des agents à l'aide des automates d'arbre [CDG+97], appelés *tree transducers*. Le résultat de leur travail consiste en un algorithme de décision pour la question du secret d'une information pendant l'exécution d'un protocole. L'algorithme fonctionne dans le cas où le nombre des messages émis et reçus est borné et où le protocole n'utilise que des clefs atomiques. L'analyse se fait en présence d'un intrus du type Dolev-Yao. Dans les mêmes articles, les auteurs montrent que la mise à disposition de certains pouvoirs complémentaires aux participants entraîne l'indécidabilité de la question du secret. Ces pouvoirs sont notamment la possibilité de tester l'égalité entre deux messages arbitraires, l'utilisation des clefs composées ou l'ajout des théories équationnelles pour permettre l'utilisation du *ou exclusif* ou de l'exponentiation de Diffie-Hellman [DH76]. Une approche similaire est présentée dans [Küs05]. Dans cet article, un autre type d'automate est utilisé pour modéliser les participants.

Dans [Tru05], l'auteur propose une autre approche, appelée *selecting theories*, qui modélise le comportement récursif des participants avec des clauses de Horn. Ce nouveau formalisme permet aux participants de recevoir des messages d'une taille non bornée, d'émettre de multiples messages pendant une étape, de comparer des messages et de les stocker. Les protocoles sont vérifiés en présence d'un intrus du type Dolev-Yao et l'auteur propose un algorithme de décision pour un nombre borné de sessions. Certaines restrictions pour l'utilisation du formalisme mentionné existent. Il n'est par exemple pas possible pour un participant de transformer un message reçu en n'importe quel autre message à envoyer. Une autre restriction est l'absence des nonces. Le travail de [Kür06] remédie à ce deuxième problème. Finalement, les auteurs de [KT07] ont, dans leur travail le plus récent, ajouté à ce formalisme la possibilité d'analyser des protocoles récursifs qui ont besoin de l'opérateur *ou exclusif*.

Chapitre 6

Modélisation des protocoles récursifs

L'objectif de ce chapitre est de définir un modèle symbolique qui nous permettra d'entreprendre des études formelles de protocoles récursifs. Comme dans la première partie de cette thèse, le modèle est un système de transitions et l'intrus est du type Dolev-Yao [DY81]. Étant donné que nous avons en tête l'étude d'un protocole concret, notre modèle comporte quelques particularités concernant l'algèbre utilisée. Néanmoins, comme nous argumentons dans la conclusion de ce chapitre, notre modèle est général et peut servir à l'étude de nombreux protocoles récursifs.

6.1 L'algèbre et le système de déduction

Pour mieux comprendre nos choix de conception concernant l'algèbre, nous commençons par une brève explication du fonctionnement du protocole que nous étudierons en détails dans le chapitre 7. Notons que notre proposition de modéliser la récursivité n'est pas due à une particularité de ce protocole, mais qu'il s'agit d'une approche très générale.

Le fonctionnement du protocole

Le protocole que nous nous proposons d'étudier est un protocole de commerce électronique. Il sert à récupérer des offres qui existent d'un bien sur le réseaux. L'utilisateur peut ensuite choisir parmi ces offres celle qui lui convient le mieux. Nous utilisons la terminologie suivante : un *site* est une adresse sur le réseau où des informations sont mises à disposition. Ces informations consistent en des *offres*. Un site propose une offre lors de la réception d'une *requête*. L'exécution du protocole se déroule comme suit : l'utilisateur engendre un message qui contient une requête et la liste des sites dont il veut obtenir les offres. Ce message est envoyé successivement à tous les sites qui figurent dans la liste. Lorsqu'un site reçoit le message, il ajoute son offre en fonction de la requête. Le message final est renvoyé à l'utilisateur. Celui-ci peut maintenant choisir l'offre. Le choix de notre algèbre est donc guidé par les structures de données dont nous avons besoin pour modéliser ce protocole. Néanmoins, dans un autre contexte, l'utilisation d'une autre algèbre est tout à fait envisageable.

La modélisation de la récursivité

Nous avons deux comportements récursifs dans le protocole que nous venons de décrire : un site doit parcourir une liste pour ajouter son offre et l'utilisateur doit parcourir une liste pour vérifier s'il a obtenu les offres des sites dont il les a demandées. Nous devons donc associer a chaque message une information supplémentaire qui décrit comment les listes sont à traiter. Cette

information que nous appelons *instance de programme* dans la suite nous renvoie à un *programme* qui exécute la tâche demandée. Nous soulignons encore une fois que notre approche de modéliser la récursivité n'est pas particulière à l'étude de cas que nous avons en tête, mais peut être considérée comme proposition générale.

6.1.1 L'algèbre

Nous dénotons $S = \{s_1, s_2, \ldots\}$ l'ensemble des identités qui représentent les sites et $R = \{r_1, r_2, \ldots\}$ les requêtes qu'on peut leur poser. Nous associons à chaque site $s \in S$ et requête $r \in R$ une offre $\mathit{off}(s, r)$. Les sites $s \in S$ disposent aussi des clefs de chiffrement et déchiffrement asymétrique et des clefs de signature et de vérification que nous notons respectivement $\mathsf{ek}(s)$, $\mathsf{dk}(s)$, $\mathsf{sk}(s)$ et $\mathsf{vk}(s)$. Nous définissons une fonction $\breve{\cdot}$ qui est utilisée pour marquer un terme. Elle est par exemple utile, si nous voulons marquer l'élément actuel lors d'un parcours d'une liste.

Lors d'une exécution répétée d'un programme, nous pouvons à chaque fois avoir besoin de nonces frais. Pour ce faire, nous faisons dépendre les nonces de l'instance du programme qui est exécutée. Ces instances sont notées par des constantes de l'ensemble infini P_{Inst}. Nous notons les nonces $\mathit{nonce}(p, i)$ où $p \in P_{\mathit{Inst}}$ fait référence à l'instance du programme qui a créé ce nonce et $i \in \mathbb{N}$ est un entier. La possibilité de faire dépendre les nonces des entiers différents nous permet d'utiliser par instance de programme autant de nonces que nous voulons. Pour des raisons expliquées ultérieurement, nous autorisons aussi l'utilisation de la variable x dans la position d'une instance de programme. Notons que cette seule variable x nous suffit dans notre modèle.

Comme fonctions cryptographiques, nous définissons le chiffrement asymétrique, la signature et le hachage. Nous notons $\{m\}_{\mathsf{ek}(s)}$ pour le chiffrement du terme m par la clef de chiffrement $\mathsf{ek}(s)$ et $[m]_{\mathsf{sk}(s)}$ la signature de m avec la clef $\mathsf{sk}(s)$. Le haché de m est noté $\mathit{hash}(m)$.

Finalement, nous définissons une fonction de paire $\langle \_, \_ \rangle$ et un constructeur des listes $m :: l$ où m est un terme et l est une liste. La liste vide est dénotée par $[\,]$. Les algèbres de termes T_x et de listes L_x sont données par

$$\begin{array}{lcl} T_x & ::= & s \mid r \mid \breve{T_x} \mid \mathit{nonce}(p, i) \mid \mathit{off}(s, r) \mid \mathsf{ek}(s) \mid \mathsf{dk}(s) \mid \mathsf{sk}(s) \mid \mathsf{vk}(s) \\ & & \mid \langle T_x, T_x \rangle \mid \{T_x\}_{\mathsf{ek}(s)} \mid [T_x]_{\mathsf{sk}(s)} \mid \mathit{hash}(T_x) \mid L_x, \\ L_x & ::= & [\,] \mid T_x :: L_x \end{array}$$

où $s \in S$ est une identité de site, $r \in R$ une requête, $p \in P_{\mathit{Inst}} \cup \{x\}$ une instance de programme ou la variable x et $i \in \mathbb{N}$ un entier. Notons que l'ensemble de listes L_x est un sous-ensemble des termes T_x. Nous autorisons donc des listes qui contiennent à leur tour des listes. Au lieu de dénoter une liste par $m_1 :: \ldots :: m_n :: [\,]$, nous écrivons aussi $[m_1, \ldots, m_n]$. Nous notons T l'ensemble des termes de T_x qui ne contiennent pas la variable x. Dans la suite, nous aurons aussi besoin des algèbres $T^{a,n} = T \cup \{\mathit{accept}, \mathit{not\_accept}\}$ et $T_x^{a,n} = T_x \cup \{\mathit{accept}, \mathit{not\_accept}\}$ où *accept* et *not_accept* sont des constantes.

6.1.2 Le système de déduction

Les règles de déduction se trouvent dans la figure 6.1. Elles décrivent les connaissances qu'un adversaire peut déduire à partir d'un ensemble de termes connus $H \subseteq T$. Il nous suffit ici de considérer l'ensemble T, car l'adversaire déduit seulement à partir des termes émis sur le réseau.

Les règles de *Connaissances initiales* permettent de déduire tout terme $m \in H$ à partir de l'ensemble H, ainsi que les identités de sites $s \in S$ et leurs clefs publiques $\mathsf{ek}(s)$ et $\mathsf{vk}(s)$. La constante $[\,]$ qui représente la liste vide est aussi déductible avec ces règles. Les clefs secrètes $\mathsf{dk}(s)$

$\dfrac{}{H \vdash m}\ m \in H$	$\dfrac{}{H \vdash s, \check{s}, r, \mathsf{ek}(s), \mathsf{vk}(s), [\,]}\ s \in S,\ r \in R$	Connaissance initiale
$\dfrac{H \vdash m_1 \quad H \vdash m_2}{H \vdash \langle m_1, m_2 \rangle}$	$\dfrac{H \vdash \langle m_1, m_2 \rangle}{H \vdash m_1, m_2}$	Composition et décomposition des paires
$\dfrac{H \vdash m \quad H \vdash l}{H \vdash m::l}\ l \in L$	$\dfrac{H \vdash m::l}{H \vdash m, l}$	Composition et décomposition des listes
$\dfrac{H \vdash m \quad H \vdash \mathsf{ek}(s)}{H \vdash \{m\}_{\mathsf{ek}(s)}}$	$\dfrac{H \vdash \{m\}_{\mathsf{ek}(s)} \quad H \vdash \mathsf{dk}(s)}{H \vdash m}$	Chiffrement et déchiffrement asymétrique
$\dfrac{S \vdash \mathsf{sk}(a) \quad S \vdash m}{S \vdash [m]_{\mathsf{sk}(a)}}\ i \in \mathbb{N}$	$\dfrac{S \vdash [m]_{\mathsf{sk}(a)}}{S \vdash m}$	Signature
$\dfrac{S \vdash m}{S \vdash hash(m)}$		Fonction de hachage

FIG. 6.1 – Règles de déduction pour l'algèbre T.

et $\mathsf{sk}(s)$, ainsi que les offres $off(s, r)$ qu'un site propose pour des requêtes $r \in R$ ne sont a priori pas déductibles.

Les règles pour la composition et décomposition des paires, le chiffrement et déchiffrement asymétrique et le hachage sont les règles habituelles. Notons que la deuxième règle de la ligne *Signature* nous permet de récupérer le terme m à partir de la connaissance d'une signature $[m]_{\mathsf{sk}(a)}$. Cette convention nous facilite la modélisation de certains protocoles.

Les règles *Composition et décomposition des listes* permettent de construire une liste $m::l$ à partir d'un terme $m \in T$ et d'une liste $l \in L$ déductible, ainsi que de décomposer une liste $m::l$ en sa tête m et sa queue l.

6.2 Les protocoles

Nous avons besoin de travailler récursivement sur des termes. Nous modélisons ce traitement par des programmes que nous notons $P_1, P_2, \ldots$. Un programme peut être utilisé plusieurs fois pendant l'exécution d'un protocole. Pour distinguer plus tard les différents exécutions d'un programme, nous associons à chaque programme un ensemble de constantes. Une constante peut être considérée comme *instance du programme*. Les programmes sont définis par :

Définition 27. Un *programme* est une fonction $P : T \to T_x^{a,n}$. Nous notons *Prog* l'ensemble des programmes.

Les programmes transforment un terme $m \in T$ en un autre terme $m' \in T_x^{a,n}$. Les constantes *accept* et *not_accept* sont utiles pour exprimer le fait qu'un programme accepte une entrée ou ne l'accepte pas. Notons qu'il n'est pas prévu qu'un programme obtienne une des constants *accept* ou *not_accept* comme entrée. Le terme m' peut aussi contenir la variable x.

Avant que le site renvoie le terme m', cette variable est instanciée par la constante p qui représente l'instance du programme que le site a exécuté pour calculer le terme m'. Le site émet donc le terme $m'[x \leftarrow p]$. Ce terme se distingue au moins par la constante p de tous les autres termes que les sites peuvent engendrer en utilisant ce programme. Les instances peuvent donc être utilisées pour générer des nonces. Nous expliquons ultérieurement l'utilisation de la variable x et des instances des programmes pour la génération des nonces plus en détails. Nous donnons d'abord un exemple de programme :

Exemple 20. Prenons comme exemple un protocole qui collecte des offres pour une requête. Les messages sont de la forme $[r, [s_1, \ldots, \breve{s}_i, s_{i+1}, \ldots, s_n], [\mathit{off}(s_1, r), \ldots, \mathit{off}(s_{i-1}, r)]]$ où r représente la requête, la liste $[s_1, \ldots, \breve{s}_i, s_{i+1}, \ldots, s_n]$ représente les sites à visiter et $[\mathit{off}(s_1, r), \ldots, \mathit{off}(s_{i-1}, r)]$ représente la liste des offres déjà obtenues. Le prochaine site s_i qui doit ajouter son offre est marqué par le symbole fonctionnel $\breve{}$. Le programme P qui ajoute l'offre est donné par

Fonction P

Entrées : $m \in T$
Sorties : $m' \in T_x^{a,n}$
begin
  si $m = [r, [s_1, \ldots, \breve{s}_i, s_{i+1}, \ldots, s_n], [\mathit{off}(s_1, r), \ldots, \mathit{off}(s_{i-1}, r)]]$ **alors**
    $m' = [r, [s_1, \ldots, s_i, \breve{s}_{i+1}, \ldots, s_n], [\mathit{off}(s_1, r), \ldots, \mathit{off}(s_{i-1}, r), \mathit{off}(s_i, r)]]$
  si $m = [r, [s_1, \ldots, s_{n-1}, \breve{s}_n], [\mathit{off}(s_1, r), \ldots, \mathit{off}(s_{n-1}, r)]]$ **alors**
    $m' = [r, [s_1, \ldots, s_n], [\mathit{off}(s_1, r), \ldots, \mathit{off}(s_n, r)]]$
  sinon
    $m' = \mathit{not\_accept}$
  retourner m'
end

Dans une exécution d'un protocole, on n'exécute pas toujours le même programme. On peut par exemple avoir le cas où on exécute d'abord le programme P_1, ensuite trois fois le programme P_2, puis le programme P_3 et finalement encore une fois le programme P_1. Les différentes exécutions des programmes se font sur des différents sites avec des différents termes comme entrées. Pour pouvoir décrire le changement du programme exécuté d'un site à l'autre, nous avons besoin de conditions.

Définition 28. Nous notons 1 pour *vrai* et 0 pour *faux*. Une *condition* est une fonction $C : T \rightarrow \{0, 1\}$. Nous notons *Cond* l'ensemble des conditions.

Une condition prend un terme comme entrée et renvoie 1, si le terme est formé d'une certaine manière, sinon elle renvoie 0. Pour pouvoir capturer tous les changements possibles entre des programmes P_i et P_j, $i, j \in \{1, \ldots, k\}$, nous avons besoin de k^2 conditions. Nous pouvons maintenant définir les protocoles.

Définition 29. Un *protocole à k programmes* est un couple

$$((P_1, \ldots, P_k), ((C_{1,1}, \ldots, C_{1,k}), \ldots, (C_{k,1}, \ldots, C_{k,k}))) \in \mathit{Prog}^k \times \mathit{Cond}^{k^2}$$

des programmes et des tuplets des conditions. Pour un terme $m \in T$ et un $i \in \{1, \ldots, k\}$, nous avons au plus un $j \in \{1, \ldots, k\}$, tel que $C_{i,j}(m) = 1$.

La condition que nous avons imposée sur les k-uplets des conditions est nécessaire, parce que nous voulons que le changement entre les programmes soit déterministe. Si le programme P_i est

exécuté avec le terme m comme entrée, il doit avoir au plus un prochain programme P_j possible, où $1 \leq j \leq k$. S'il n'y a aucune condition $C_{i,j}$ qui est vraie pour le terme m après avoir exécuté le programme P_i, alors l'exécution du protocole est terminée. Nous formalisons cette situation dans les prochains paragraphes.

Exemple 21. Soit $\Pi = ((P_1, P_2), ((C_{1,1}, C_{1,2}), (C_{2,1}, C_{2,2})))$ un protocole et $m \in T$ un terme. Le changement entre les programmes P_1 et P_2 se fait de la manière suivante :

- Si le dernier programme exécuté est P_1 et $C_{1,1}(m) = 1$, alors le prochain programme à exécuter est aussi P_1.
- Si le dernier programme exécuté est P_1 et $C_{1,2}(m) = 1$, alors le prochain programme à exécuter est P_2.
- Sinon l'exécution s'arrête et symétriquement pour le programme P_2...

Pour pouvoir formellement définir ces changements des programmes, nous avons encore besoin de la notion d'historiques d'exécution que nous présentons ultérieurement.

6.3 Le modèle d'exécution

Le modèle d'exécution est un *modèle de transitions*. Les *transitions* se font entre des *états globaux*. Un état global contient les connaissances secrètes des agents corrompus, à savoir les clefs secrètes et les offres qu'il propose, ainsi que l'état actuel de chaque session. Les transitions sont considérées comme étant provoquées par l'intrus. Il y a des transitions pour corrompre des agents, commencer de nouvelles sessions et faire avancer l'exécution d'un protocole.

6.3.1 Les messages et l'exécution d'un programme

Soit $\Pi = ((P_1, \ldots, P_k), \ldots)$ un protocole. Rappelons que P_{Inst} est l'ensemble des constantes qui représentent les instances des programmes. À chaque programme P_i, nous associons un ensemble infini de constantes $S_{P_i} = \{p_1^i, p_2^i, \ldots\} \subseteq P_{Inst}$ qui nous servent à nous référer au programme P_i, c'est-à-dire que pour une constante $p \in S_{P_i}$, il est possible de retrouver P_i. Les ensembles sont définis de façon à ce que $S_{P_i} \cap S_{P_j} = \emptyset$, si $i \neq j$. Nous notons $S_P = \bigcup_i S_{P_i} \cup \{\perp\}$ l'ensemble qui contient toutes les instances des programmes d'un protocole et une constante spéciale $\perp$ qui est utilisée pour indiquer que l'exécution d'un protocole est terminée.

Définition 30. Un *message* est une paire dans $\mathcal{M} = S_P \times T$.

L'intuition derrière cette définition est la suivante : un site reçoit le message (p, m). Il identifie l'ensemble S_{P_i}, tel que $p \in S_{P_i}$. Ensuite, il exécute le programme P_i avec le terme m comme entrée.

Définition 31. Nous notons $P_i(m)$ l'*exécution du programme P_i avec le terme m* comme entrée.

6.3.2 Les historiques d'exécution

Soit $m_1 \in T$ un terme. Avec les programmes $P_1, \ldots, P_k$ d'un protocole, nous pouvons décrire une partie de son exécution avec comme entrée m_1 par $P_{i_1}(m_1) = m_2, P_{i_2}(m_2) = m_3, \ldots, P_{i_n}(m_n) = m_{n+1}$ où $i_1, \ldots, i_n \in \{1, \ldots, k\}$. Notons que cette exécution ne correspond pas nécessairement à l'utilisation pour laquelle le protocole a été conçu. L'adversaire pourrait essayer de faire exécuter une autre suite de programmes que celle prévue. Nous appelons cette suite des programmes avec leurs entrées respectives un historique d'exécution et nous la décrivons par une suite des messages.

Définition 32. Soit Π un protocole. Une séquence dans $ExecHist = \mathcal{M}^*$ est appelée un *historique d'exécution de* Π.

Un historique d'exécution correspond à une session. Nous appelons un message (p_i, m_i) d'un historique d'exécution la *i-ième étape* et le terme m_i est appelé le *résultat de l'étape* $i-1$. Nous définissons l'ensemble des termes des messages d'un historique d'exécution.

Définition 33. Soit $h = (p_i, m_i)_{1 \leq i \leq n}$ un historique d'exécution. L'ensemble des *termes des messages de* h est défini par

$$Msg(h) = \{m_i \mid (p_i, m_i) \text{ est un message de } h\}.$$

Nous définissons $Msg(\emptyset) = \emptyset$ et, pour un ensemble $H \subseteq ExecHist$, $Msg(H) = \bigcup_{h \in H} Msg(h)$.

De même, nous définissons l'ensemble des instances dans un historique d'exécution qui représentent un programme.

Définition 34. Soit $h = (p_i, m_i)_{1 \leq i \leq n}$ un historique d'exécution. L'ensemble des *instances des programmes de* h est défini par

$$Inst(h) = \{p_i \mid (p_i, m_i) \text{ est un message de } h\}.$$

Nous définissons $Inst(\emptyset) = \emptyset$ et, pour un ensemble $H \subseteq ExecHist$, $Inst(H) = \bigcup_{h \in H} Inst(h)$.

6.3.3 La fonction $step_{\Pi}$

À l'arrivée d'un message (p, m), un site doit exécuter le programme adéquat. Pour ce faire, nous définissons une fonction d'exécution qui renvoie le résultat de ce calcul. Si le programme n'est pas défini pour le terme m, alors elle renvoie la constante not_accept par défaut.

Définition 35. Soient $\Pi = ((P_1, \ldots, P_k), (\ldots))$ un protocole. Soit $(p, m) \in \mathcal{M}$ un message. Nous définissons la *fonction d'exécution* $exec_{\Pi} : S_P \times T \to T^{a,n}$ par

$$exec_{\Pi}(p, m) = \begin{cases} P_1(m)[x \leftarrow p] & \text{si } p \in S_{P_1}, \\ \vdots & \vdots \\ P_k(m)[x \leftarrow p] & \text{si } p \in S_{P_k}. \end{cases}$$

Notons que, dans les termes renvoyés par les programmes, les occurrences de la variable x sont remplacées par l'instance actuel du programme. De cette façon, nous pouvons obtenir des nonces frais. Nous illustrons ce mécanisme par un exemple.

Exemple 22. Nous utilisons dans cet exemple une fonction *tail* qui renvoie la queue d'une liste. Pour le protocole $\Pi = ((P_1, \ldots), \ldots)$, nous définissons l'ensemble des instances S_{P_1}. Le programme P_1 est donné par

Fonction P_1

Entrées : $m \in T$
Sorties : $l \in T_x^{a,n}$
begin
  si *m n'est pas une liste* **alors**
    $l = not\_accept$
  sinon
    $i = 1$
    $l = [\,]$
    tant que $m \neq [\,]$ **faire**
      $l = nonce(x, i) :: l$
      $m = tail(m)$
      $i = i + 1$
  retourner l
end

Pour une instance $p \in S_{P_1}$ et le terme $[[\,], [\,], [\,]] \in T$, nous avons par exemple $exec_{\Pi}(p, [[\,], [\,], [\,]]) = P_1([[\,], [\,], [\,]])[x \leftarrow p] = nonce(x, 3) :: nonce(x, 2) :: nonce(x, 1) :: [\,][x \leftarrow p] = [nonce(p, 3), nonce(p, 2), nonce(p, 1)]$. Si nous veillons à ce que les instances du programme P_1 ne soient pas réutilisées dans les exécutions du protocole, nous aurons toujours des nonces frais.

Après l'exécution du programme adéquat avec le terme m comme entrée, un changement de programme doit éventuellement être effectué. Nous définissons une fonction de changement.

Définition 36. Soit $\Pi = ((P_1, \ldots, P_k), ((C_{1,1}, \ldots, C_{1,k}), \ldots, (C_{k,1}, \ldots, C_{k,k})))$ un protocole. Nous définissons la *fonction de changement* $change_{\Pi} : S_P \times T \times 2^{ExecHist} \rightarrow 2^{S_P}$ par

$$change_{\Pi}(p, m, H) = \begin{cases} S_{P_i} \setminus Inst(H) & \text{si } p \in S_{P_1} \text{ et } \exists i \in \{1, \ldots, k\}, \text{ tel que } C_{1,i}(m) = 1, \\ S_{P_i} \setminus Inst(H) & \text{si } p \in S_{P_2} \text{ et } \exists i \in \{1, \ldots, k\}, \text{ tel que } C_{2,i}(m) = 1, \\ \quad \vdots & \quad \vdots \\ S_{P_i} \setminus Inst(H) & \text{si } p \in S_{P_k} \text{ et } \exists i \in \{1, \ldots, k\}, \text{ tel que } C_{k,i}(m) = 1, \\ \{\perp\} & \text{sinon.} \end{cases}$$

Notons que la définition d'un protocole implique qu'il y a pour un $j \in \{1, \ldots, k\}$ et un terme $m \in T$ au plus un $i \in \{1, \ldots, k\}$, tel que $C_{j,i}(m) = 1$. Par conséquent, si ce i existe, l'ensemble $S_{P_i} \setminus Inst(H)$ renvoyé par la fonction $change_{\Pi}$ est bien déterminé. Si ce i n'existe pas ou $p = \perp$, l'ensemble $\{\perp\}$ est retourné. Cela signifie de fait un arrêt d'exécution du protocole.

À l'aide des fonctions $exec_{\Pi}$ et $change_{\Pi}$, nous définissons une fonction $step_{\Pi}$ qui calcule la prochaine étape de l'exécution d'un protocole Π pour un historique d'exécution donné. Pour ne pas réutiliser les instances des programmes qui existent déjà, nous passons aussi l'ensemble des historiques d'exécution déjà existantes. Une description détaillée de la fonction $step_{\Pi}$ est donnée dans le paragraphe suivant lors de l'explication des transitions.

Définition 37. Nous définissons la *fonction de la prochaine étape* $step_{\Pi} : ExecHist \times 2^{ExecHist} \rightarrow ExecHist$ pour un protocole Π. Le résultat $h \in ExecHist$ d'une application de $step_{\Pi}((p_i, m_i)_{1 \leq i \leq n}, H)$ est calculé comme suit :

Fonction $step_{\Pi}$

Entrées : $(p_i, m_i)_{1 \leq i \leq n} \in ExecHist$, $H \subseteq ExecHist$
Sorties : $h \in ExecHist$
begin
  si $p_n = \perp$ **alors**
    $h = (p_i, m_i)_{1 \leq i \leq n}$
  sinon
    $m = exec_{\Pi}(p_n, m_n)$
    si $m \in \{accept, not\_accept\}$ **alors**
      $h = (p_i, m_i)_{1 \leq i \leq n}(\perp, m)$
    sinon
      Choisir $p \in change_{\Pi}(p_n, m_n, H)$
      $h = (p_i, m_i)_{1 \leq i \leq n}(p, m)$
  retourner h
end

6.3.4 Les états globaux et les transitions

Définition 38. Un *état global* est une paire dans $\mathsf{GState} = 2^T \times 2^{ExecHist}$.

Soit (Sec, H) un état global. L'ensemble Sec est censé contenir les offres et clefs secrètes des agents corrompus. L'ensemble H contient, pour chaque session d'un protocole, un historique d'exécution qui décrit l'état actuel de cette session. L'adversaire peut provoquer les évènements suivants pour influencer l'exécution du protocole. Ces évènements sont les transitions.

Définition 39. Pour un protocole Π, nous définissons trois *transitions*.

1. Un site $s \in S$ peut être corrompu :

$$(Sec, H) \xrightarrow{\mathbf{corrupt}(s)} (Sec \cup \{\mathsf{dk}(s), \mathsf{sk}(s)\} \cup \{off(s, r) \mid r \in R\}, H).$$

2. L'adversaire peut créer de nouvelles sessions :

$$(Sec, H) \xrightarrow{\mathbf{new}(p,m)} (Sec, H \cup \{(p, m)\})$$

   où $p \in S_P \backslash Inst(H)$ et $m \in T$ est un terme, tel que $(Msg(H) \cup Sec) \backslash \{accept, not\_accept\} \vdash m$.

3. L'adversaire peut faire avancer l'exécution du protocole :

$$(Sec, H) \xrightarrow{\mathbf{advance}(h)} (Sec, H')$$

   où $h \in H$ et $H' = H \setminus \{h\} \cup \{h'\}$ avec $h' = step_{\Pi}(h, H)$.

L'ensemble des transitions est donné par $\mathsf{Trans} = (\{\mathbf{corrupt}\} \times S) \cup (\{\mathbf{new}\} \times S_P \times T) \cup (\{\mathbf{advance}\} \times ExecHist)$. Nous dénotons $\mathsf{Exec}(\Pi) \subseteq \mathsf{GState}(\mathsf{Trans} \times \mathsf{GState})^*$ l'ensemble des traces d'exécution, telles qu'elles commencent avec l'état global $(\emptyset, \emptyset)$ et que les transitions entre les états globaux sont les transitions définies ci-dessus.

Soit $\Pi = ((P_1, \ldots, P_k), ((C_{1,1}, \ldots, C_{1,k}), \ldots, (C_{k,1}, \ldots, C_{k,k})))$ un protocole. Nous expliquons maintenant la signification intuitive des transitions. Un exemple détaillé d'une trace d'exécution se trouve dans le paragraphe 7.2.4.

1. $(Sec, H) \xrightarrow{\mathbf{corrupt}(s)} (Sec', H')$: Cette transition permet à l'adversaire d'obtenir les clefs secrètes $\mathsf{dk}(s)$ et $\mathsf{sk}(s)$, ainsi que les offres $\mathit{off}(s, r)$ d'un site $s \in S$ pour toutes les requêtes $r \in R$. Notons que l'intrus peut corrompre à tout instant.

2. $(Sec, H) \xrightarrow{\mathbf{new}(p,m)} (Sec', H')$: La transition **new** initialise une nouvelle session. Pour ce faire, l'adversaire choisit un programme P_i du protocole qui est représenté par la constante $p \in S_{P_i}$. Cette constante peut être considérée comme instance du programme. Pour bien pouvoir distinguer les historiques d'exécution dans H, il est important qu'elle ne soit pas contenue dans H. Sinon, l'adversaire pourrait créer deux historiques d'exécution qui coïncident et, par conséquent, ne sont plus à distinguer, ce que nous ne voulons pas. L'adversaire doit aussi choisir un terme $m \in T$ avec lequel, comme entrée, le programme P_i est exécuté. Ce terme doit être déductible à partir de la connaissance de l'adversaire, à l'exception qu'il ne doit pas s'agir des constantes *accept* et *not_accept*.

3. $(Sec, H) \xrightarrow{\mathbf{advance}((p_i,m_i)_{1\leq i\leq n})} (Sec', H')$: L'adversaire peut choisir un historique d'exécution $h = (p_i, m_i)_{1\leq i\leq n} \in H$ et faire avancer l'exécution du protocole d'une étape. Cette transition correspond à l'envoi du message (p_n, m_n) à un site. L'historique d'exécution résultant est calculé à l'aide de la fonction step_{Π}.

   Dans le cas où $p_n = \perp$ nous avons $h' = h$. L'historique de session ne change pas, parce que l'exécution du protocole est déjà finie.

   Dans le cas général, c'est-à-dire si $p_n \in S_{P_i}$ pour $i \in \{1, \ldots, k\}$ et $m_n \in T$, l'historique d'exécution résultant $h' = (p_i, m_i)_{1\leq i\leq n}(p_{n+1}, m_{n+1})$ est calculé de façon à ce que nous ayons $m_{n+1} = P_i(m_n)$. Le prochaine programme P_j à exécuter, représenté par la constante $p_{n+1} \in S_{P_j}$, est choisi, parce la condition $C_{i,j}$ est vraie pour le terme m_n.

   Nous devons encore considérer deux cas particuliers : si $m_{n+1} \in \{\mathit{accept}, \mathit{not\_accept}\}$, alors nous avons $p_{n+1} = \perp$. Cela signifie que l'exécution du protocole finit à cette étape, soit correctement (*accept*), soit avec une erreur (*not_accept*). Le deuxième cas arrive, lorsque toutes les conditions $C_{i,j}$ sont fausses pour le terme m_n. Nous avons alors, $p_{n+1} = \perp$ par la définition de la fonction change_{Π}.

   Nous avons déjà dit que cette transition correspond à l'envoi du message (p_n, m_n) à un site. Notons qu'un site doit exécuter le programme P_i indiqué par l'instance p_n avec le terme m_n comme entrée, indépendemment de son statut de corruption. Cela est en apparence une restriction du pouvoir de l'intrus. Cependant, admettons que le site étant censé exécuter le programme P_i soit corrompu et l'intrus puisse exécuter un programme à son gré, même non défini dans le cadre du protocole. Sous l'hypothèse que le système de déduction de la figure 6.1 modélise tout ce qui est faisable, l'adversaire ne pourrait pourtant pas construire d'autres termes que ceux qu'il peut déduire à partir de $(\mathit{Msg}(H) \cup \mathit{Sec}) \setminus \{\mathit{accept}, \mathit{not\_accept}\}$, à l'exception de deux constantes *accept* et *not_accept* qu'un programme peut renvoyer. Nous concluons qu'une transition **new** en combinaison avec d'éventuelles transitions **advance** lui donne autant de pouvoir que le fait de pouvoir exécuter un programme quelconque avec le terme m_n. La restriction en question ne se montre être qu'une apparence.

6.4 Conclusion

Dans ce chapitre, nous avons présenté un modèle symbolique qui permet de spécifier formellement des protocoles récursifs. Considérant les trois types de récursivité que nous avons distingués dans le paragraphe 5.2.1, nous pensons qu'il y a pour chacun parmi eux des exemples

de protocoles que nous pouvons modéliser. Ensuite, notre modèle nous permettra d'étudier des questions de sécurité en présence d'un intrus puissant du type Dolev-Yao.

La conception même de notre modèle est modulaire : Premièrement, nous avons par exemple choisi une algèbre qui nous facilite l'étude de cas que nous avons en tête, mais rien ne nous empêche d'utiliser une autre algèbre. Nous sommes donc en mesure de modéliser convenablement des protocoles qui utilisent d'autres structures de données récursives, telles que les arbres, etc. Un deuxième aspect modulaire de notre modèle est le fait que nous n'avons pas de restriction sur la façon comment le traitement récursif doit être décrit. Dans notre étude de cas, nous préférerons de le décrire à l'aide d'un pseudo-langage. Néanmoins, nous pourrions aussi utiliser des règles de réécriture ou encore des automates. Cela peut s'avérer intéressant pour l'étude de questions de décidabilité.

Un dernier aspect intéressant de notre modèle est la façon dont les nonces sont modélisés, car il nous semble assez naturel de les faire dépendre de l'instance du programme dans lequel ils sont utilisés.

Chapitre 7

Étude de cas : le protocole P1

Le protocole que nous nous proposons d'étudier est appelé *publicly verifiable chained digital signature protocol* dans [KAG98]. Nous nous y référons par P1, abréviation qui à été choisie par les auteurs. C'est un protocole où ni le nombre de participants, ni la taille des messages est borné.

Dans un premier temps, nous présentons le protocole que les auteurs ont proposé. Cette description est très informelle et il nous reste à prendre certaines décisions pour pouvoir modéliser formellement ce protocole. Nous citons aussi les propriétés que le protocole satisfait selon les auteurs. Ensuite, nous mettons en avant une des propriétés, appelée *intégrité forte en avant*. L'argumentation informelle donnée par les auteurs pour justifier l'affirmation que le protocole satisfait cette propriété est basée sur un mécanisme appelé *relation de chaînage* que nous expliquons au cours du chapitre. Cependant, cette argumentation s'est avérée fautive et nous citons une attaque qui a été découverte dans [Rot01].

L'insuffisance apparente de l'argumentation informelle nous mène, dans la deuxième partie de ce chapitre, à proposer une modélisation formelle du protocole P1. À cette fin, nous utilisons le modèle défini dans le chapitre 6. Nous donnons un exemple de trace d'exécution qui montre que notre modélisation permet de retrouver l'attaque décrite précédemment.

La troisième partie du chapitre est consacrée à explorer des modifications qui puissent remédier au défaut que le protocole original présente à l'égard de la propriété de l'intégrité forte en avant. Les modifications sont, elles aussi, basées sur une relation de chaînage. Nous examinons d'une manière pratique comment nous pouvons nous servir de cette relation. Pour ce faire, nous proposons trois protocoles nouveaux et nous prouvons formellement une version légèrement modifiée de la propriété pour deux parmi eux, contrairement au troisième pour lequel une attaque existe. Nous donnons une brève description informelle de cette attaque.

Quoique le troisième protocole ne satisfasse pas la propriété de l'intégrité forte en avant, il nous permet pourtant de mieux comprendre les mécanismes qui nous ont permis de la prouver pour les deux autres. La dernière partie de ce chapitre présente une discussion de ces mécanismes.

7.1 Description informelle du protocole P1 et ses propriétés

Nous adoptons la terminologie suivante que nous avons déjà utilisée en partie dans le chapitre 6 : un *site* est une adresse sur le réseau où des données sont mises à disposition. Les sites sont construits de façon à ce que des programmes puissent communiquer avec eux. De la part d'un agent, la communication consiste en des *requêtes* et, de la part d'un site, en des *réponses*. Un agent est lancé par un *utilisateur* qui, lui-même, est aussi un site. Les données récupérées par un agent à la suite d'une requête sont appelées *offres*, car, dans notre contexte, les agents

demandent des prix des biens désirés par l'utilisateur. Les agents que nous considérons travaillent sur des listes. Les éléments d'une liste sont appelés les *maillons*. Dans le cas spécial où tous les maillons d'une liste sont des sites, nous parlons d'un *itinéraire*.

7.1.1 Description informelle

Nous décrivons informellement le protocole P1, présenté dans [KAG98]. Il s'y agit d'une modification d'un protocole présenté auparavant dans [Yee97]. Il a été conçu pour récupérer les différentes offres d'un bien qui existent sur les sites du réseau. Son fonctionnement est comme suit : un agent stocke une liste d'offres qu'il a obtenu comme réponse par des sites auxquels il a déjà rendu visite. Cette liste commence avec un maillon que l'utilisateur initial a créé lui-même. L'existence de ce maillon spécial se justifie par des raisons techniques. En arrivant à un nouveau site, l'agent transmet la requête du bien recherché. Comme réponse, le site lui transmet une offre. Le programme ajoute cette offre à la fin de la liste et continue son itinéraire. Après avoir obtenu toutes les offres, l'agent rentre chez l'utilisateur initial. Celui-ci peut maintenant, en examinant la liste des offres, choisir l'offre qui lui convient.

Nous dénotons des sites sur le réseau par $S_1, S_2, \ldots$. Chaque site S dispose des clefs de chiffrement asymétrique que nous désignons par $\mathsf{ek}(S)$ et $\mathsf{dk}(S)$. Dans ce paragraphe, nous stipulons que la clef $\mathsf{ek}(S)$ est utilisée comme clef publique et la clef $\mathsf{dk}(S)$ comme clef privée. Nous notons le chiffrement d'une donnée m par $\{m\}_{\mathsf{ek}(S)}$. Nous avons aussi besoin de la signature. Pour cette raisons nous associons à chaque site S deux clefs $\mathsf{sk}(S)$ et $\mathsf{vk}(S)$. La première clef est secrète et sert à signer. La deuxième est publique et peut être utilisée pour vérifier l'identité du site. Nous notons $[m]_{\mathsf{sk}(S)}$ le message m signé par le site S. Étant donné un message signé $[m]_{\mathsf{sk}(S)}$, nous convenons que n'importe qui peut connaître le contenu m. Nous notons $hash(m)$ le haché de m. Finalement, une liste est notée par le symbole [ ]. Par exemple, la liste $[S_1, \ldots, S_n]$ est un itinéraire.

Nous notons Π l'agent, c'est-à-dire le programme qui est envoyé à travers le réseau. Il stocke les informations nécessaires pour travailler avec la liste des offres déjà récupérées dans des variables internes dont nous ne nous occupons pas. Le programme sait par exemple lors d'un parcours de la liste qui est l'élément actuel, etc. Notons que ni [KAG98] ni [Yee97] n'indiquent d'où l'agent obtient les informations concernant son itinéraire. Il se peut même qu'il soit construit au fur et à mesure de son trajet à travers le réseau. Nous ne pouvons donc pas supposer que l'utilisateur connaît dès le début de la session les sites où l'agent se rendra.

L'étape récursive du protocole P1 peut être décrite comme suit :

$$S_i \to S_{i+1} \quad : \quad \Pi, [O_1, \ldots, O_i].$$

où $O_j = [\{o_j, r_j\}_{\mathsf{ek}(S_1)}, h_j]_{\mathsf{sk}(S_j)}$ pour $1 \leq j \leq i$, avec $h_j = hash(O_{j-1}, S_{j+1})$ et, dans le cas où $j = 1$, $h_1 = hash(r'_1, S_2)$. Les o_j représentent les offres transmises et les r_j qui apparaissent dans les O_j et dans h_1 sont des nonces choisis par les sites S_j. Les nonces servent à rendre le protocole invulnérable contre des attaques à texte clair choisi. La structure générale de cette liste est représentée dans la figure 7.1. La relation qui existe entre les valeurs h_j est appelée *relation de chaînage*. D'une côté, elle doit assurer que la valeur O_{j-1} ne peut pas être modifiée sans modifier la valeur O_j et, d'autre côté, que c'est vraiment le site S_{j+1} qui ajoute l'offre suivante. Une liste de maillons O_j qui satisfait la relation de chaînage est appelée *valide*. Notons qu'une relation de chaînage valide, ainsi que le connaissance des sites S_j qui ont signé les maillons O_j devrait permettre à l'utilisateur de reconstituer l'itinéraire de l'agent.

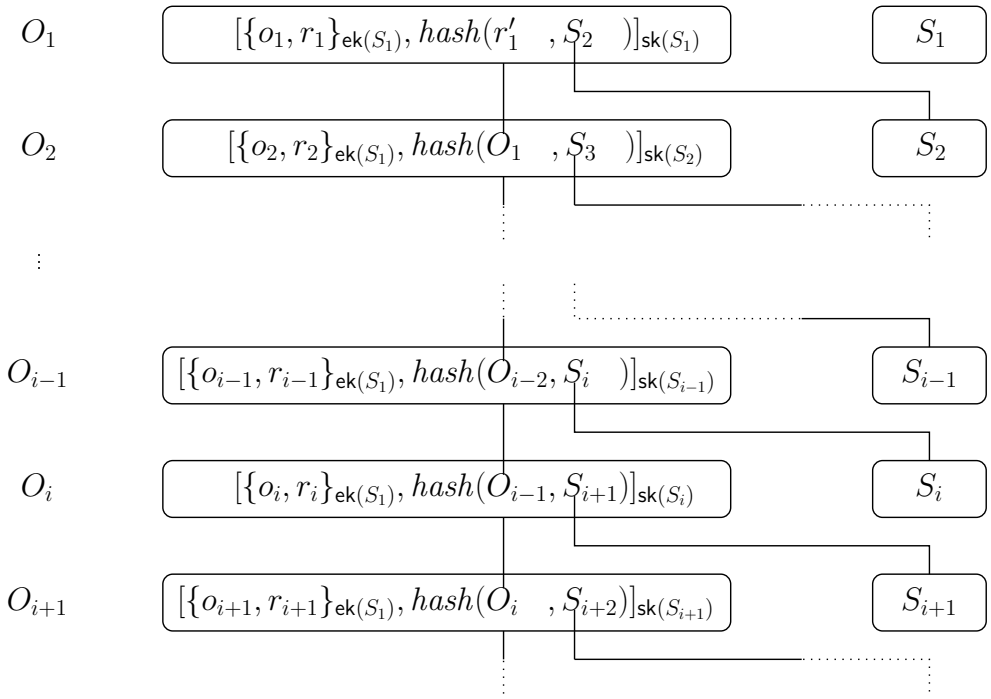


FIG. 7.1 – Chaque site S_i génère le maillon O_i. La figure explicite la relation de chaînage qui consiste en les valeurs de hachage contenues dans chaque maillon O_i. Une valeur de hachage contient en général le maillon précédent O_{i-1} et le site prochain S_{i+1}, la valeur de hachage du maillon O_1 contenant de façon exceptionnelle un nonce r'_1 à défaut d'un maillon précédent.

7.1.2 Les propriétés du protocole

Soit la liste d'offres $[O_1, \ldots, O_n]$ construite par le protocole P1. Les auteurs de [KAG98] revendiquent les propriétés suivantes pour le protocole P1. Toutefois, la définition des propriétés reste très informelle :

- *Confidentialité des données.* Seulement l'utilisateur S_1 peut connaître les offres o_i, $1 \leq i \leq n$.
- *Non-repudiabilité.* Un site S_i ne peut pas nier avoir construit le maillon O_i, $2 \leq i \leq n$.
- *Intégrité forte en avant.* Aucun maillon O_i, $i < n$, ne peut être modifié sans rompre la relation de chaînage, tant que le site S_n est honnête, c'est-à-dire qu'il ne révèle pas ses clefs secrètes.
- *Intégrité en avant publiquement vérifiable.* Tout le monde peut vérifier que la liste $[O_1, \ldots, O_n]$ satisfait la relation de chaînage.
- *Résilience contre une insertion.* On ne peut pas insérer de maillon à une position $i < n$ sans rompre la relation de chaînage.
- *Résilience contre le tronquage.* La liste $[O_1, \ldots, O_n]$ ne peut pas être tronquer à la position $i < n$, sauf si le site S_i est malhonnête.

7.1.3 Intégrité forte en avant

Dans la suite, nous nous intéressons à la propriété de l'intégrité forte en avant. Nous donnons d'abord l'argumentation informelle de [KAG98], selon laquelle le protocole P1 satisferait cette

propriété. Cela n'est cependant pas le cas comme l'attaque, trouvée par [Rot01], montre. Une vérification formelle aurait donc été souhaitable, afin que cette attaque ne passe pas inaperçue aussi longtemps.

Argumentation informelle fautive

Nous décrivons maintenant l'argumentation informelle de [KAG98]. Cette argumentation sert aux auteurs pour étayer l'assertion que le protocole P1 satisfait la propriété de l'intégrité forte en avant. Certaines prémisses de l'argumentation sont assumée d'une manière sous-entendue. L'argument principal est d'une nature inductive. Nous commençons d'abord à expliciter les prémisses.

La première prémisse de l'argumentation est l'hypothèse que chaque site propose toujours la même offre, lorsqu'on lui transmet la même requête. Cela n'est évidemment pas le cas dans un marché qui obéit à la loi de l'offre et de la demande. Cependant, cette hypothèse est nécessaire, parce qu'un site malhonnête pourrait tronquer une liste d'offres $[O_1, \ldots, O_i, \ldots O_n]$ à la position $i-1$ et l'envoyer avec son agent au site S_i. On obtiendrait ainsi une liste $[O_1, \ldots, O'_i, \ldots O'_n]$ où on n'a pas forcément $O_k = O'_k$ pour $i \leq k \leq n$, bien que le site S_n soit honnête, car les offres peuvent changer en fonction du temps.

La deuxième prémisse qui semble avoir été oubliée par les auteurs concerne les nonces qui apparaissent dans les maillons de la liste. Il est clair que nous devons imposer la même condition sur les nonces que celle que nous avons imposée sur les offres, c'est-à-dire qu'un site utilise toujours les mêmes nonces, lorsqu'on lui demande de chiffrer et signer son maillon. Autrement, nous avons le même problème de tronquage que nous venons de décrire ci-dessus. Qu'un nonce soit répété semble cependant contraire au concept de nonces qui prévoit qu'il ne soit pas répété. Rappelons aussi que, selon les auteurs, les nonces sont utilisés pour prévenir des attaques à texte clair choisi. Or, un nonce qui est toujours répété ne peut pas remplir cette fonction.

La dernière prémisse qui est nécessaire pour établir l'intégrité forte en avant pour le protocole P1 est la croyance que le fait que le site S_n qui ajoute le maillon O_n est honnête implique que le maillon O_n ne peut pas être changé par l'adversaire. Nous verrons dans le prochain paragraphe que cette implication n'est pas vraie. L'attaque qui y est présentée montre que cette croyance est fausse.

Nous présentons maintenant l'argument inductif qui sert à établir l'intégrité forte en avant pour le protocole P1 : soit $[O_1, \ldots, O_n]$ une liste d'offres construite par le protocole P1 et soit S_n un site honnête. Supposons qu'un intrus remplace le maillon O_{n-1} par O'_{n-1}. Sans pouvoir modifier le maillon O_n, la relation de chaînage doit toujours être satisfait pour la liste $[O_1, \ldots, O'_{n-1}, O_n]$. Le maillon O_n a la forme $[\{o_n, r_n\}_{\mathsf{ek}(S_1)}, h_n]_{\mathsf{sk}(S_n)}$ où $h_n = \mathit{hash}(O_{n-1}, S_{n+1})$. Pour que la relation de chaînage soit satisfaite, nous devons avoir $\mathit{hash}(O'_{n-1}, S_{n+1}) = \mathit{hash}(O_{n-1}, S_{n+1})$. Par l'hypothèse qu'on fait normalement sur les fonctions de hachage idéales, il est impossible de trouver une valeur x pour une valeur y donnée, telle que $\mathit{hash}(x) = y$. Il est donc impossible de trouver le maillon O'_{n-1} et de remplacer O_{n-1} sans modifier le maillon O_n. Le même raisonnement s'applique aux maillons O_i, $1 \leq i < n-1$.

Notons que la propriété forte en avant n'est défini qu'à l'égard d'un seul agent. Soient par exemple $[O_1, \ldots, O_n]$ et $[O'_1, \ldots, O'_n]$ deux listes dont les offres ont été collectées par deux agents et respectivement été créées par les sites $S_1, \ldots, S_n$ et $S'_1, \ldots, S'_n$. Bien qu'il se puisse que le site $S_n = S'_n$ soit honnête, cela ne garantit pas que nous avons $O_i = O'_i$ pour $1 \leq i < n$, parce qu'il peut y avoir deux sites S_i et S'_i tels que $S_i \neq S'_i$. Dans ce cas, les maillons O_i et O'_i ne sont pas forcément égaux.

Remarquons également que l'intégrité fort en avant n'exclut pas que l'adversaire peut tronquer la liste pour en obtenir une autre avec un itinéraire modifié. Cependant, si le maillon O_i ne peut pas être modifié, alors, à la suite d'un tronquage, l'agent qui collecte les offres doit toujours prendre le même itinéraire pour préserver la relation de chaînage.

Attaque

La preuve donnée ci-dessus repose sur la croyance que le dernier maillon de la liste partielle $[O_1, \ldots, O_n]$ ne peut pas être modifié, tant que l'intrus ne connaît pas la clef de signature du site S_n. Cela n'est cependant pas le cas comme l'attaque de [Rot01] le montre.

Considérons le scénario suivant : soit $[O_1, \ldots, O_n]$ une liste d'offres construite par le protocole P1. Les offres ont été proposées par les sites $S_1, \ldots, S_n$ où S_n est un site non corrompu. L'agent Π arrive avec cette liste à un site malhonnête S. Ce site choisit un entier $i \in \{1, \ldots, n\}$ et un nouveau site S'_{i+1}. Il envoie son propre agent Π_S pour demander au site S_i une nouvelle offre :

$$\begin{aligned} S \rightarrow S_i &: \Pi_S, [O_1, \ldots, O_{i-1}] \\ S_i \rightarrow S'_{i+1} &: \Pi_S, [O_1, \ldots, O_{i-1}, O'_i] \\ S'_{i+1} \rightarrow S &: \Pi_S, [O_1, \ldots, O_{i-1}, O'_i, O'_{i+1}] \end{aligned}$$

Après avoir reçu la liste $[O_1, \ldots, O_{i-1}, O'_i, O'_{i+1}]$, le site S enlève le maillon O'_{i+1}, puis il pose $i' = i + 1$ et choisit un nouveau site $S'_{i'+1}$. Maintenant, l'étape décrite recommence avec, respectivement, les sites S'_{i+1} et $S'_{i'+1}$ dans les rôles des site S_i et le site S'_{i+1}. Après avoir fait construire la liste $[O_1, \ldots, O'_i, \ldots, O'_k]$ où k est un entier qui n'est pas forcément égal à n et $O'_k = [\{o'_k, r'_k\}_{\mathsf{ek}(S_1)}, h'_k]_{\mathsf{sk}(S'_k)}$ avec $h'_k = hash(O'_{k-1}, S)$, le site S ajoute son propre offre et envoie l'agent original Π avec cette liste au prochain site.

Cette procédure permet à un site malhonnête de changer une liste $[O_1, \ldots, O_i, \ldots, O_n]$, construite par des sites honnêtes $S_1, \ldots, S_n$ en une liste altérée $[O_1, \ldots, O'_i, \ldots, O'_n]$, construite par les mêmes sites, mais où nous n'avons pas forcément $O_k \neq O'_k$ pour $i \leq k \leq n$. L'inégalité est par exemple vraie, si l'agent Π_S ne transmet pas la même requête aux sites S_k. La liste altérée satisfait pourtant la relation de chaînage. Nous voyons que l'honnêteté du site S_n n'implique pas que le maillon O_n ne peut pas être changé. Par conséquent, elle n'implique pas l'intégrité forte en avant non plus.

L'attaque montre qu'un site malhonnête S peut se faire passer pour l'utilisateur S_1. C'est le cas parce que c'est ce site qui a signé la première offre O_1. Les sites qui ajoutent subséquemment leurs offres peuvent donc être amenés à penser que c'est S_1 qui leur transmet une requête. Un exemple simple pour exploiter cette attaque serait le cas où l'utilisateur S_1 demande des offres d'un bien moins cher qu'un prix p. Le site malhonnête S utilise son agent pour demander des offres pour un prix moins cher que 0. Ensuite, il ajoute son offre pour le prix p à la dernière position de la liste, avant de renvoyer l'agent Π à l'utilisateur S_1. Comme il n'y a pas d'offres moins chères que 0, il se trouve que le site S ait fait la seule offre.

Il y a deux conditions nécessaires pour que cette attaque fonctionne. Premièrement, il est important que le site malhonnête S puisse déterminer l'itinéraire de son agent. Sinon, l'agent Π_S n'irait peut-être pas au site S_i, ce qui rendre la relation de chaînage invalide. Deuxièmement, l'agent Π doit accepter que le site S change la liste des offres en une autre liste arbitraire avant qu'il ajoute son propre offre. D'après la description du fonctionnement des agents qui est donnée dans [KAG98], il n'est pas clair, si ces conditions sont satisfaites.

7.2 Modélisation formelle du protocole P1

Dans la suite, nous modélisons le protocole P1. Nous avons déjà mentionné certains points peu clairs dans [KAG98], mais étant donné que nous visons une modélisation formelle de ce protocole, nous sommes obligés de prendre des décisions sur ces points.

La première question qui se pose concerne l'itinéraire de l'agent Π. Deux variantes semblent a priori possibles : soit l'itinéraire est connu dès le début du trajet de l'agent, soit il se construit au fur et à mesure pendant le trajet. Nous préférons la première variante, car une construction successive de cette liste demanderait des hypothèses exigeantes sur l'infrastructure. Quoique des infrastructures semblables existent [DEW97, CM96], à notre connaissance aucune n'a été implantée pour le protocole P1. Notons que le fait que l'agent Π connaît l'itinéraire dès le début de son trajet n'implique pas que l'utilisateur connaît cet itinéraire. Il peut donc possiblement être modifié, sans que l'utilisateur s'en rende compte.

Deuxièmement, l'agent Π doit s'authentifier auprès des sites visités. La solution la plus naturelle nous semble celle où les sites utilisent la signature du premier maillon pour connaître le site qui a démarré la session du protocole. En outre, nous supposons que la requête transmise n'est ni chiffrée, ni signée. Les sites ne peuvent donc pas supposer qu'elle est authentique.

Dans notre modélisation, un message contient donc trois parties : la requête R, l'itinéraire $S_1, \ldots, S_n$ et la liste des offres déjà obtenues $O_0, \ldots, O_i$. Rappelons que l'utilisateur lui-même ajoute la première offre O_0 dans cette liste pour garantir une relation de chaînage correcte. Nous donnons le protocole P1 d'une manière plus précise, mais toujours informelle :

$$\begin{array}{lcll} \text{Étape } 0 & : & U \to S_1 & : & [R, [S_1, \ldots, S_n], [O_0]] \\ \text{Étape } i & : & S_i \to S_{i+1} & : & [R, [S_1, \ldots, S_n], [O_0, \ldots, O_i]] \\ \text{Étape } n & : & S_n \to U & : & [R, [S_1, \ldots, S_n], [O_0, \ldots, O_n]] \end{array}$$

7.2.1 Les programmes utilisés dans le protocole P1

Le programme P_1

Le programme P_1 est censé être exécuté par l'utilisateur du protocole. Il initialise le premier message. Il prend comme entrée une paire consistant en le nom de l'utilisateur et la requête, et une liste des sites dont on veut collecter les offres. Il émet ensuite le premier message qui lance l'exécution du protocole. Notons l'emploi de la fonction $\breve{}$, à l'aide de laquelle nous marquons le site qui doit ajouter son offre à l'étape prochaine.

Fonction P_1

Entrées : $\langle\langle u, r\rangle, [s_1, \ldots, s_n]\rangle$, où $u, s_1, \ldots, s_n \in S, r \in R$ et $[s_1, \ldots, s_n]$ n'est pas vide
Sorties : $m \in T_x^{a,n}$ ou *not_accept* si l'entrée n'a pas la forme attendue
begin
  $m = [r, [\breve{s}_1, \ldots, s_n], [[\{off(u, r), nonce(x, 1)\}_{\mathsf{ek}(u)}, hash(\langle nonce(x, 2), s_1\rangle)]_{\mathsf{sk}(u)}]]$
  retourner m
end

Le programme P_2

Le programme P_2 ajoute pendant la deuxième phase de l'exécution du protocole l'offre d'un site qui figure dans la liste des sites. Pour connaître le site au nom duquel il doit ajouter une offre, il s'oriente à l'aide de la fonction $\breve{}$ qui avance après d'une position dans la liste. Si le site

actuel est le dernier élément de la liste, alors le programme P_2 ôte ce marqueur. le programme P_2 est donné par

---

Fonction P_2

Entrées : $[r, \mathbf{s}, \mathbf{o}::o::[]]$ où $r \in R$, l'itinéraire $\mathbf{s}$ contient un $\breve{\cdot}$, o est la dernière offre et $\mathbf{o}$ sont toutes les offres ajoutées auparavant

Sorties : $m \in T_x^{a,n}$ ou *not_accept* si l'entrée n'a pas la forme attendue

begin

`/* nous dénotons` u `le site qui a signé la première offre de la` `*/`

`/* liste` $\mathbf{o}::o::[]$ `*/`

si $\mathbf{s} = [s_1, \ldots, \breve{s}_n]$ **alors**

$m = [r, [s_1, \ldots, s_n], \mathbf{o}::o::[\{off(s_n, r), nonce(x, 1)\}_{\mathsf{ek}(u)}, hash(\langle o, u\rangle)]_{\mathsf{sk}(s_n)}::[]]$

sinon

`/* s a la forme` $[\ldots, \breve{s}_i, s_{i+1}, \ldots]$ `où` $i < n$ `*/`

$m = [r, [\ldots, s_i, \breve{s}_{i+1}, \ldots],$
$\mathbf{o}::o::[\{off(s_i, r), nonce(x, 1)\}_{\mathsf{ek}(u)}, hash(\langle o, s_{i+1}\rangle)]_{\mathsf{sk}(s_i)}::[]]$

retourner m

end

Le programme P_3

Le programme P_3 est exécuté au nom de l'utilisateur. Il vérifie que la relation de chaînage est valide. Notons que la revendication de la propriété de l'intégrité forte en avant pour le protocole P1 dans [KAG98] est seulement basée sur le fait que la relation de chaînage est valide. Nous formulons donc le programme P_3 de façon à ce que l'itinéraire ne soit pas pris en compte pour valider les offres. Il est en tout cas a priori inconnu par l'utilisateur et non protégé contre des altérations éventuelles. Il n'est donc pas fiable et l'utilisateur ne peut pas baser sa vérification sur cette information. De même, nous ne faisons pas attention au site s et la requête r contenus dans chaque offre $off(s, r)$, mais nous vérifions seulement que chaque maillon contient vraiment une offre et pas un autre terme à sa place.

Fonction P_3

Entrées : $[m_1, m_2, o_0 :: \ldots :: o_n :: [\,]] \in T$
Sorties : $m \in \{accept, not\_accept\}$
begin
  /* pour des variables a, b, s, r, u, v, y */
  $m = accept$
  si o_0 *ne s'unifie pas avec le terme* $[\{off(s,r), v\}_{\mathsf{ek}(u)}, hash(\langle y, a\rangle)]_{\mathsf{sk}(b)}$
    ou o_1 *n'est pas signé par la clef* $\mathsf{sk}(a)$ **alors**
    $m = not\_accept$
  pour $i = 1$ **à** $n-1$ **faire**
    si o_i *ne s'unifie pas avec le terme* $[\{off(s,r), v\}_{\mathsf{ek}(u)} hash(\langle o_{i-1}, a\rangle)]_{\mathsf{sk}(b)}$
      ou o_{i+1} *n'est pas signé par la clef* $\mathsf{sk}(a)$ **alors**
      $m = not\_accept$
  si o_n *ne s'unifie pas avec le terme* $[\{off(s,r), v\}_{\mathsf{ek}(u)}, hash(\langle o_{n-1}, a\rangle)]_{\mathsf{sk}(b)}$ **alors**
    $m = not\_accept$
  retourner m
end

Le programme P_3 doit traiter le premier maillon o_0 de la liste séparément, car nous avons un nonce au lieu d'une offre précédente dans la fonction de hachage. Le dernier maillon o_n doit être traité séparément, parce qu'il n'est plus nécessaire de tester, si le prochain maillon est signé avec la clef correcte.

7.2.2 Les conditions utilisées dans le protocole P1

Les seules conditions intéressantes sont les conditions qui permettent d'aller du programme P_1 vers le programme P_2, de continuer avec le programme P_2 et d'aller du programme P_2 vers le programme P_3. Nous appelons ces trois conditions $C_{1,2}$, $C_{2,2}$ et $C_{2,3}$. Soit m un terme de la forme $[r, \mathbf{s}, \mathbf{o}]$ où $\mathbf{s} = [s_1, \ldots, s_n]$ avec $s_1, \ldots, s_n \in S$. Les conditions en question sont définies par

- $C_{1,2}(m') = 1$ si et seulement si m' est de la forme $\langle\langle u, r\rangle, [s_1, \ldots, s_n]\rangle$ où $r \in R$ et $u, s_1, \ldots, s_n \in S$,
- $C_{2,2}(m) = 1$ si et seulement si $\mathbf{s}$ est une liste de la forme $[\ldots, \breve{s}_i, s_{i+1}, \ldots]$ avec un seul $\breve{\cdot}$ et
- $C_{2,3}(m) = 1$ si et seulement si $\mathbf{s}$ est de la forme $[\ldots, \breve{s}_n]$ avec un seul $\breve{\cdot}$.

Toutes les autres conditions sont toujours fausses. Le protocole P1 est donc entièrement spécifié par

$$\mathrm{P1} = ((P_1, P_2, P_3), ((C_{1,1}, C_{1,2}, C_{1,3}), (C_{2,1}, C_{2,2}, C_{2,3}), (C_{3,1}, C_{3,2}, C_{3,3}))).$$

7.2.3 Intégrité forte en avant pour le protocole P1

Nous avons besoin de quelques définitions préliminaires pour nous faciliter la formulation de la propriété de l'intégrité forte en avant. Les auteurs de [KAG98] considèrent que deux maillons sont égaux, s'ils se distinguent au plus par les nonces. Nous définissons formellement cette notion d'égalité.

Définition 40. Soient $o = [\{v_1, v_2\}_{\mathsf{ek}(u)}, hash(t_1, t_2)]_{\mathsf{sk}(s)}$ et $o' = [\{v_1', v_2'\}_{\mathsf{ek}(u')}, hash(t_1', t_2')]_{\mathsf{sk}(s')}$ deux termes dans T. Nous disons que o et o' sont équivalents, noté $o \simeq o'$, si $v_1 = v_1'$, $u = u'$, $t_2 = t_2'$ et $s = s'$. Si t_1 ou t_1' n'est pas de la forme $nonce(p, i)$, nous exigeons aussi que $t_1 = t_1'$.

Deuxièmement, nous définissons des fonctions qui nous permettent d'extraire un maillon particulier d'une liste et de récupérer le site qui a signé un maillon particulier d'une liste. Finalement, nous définissions la longueur d'une liste.

Définition 41. Soit $m = [o_0, \ldots, o_n]$ une liste de maillons de la forme $[v]_{\mathsf{sk}(s)}$, signés respectivement par les clefs de $s_0, \ldots, s_n$. Nous définissons $OFF_i(m) = o_i$, $SITE_i(m) = s_i$ et $length(m) = n$.

Un terme doit avoir une certaine forme, afin qu'on puisse appliquer les fonctions *OFF*, *SITE* et *length*. Cette forme est la forme des termes m, tel que le programme P_3 l'accepte, c'est-à-dire que nous avons $P_3(m) = accept$. Le lemme suivant nous dit quand nous pouvons en tout cas appliquer les fonctions.

Lemme 18. *Soit* $tr \in \mathsf{Exec}(\mathrm{P1})$ *une trace d'exécution. Elle contient un état global* (Sec, H) *avec* $h = (p_1, m_1) \ldots (p_k, m_k)(\bot, accept) \in H$ *si et seulement si elle contient un état global* (Sec', H')*, tel que* $h' = (p_1, m_1) \ldots (p_k, m_k) \in H'$ *et une transition* $\mathbf{advance}(h')$ *et nous avons* $p_k \in S_{P_3}$ *et* $P_3(m_k) = accept$*.*

Preuve. $\Longrightarrow$: Nous considérons d'abord le cas particulier où $h = (\bot, accept)$. La seule façon d'obtenir un historique d'exécution qui ne consiste qu'en une seule étape est de passer par une transition $\mathbf{new}(\bot, accept)$. Or, la constante *accept* n'est pas déductible dans une telle transition. Nous concluons que ce cas n'existe pas.

Considérons maintenant le cas où $h = (p_1, m_1) \ldots (p_k, m_k)(\bot, accept)$ avec $k \geq 1$. La seule façon de modifier un historique d'exécution déjà existant est de passer par une transition **advance**. Il doit donc exister un état global (Sec', H') tel que $h' = (p_1, m_1) \ldots (p_k, m_k) \in H'$ et la transition $\mathbf{advance}(h')$ transforme h' en h. Nous avons $h = step_{\mathrm{P1}}(h', H')$. Pour que la dernière étape de h ait la forme $(\bot, accept)$, nous devons avoir $p_k \neq \bot$ et $accept = exec_{\mathrm{P1}}(p_k, m_k)$. Étant donné que le programme P_3 seul est susceptible de renvoyer la constante *accept*, nous concluons que $p_k \in S_{P_3}$ et $P_3(m_k) = accept$.

$\Longleftarrow$: Nous considérons un état global (Sec', H'), tel que $h' = (p_1, m_1) \ldots (p_k, m_k) \in H'$ avec $p_k \in S_{P_3}$ et $P_3(m_k) = accept$. Le fait qu'il existe une transition $\mathbf{advance}(h')$ implique que la trace *tr* contient un état global (Sec, H) tel que $h \in H$ avec $h = step_{\mathrm{P1}}(h', H')$. Étant donné que $p_k \in S_{P_3}$ et $P_3(m_k) = accept$, nous avons $h = (p_1, m_1) \ldots (p_k, m_k)(\bot, accept)$. □

Ainsi, si nous avons un historique d'exécution qui finit avec l'étape $(\bot, accept)$, alors les fonctions *OFF*, *SITE* et *length* sont bien définies pour le terme qui figure dans l'avant-dernière étape.

Nous définissons maintenant la propriété de l'intégrité forte en avant pour le protocole P1. Rappelons que cette propriété n'est définie qu'à l'égard d'un seul agent. Dans la situation, où nous comparons deux listes d'offres collectées par deux agents différents, il est évident qu'elles peuvent différer, parce que l'itinéraire de deux agents n'a par exemple pas été le même ou la requête a été différente. Nous comparons donc seulement deux listes dont les premiers maillons sont *syntaxiquement égaux*. L'égalité des nonces compris dans ces maillons garantit que nous ne comparons que des listes dont les premiers maillons ont été crées par la même instance d'un programme. Dans ce cas nous considérons que c'est le même agent qui a commencé à collecter les offres. Dans la définition, nous utilisons la fonction $min(\_, \_)$ qui retourne le minimum de deux entiers.

Définition 42. Nous disons que le protocole P1 satisfait la propriété de l'*intégrité forte en avant*, si et seulement si

– pour toutes les traces $tr \in \mathsf{Exec}(\mathrm{P1})$,
– pour tous les historiques $h = (p_1, m_1) \dots (p_k, m_k)$ $(\bot, accept)$ et $h' = (p'_1, m'_1) \dots (p'_{k'}, m'_{k'})$ $(\bot, accept)$ de tr, tels que $OFF_0(m_k)$ et $OFF_0(m'_{k'})$ sont syntaxiquement égaux,
– pour tous les $j \in \{0, \dots, min(length(m_k), length(m'_{k'}))\}$, tels que $SITE_j(m_k) = SITE_j(m'_{k'})$ si le site $SITE_j(m_k)$ n'est pas corrompu, alors $OFF_i(m_k) \simeq OFF_i(m'_{k'})$ pour $i \in \{0, \dots, j\}$.

7.2.4 Exemple d'une trace d'exécution du protocole P1

Nous montrons dans ce paragraphe qu'il est possible de retrouver dans notre formalisme l'attaque qui a déjà été décrite dans le paragraphe 7.1.3. Pour les trois programmes P_1, P_2 et P_3 du protocole, nous définissons les ensembles $S_{P_i} = \{p_j^i \mid j \in \mathbb{N}, i = 1, 2, 3\}$. Nous commençons par donner un exemple d'une exécution normale du protocole P1.

1. Initialisation.
$$(\emptyset, \emptyset) \xrightarrow{\mathbf{new}(p_1^1, m_1)} (\emptyset, \{h_1\})$$
où $h_1 = (p_1^1, m_1)$ avec $p_1^1 \in S_{P_1} \setminus Inst(\emptyset)$ et $\emptyset \vdash m_1 = \langle\langle u, r\rangle, [s_1, s_2]\rangle \in T$.
2. Étape.
$$(\emptyset, \{h_1\}) \xrightarrow{\mathbf{advance}(h_1)} (\emptyset, \{h_2\})$$
où $h_2 = step_{\mathrm{P1}}(h_1, \{h_1\}) = h_1(p_1^2, m_2)$ avec $m_2 = exec_{\Pi}(p_1^1, m_1) = P_1(m_1)[x \leftarrow p_1^1] = [\langle u, r\rangle, [\check{s}_1, s_2], [o_0]]$ où $o_0 = [\{off(u, r), nonce(p_1^1, 1)\}_{\mathsf{ek}(u)}, hash(\langle nonce(p_1^1, 2), s_1\rangle)]_{\mathsf{sk}(u)}$. Nous choisissons $p_1^2 \in change_{\mathrm{P1}}(p_1^1, m_1, \{h_1\}) = S_{P_2} \setminus Inst(h_1)$, parce que $p_1^1 \in S_{P_1}$ et $C_{1,2}(m_1) = 1$.
3. Étape.
$$(\emptyset, \{h_2\}) \xrightarrow{\mathbf{advance}(h_2)} (\emptyset, \{h_3\})$$
où $h_3 = step_{\mathrm{P1}}(h_2, \{h_2\}) = h_2(p_2^2, m_3)$ avec $m_3 = exec_{\mathrm{P1}}(p_1^2, m_2) = P_2(m_2)[x \leftarrow p_1^2] = [\langle u, r\rangle, [s_1, \check{s}_2], [o_0, o_1]]$ où $o_1 = [\{off(s_1, r), nonce(p_1^2, 1)\}_{\mathsf{ek}(u)}, hash(\langle o_0, s_2\rangle)]_{\mathsf{sk}(s_1)}$. Nous choisissons $p_2^2 \in change_{\mathrm{P1}}(p_1^2, m_2, \{h_2\}) = S_{P_2} \setminus Inst(h_1) = S_{P_2} \setminus \{p_1^1, p_1^2\}$, parce que $p_1^2 \in S_{P_2}$ et $C_{2,2}(m_2) = 1$.
4. Étape.
$$(\emptyset, \{h_3\}) \xrightarrow{\mathbf{advance}(h_3)} (\emptyset, \{h_4\})$$
où $h_4 = step_{\mathrm{P1}}(h_3, \{h_3\}) = h_3(p_1^3, m_4)$ avec $m_4 = exec_{\mathrm{P1}}(p_2^2, m_3) = P_2(m_3)[x \leftarrow p_2^2] = [\langle u, r\rangle, [s_1, s_2], [o_0, o_1, o_2]]$ où $o_2 = [\{off(s_2, r), nonce(p_2^2, 1)\}_{\mathsf{ek}(u)}, hash(\langle o_1, u\rangle)]_{\mathsf{sk}(s_2)}$. Nous choisissons $p_1^3 \in change_{\mathrm{P1}}(p_2^2, m_3, \{h_3\}) = S_{P_3} \setminus Inst(\{h_3\}) = S_{P_3} \setminus \{p_1^1, p_1^2, p_2^2\}$, parce que $p_2^2 \in S_{P_2}$ et $C_{2,3}(m_3) = 1$.
5. Étape.
$$(\emptyset, \{h_4\}) \xrightarrow{\mathbf{advance}(h_4)} (\emptyset, \{h_5\})$$
où $h_5 = step_{\mathrm{P1}}(h_4, \{h_4\}) = h_4(\bot, accept)$, parce que $exec_{\mathrm{P1}}(p_1^3, m_4) = P_3(m_4)[x \leftarrow p_1^3] = accept$.

À partir de cet exemple d'exécution normale, nous montons maintenant l'attaque. Nous utilisons le terme m_2 qui apparaît pendant la deuxième étape de la trace pour en déduire le terme $m'_1 = [\langle u, r'\rangle, [s_1, s_2], [o_0]]$. Ce terme se distingue du terme m_2 par la requête r'. Nous initialisons un nouvel historique d'exécution avec le terme m'_1. Ce historique peut être considéré comme un deuxième agent qui collecte des offres au nom de l'utilisateur u, mais dont la requête à été modifiée. À la fin, nous nous retrouvons avec les deux historiques h_5 et h'_4, qui violent la propriété de l'intégrité forte en avant. Voici la continuation de la trace.

6. Étape.
$$(\emptyset, \{h_5\}) \xrightarrow{\mathbf{new}(p_3^2, m_1')} (\emptyset, \{h_5, h_1'\})$$
avec $p_3^2 \in S_{P_2} \setminus Inst(\{h_5\}) = S_{P_2} \setminus \{p_1^1, p_1^2, p_2^2, p_1^3\}$ et $Msg(\{h_5\}) \setminus \{accept, not\_accept\} \vdash m_1' = [\langle u, r'\rangle, [\check{s}_1, s_2], [o_0]]$ avec $r' \neq r$. Nous avons donc $h_1' = (p_3^2, m_1')$.

7. Étape.
$$(\emptyset, \{h_5, h_1'\}) \xrightarrow{\mathbf{advance}(h_1')} (\emptyset, \{h_5, h_2'\})$$
où $h_2' = step_{\text{P1}}(h_1', \{h_5, h_1'\}) = h_1'(p_4^2, m_2')$ avec $m_2' = exec_{\text{P1}}(p_3^2, m_1') = P_2(m_1')[x \leftarrow p_3^2] = [\langle u, r'\rangle, [s_1, \check{s}_2], [o_0, o_1']]$ où $o_1' = [\{off(s_1, r'), nonce(p_3^2, 1)\}_{\mathsf{ek}(u)}, hash(o_0, s_2)]_{\mathsf{sk}(s_1)}$. Nous choisissons $p_4^2 \in change_{\text{P1}}(p_3^2, m_1', \{h_5, h_1'\}) = S_{P_2} \setminus Inst(\{h_5, h_1'\}) = S_{P_2} \setminus \{p_1^1, p_1^2, p_2^2, p_1^3, p_3^2\}$, parce que $p_3^2 \in S_{P_2}$ et $C_{2,2}(m_1') = 1$.

8. Étape.
$$(\emptyset, \{h_5, h_2'\}) \xrightarrow{\mathbf{advance}(h_2')} (\emptyset, \{h_5, h_3'\})$$
où $h_3' = step_{\text{P1}}(h_2', \{h_5, h_2'\}) = h_2'(p_2^3, m_3')$ avec $m_3' = exec_{\text{P1}}(p_4^2, m_2') = P_2(m_2')[x \leftarrow p_4^2] = [\langle u, r'\rangle, [s_1, s_2], [o_0, o_1', o_2']]$ où $o_2' = [\{off(s_2, r'), nonce(p_4^2, 1)\}_{\mathsf{ek}(u)}, hash(o_1', u)]_{\mathsf{sk}(s_2)}$. Nous choisissons $p_2^3 \in change_{\text{P1}}(p_4^2, m_2', \{h_5, h_2'\}) = S_{P_3} \setminus Inst(\{h_5, h_2'\}) = S_{P_3} \setminus \{p_1^1, p_1^2, p_2^2, p_1^3, p_3^2, p_4^2\}$, parce que $p_4^2 \in S_{P_2}$ et $C_{2,3}(m_2') = 1$.

9. Étape.
$$(\emptyset, \{h_5, h_3'\}) \xrightarrow{\mathbf{advance}(h_3')} (\emptyset, \{h_5, h_4'\})$$
où $h_4' = step_{\text{P1}}(h_3', \{h_5, h_3'\}) = h_3'(\bot, accept)$, parce que $exec_{\text{P1}}(p_2^3, m_3') = P_3(m_3')[x \leftarrow p_2^3] = accept$.

Nous montrons maintenant que cette trace d'exécution viole la propriété de l'intégrité forte en avant. Nous considérons les deux historiques h_5 et h_4'. Les termes m_4 et m_3' sont acceptés par le programme P_3. Nous avons l'égalité syntaxique entre $OFF_0(m_4)$ et $OFF_0(m_3')$. Nous choisissons $j = 2$ et nous avons $SITE_j(m_4) = SITE_j(m_3') = s_2$. Le site s_2 n'est pas corrompu. Mais, pour $i = 1$, nous avons $OFF_i(m_4) \not\approx OFF_i(m_3')$, car $off(s_1, r) \neq off(s_1, r')$, parce que nous avons choisi $r \neq r'$. Le protocole P1 ne satisfait donc pas la propriété de l'intégrité forte en avant.

7.3 Modifications du protocole P1

Nous proposons dans ce paragraphe trois modifications du protocole P1 que nous appelons P1bis, P1ter et P1quater. Pour ces protocoles, nous essayons de garantir une version modifiée de l'intégrité forte en avant. Nous expliquons cette version dans la suite. Pour les deux premiers protocoles, nous parvenons à prouver formellement qu'ils satisfont cette propriété, contrairement au protocole P1quater pour lequel nous montrons une attaque. Ce dernier protocole permet néanmoins de justifier les décisions de conception que nous avons prises à l'égard des deux premiers.

L'intégrité forte en avant modifiée

La description de l'intégrité forte en avant donnée dans le paragraphe 7.1.2 définit cette propriété par rapport à une seule liste $[O_1, \ldots, O_n]$. Si le maillon O_n ne peut pas être changé, alors aucun de maillons O_i, $1 \leq i < n$ peut être modifié. Les auteurs de [KAG98] pensent que l'honnêteté de l'agent S_n qui a ajouté le maillon O_n implique que ce maillon ne peut pas être modifié. Cette implication n'est cependant pas vraie, comme nous avons mentionné.

Nous avons modélisé cette propriété en considérant toutes les paires de listes dont le premier maillon est issu de la même session. Pour assurer cette condition, nous avons exigé que les premier maillons de deux listes comparées soient syntaxiquement égaux, c'est-à-dire que nous incluons les nonces dans notre comparaison. Nous comparons dans chacune des paires de listes les maillons qui ont été ajouté avant qu'un site honnête ajoute son maillons. Ces maillons doivent être équivalents dans les deux listes.

Outre que nous avons présenté une attaque, cette propriété nous semble assez particulière, parce que le bénéfice qu'un utilisateur pourrait attendre d'un protocole qui la satisfait n'est pas clair : premièrement, les auteurs admettent qu'un intrus peut tronquer la liste et en construire une différente, sans que l'utilisateur puisse s'en rendre compte. Deuxièmement, l'intrus pourrait lancer une deuxième session et retourner le résultat de celle-ci comme le résultat de la première session, sans que l'utilisateur s'en rende compte. À cause de la condition que le maillon O_n ne doit pas être modifié, ces deux cas ne sont cependant pas d'attaques contre l'intégrité forte en avant. La protection contre ces deux situations mentionnées nous semble pourtant être plus intéressante que l'intégrité forte en avant comme nous l'avons définie jusqu'alors.

Nous proposons donc une version modifiée de la propriété de l'intégrité forte en avant. Nous comparons toutes les paires de listes sans prendre en compte de quelle session elles sont issues. Si l'agent a fait le même itinéraire, alors les maillons des deux listes doivent être égaux jusqu'au premier site corrompu. Il est évident que nous ne pouvons rien garantir pour le maillon ajouté par un site corrompu, ainsi que les maillons qui suivent, parce que l'intrus peux commencer une deuxième session avec le même début de la liste et ajouter un maillon différent pour le site corrompu. Voici notre définition formelle :

Définition 43. Nous disons que le protocole $\Pi \in \{\text{P1bis}, \text{P1ter}, \text{P1quater}\}$ satisfait la propriété de l'*intégrité forte en avant*, si et seulement si

- pour toutes les traces $tr \in \mathsf{Exec}(\Pi)$,
- pour tous les historiques $h = (p_1, m_1) \dots (p_k, m_k)$ $(\perp, accept)$ et $h' = (p'_1, m'_1) \dots (p'_{k'}, m'_{k'})$ $(\perp, accept)$ de tr, tel que $length(m_k) = length(m'_{k'})$ et $SITE_l(m_k) = SITE_l(m'_{k'})$ où $0 \leq l \leq length(m_k)$,
- pour $i \in \{0, \dots, length(m_k)\}$, tel que $SITE_i(m_k)$ est le premier agent corrompu,

s'il existe un $j \in \{0, \dots, length(m_k)\}$, tel que $OFF_j(m_k) \neq OFF_j(m'_{k'})$, alors $j \geq i$.

Nous prenons seulement en compte des listes de la même longueur, pour la construction desquelles les agents ont fait un itinéraire identique. Cela nous facilite l'énoncé et les preuves de cette propriété, mais il nous semble suffisant de prendre en compte toutes les listes sans considérer leurs longueurs respectives, ainsi que leurs itinéraires complets, tant que l'agent a suivi le même itinéraire jusqu'au premier site corrompu.

7.3.1 Première modification : le protocole P1bis

Le protocole P1bis permet d'obtenir des offres pour une seule requête. Il utilise trois programmes, appelés $P_{1\text{bis}}$, $P_{2\text{bis}}$ et $P_{3\text{bis}}$. Le premier programme est exécuté par le site qui commence la session. Le deuxième programme est utilisé par les sites qui ajoutent leurs offres. Étant donné que nous n'admettons qu'une seule requête dans le protocole P1bis, il n'est pas nécessaire que les messages envoyées contiennent la requête, parce que les sites donnent les offres toujours pour la même requête. Nous fixons donc la requête dans les programmes $P_{1\text{bis}}$ et $P_{2\text{bis}}$. Le programme $P_{3\text{bis}}$ sert au site qui a commencé la session pour vérifier la liste des offres obtenues.

Les programmes et conditions

Le programme $P_{1\text{bis}}$ prend comme entrée le site u qui initialise la session et la liste des sites $[s_1, \ldots, s_n]$ dont il veut obtenir les offres. Il est superflu d'expliciter la requête, parce qu'il n'y a qu'une seule requête. Le programme renvoie le premier maillon de la liste des offres. Il est donné par

---

Fonction $P_{1\text{bis}}$

Entrées : $\langle u, [s_1, \ldots, s_n] \rangle$, où $u, s_1, \ldots, s_n \in S$ et $[s_1, \ldots, s_n]$ n'est pas vide
Sorties : $[[[s_1, \ldots, s_n]]_{\mathsf{sk}(u)}]$ ou *not_accept* si l'entrée n'a pas la forme attendue

---

Le programme $P_{2\text{bis}}$ est censé être exécuté par les sites dont on veut obtenir les offres. Il attend une liste d'offres comme entrée et ajoute l'offre du site actuel la liste des offres. Pour déterminer le site actuel, le programme compare la longueur de l'itinéraire$[s_1, \ldots, s_n]$ contenu dans le premier maillon avec la longueur de la liste des offres. Comme pour le programme $P_{1\text{bis}}$, il n'est pas nécessaire de donner une requête comme entrée, parce qu'il n'y en a qu'une seule possible. Le programme $P_{2\text{bis}}$ est donné par

---

Fonction $P_{2\text{bis}}$

Entrées : $o_0 :: \ldots :: o_k :: [\,]$ où o_0 a la forme $[[s_1, \ldots, s_n]]_{\mathsf{sk}(u)}$ avec $n > k$
Sorties : $m \in T_x^{a,n}$ ou *not_accept* si l'entrée n'a pas la forme attendue
begin
  $m = o_0 :: \ldots :: o_k :: [off(s_{k+1}, r), hash(o_k)]_{\mathsf{sk}(s_{k+1})} :: [\,]$
  retourner m
end

Le programme $P_{3\text{bis}}$ vérifie que les maillons ont respectivement été signés par les sites qui figurent dans l'itinéraire contenu dans le premier maillon et que la relation de chaînage est valide. Cette relation consiste en les termes de hachage que contiennent les maillons, à l'exception du premier. La relation est valide, si chaque terme de hachage contient le maillon précédant de la liste. Notons que le programme ne s'occupe pas des termes qui se trouvent dans la position des offres. Si la vérification est positive, alors le programme retourne la constante *accept*, sinon *not_accept*. Le programme $P_{3\text{bis}}$ est donné par

---

Fonction $P_{3\text{bis}}$

Entrées : $o_0 :: o_1 :: \ldots :: o_n :: [\,] \in T$
Sorties : $m \in \{accept, not\_accept\}$ ou *not_accept* si l'entrée n'a pas la forme attendue
begin
  $m = accept$
  si o_0 *ne s'unifie pas avec le terme* $[[s_1, \ldots, s_n]]_{\mathsf{sk}(u)}$
    ou *il existe un* o_i *qui n'est pas signé avec la clef* $\mathsf{sk}(s_i)$, $1 \leq i \leq n$, **alors**
    $m = not\_accept$
  pour $i = 1$ **à** n **faire**
    si o_i *ne s'unifie pas avec* $[x, hash(o_{i-1})]_{\mathsf{sk}(s)}$ **alors**
      $m = not\_accept$
  retourner m
end

Il nous reste à spécifier les conditions qui permettent de changer entre les exécutions des programmes $P_{1\text{bis}}$, $P_{2\text{bis}}$ et $P_{1\text{bis}}$. Soit m un terme de la forme $o_0 :: \ldots :: o_k :: [\,]$ et soit $[s_1, \ldots, s_n]$

la liste des sites à visiter contenue dans le maillon o_0. En général, toutes les conditions $C_{i,j\text{bis}}$, $1 \leq i, j \leq 3$, sont fausses, sauf les suivantes :

- $C_{1,2\text{bis}}(m') = 1$ si et seulement si m' a la forme $\langle u, [s_1, \ldots, s_n] \rangle$,
- $C_{2,2\text{bis}}(m) = 1$ si et seulement si $k < n - 1$,
- $C_{2,3\text{bis}}(m) = 1$ si et seulement si $k = n - 1$.

Le protocole P1bis est entièrement spécifié par

$$\text{P1bis} = ((P_{1\text{bis}}, P_{2\text{bis}}, P_{3\text{bis}}), ((C_{1,1\text{bis}}, C_{1,2\text{bis}}, C_{1,3\text{bis}}), (C_{2,1\text{bis}}, C_{2,2\text{bis}}, C_{2,3\text{bis}}), (C_{3,1\text{bis}}, C_{3,2\text{bis}}, C_{3,3\text{bis}}))).$$

Preuve de l'intégrité forte en avant

Nous prouvons maintenant que le protocole P1bis satisfait la propriété de l'intégrité forte en avant, donnée par la définition 43. Pour ce faire, nous prouvons d'abord deux lemmes auxiliaires. Le premier lemme est une version du lemme 18 qui nous permet d'identifier les situations dans lesquelles nous pouvons appliquer les fonctions *OFF*, *SITE* et *length* à un terme. Une telle situation est notamment donnée, si le programme $P_{3\text{bis}}$ renvoie *accept* pour un terme. Le deuxième lemme auxiliaire nous dit qu'il est impossible pour l'adversaire de déduire la clef de signature d'un site, si le site n'est pas corrompu. Basés sur ces deux lemmes, nous pouvons enfin prouver la propriété souhaitée.

Lemme 19. *Soit $tr \in \mathsf{Exec}(\text{P1bis})$ une trace d'exécution. Elle contient un état global (Sec, H) avec $h = (p_1, m_1) \ldots (p_k, m_k)(\perp, accept) \in H$ si et seulement si elle contient un état global (Sec', H'), tel que $h' = (p_1, m_1) \ldots (p_k, m_k) \in H'$ et une transition* **advance**(h') *et nous avons $p_k \in S_{P_{3\text{bis}}}$ et $P_{3\text{bis}}(m_k) = accept$.*

Preuve. La preuve est analogue à la preuve du lemme 18. □

Lemme 20. *Soit $tr \in \mathsf{Exec}(\text{P1bis})$ une trace d'exécution. Si un site $s \in S$ n'apparaît pas dans une transition* **corrupt** *de tr, alors il n'existe pas d'état global (Sec, H) dans cette trace, tel que $Msg(H) \cup Sec \vdash \mathsf{sk}(s)$.*

Preuve. Soit $s \in S$ un site qui n'apparaît pas dans une transition **corrupt** d'une trace $tr \in \mathsf{Exec}(\text{P1bis})$. Nous montrons le lemme par induction sur les états globaux qui apparaissent dans tr. Plus précisément, nous montrons la propriété suivant : soit $(Sec, H) \xrightarrow{\textbf{trans}} (Sec', H')$ une transition de la trace tr. Si nous n'avons pas $Msg(H) \cup Sec \vdash \mathsf{sk}(s)$, alors nous n'avons pas $Msg(H') \cup Sec' \vdash \mathsf{sk}(s)$.

- L'état $(\emptyset, \emptyset)$ est le premier état global de la trace tr. Nous avons $Msg(\emptyset) \cup \emptyset \nvdash \mathsf{sk}(s)$.
- Nous considérons les transitions $(Sec, H) \xrightarrow{\textbf{corrupt}(s')} (Sec', H)$ de tr où $s' \neq s$ et $Msg(H) \cup Sec \nvdash \mathsf{sk}(s)$. Étant donné que l'ensemble Sec' ne permet pas de déduire un terme qui contient le sous-terme $\mathsf{sk}(s)$, l'adversaire doit utiliser un terme retourné par le programme $P_{1\text{bis}}$ ou $P_{2\text{bis}}$ pour déduire à partir de $Msg(H) \cup Sec'$ un terme qui contient le sous-terme $\mathsf{sk}(s)$. Dans les termes renvoyés par ces programmes, la clef $\mathsf{sk}(s)$ est toujours dans une position de clef. Il n'y a pas de règle de déduction qui permet de la déduire de cette position. Les connaissance que l'adversaire acquiert pendant la transition **corrupt**(s') ne peuvent donc pas servir non plus pour la déduire. Nous avons donc $Msg(H) \cup Sec' \nvdash \mathsf{sk}(s)$. Si nous n'avons pas $Msg(H) \cup Sec \vdash \mathsf{sk}(s)$, alors nous n'avons pas $Msg(H) \cup Sec' \vdash \mathsf{sk}(s)$.
- Quant aux transitions $(Sec, H) \xrightarrow{\textbf{new}(p,m)} (Sec, H')$ de tr, si nous n'avons pas $Msg(H) \cup Sec \vdash \mathsf{sk}(s)$, alors nous n'avons pas $Msg(H') \cup Sec \vdash \mathsf{sk}(s)$, parce que les ensembles $Msg(H) \cup Sec$ et $Msg(H') \cup Sec$ permettent de déduire exactement le même ensemble de termes.

– Considérons maintenant les transitions $(Sec, H) \xrightarrow{\textbf{advance}(h)} (Sec, H')$ de tr où $h \in H$ et $H' = H \setminus \{h\} \cup \{h'\}$ avec $h' = step_{\text{P1bis}}(h, H)$. Les programmes P_{1bis}, P_{2bis} et P_{3bis} peuvent être exécutés pour calculer h'. Les deux premiers renvoient des termes où les clefs sont seulement dans des positions, telles qu'il n'y a pas de règle de déduction pour les déduire. Le troisième ne renvoie que les constantes *accept* et *not_accept*. Également, si nous n'avons pas $Msg(H) \cup Sec \vdash \mathsf{sk}(s)$, alors nous n'avons pas $Msg(H') \cup Sec \vdash \mathsf{sk}(s)$.

Nous concluons qu'il n'y a aucun état global (Sec, H) dans tr, tel que $Msg(H) \cup Sec \vdash \mathsf{sk}(s)$. □

Nous venons maintenant à la preuve de la propriété de l'intégrité forte en avant pour le protocole P1bis. Cette preuve se fait par récurrence.

Théorème 4. *Le protocole* P1bis *satisfait la propriété de l'intégrité forte en avant de la définition 43.*

Preuve. Soient $m_k = [o_0, \ldots, o_n]$ et $m'_{k'} = [o'_0, \ldots, o'_n]$ les deux termes qui apparaissent dans la trace $tr \in \mathsf{Exec}(\text{P1bis})$ de la définition de la propriété forte en avant pour le protocole P1bis. Nous pouvons être sûrs de la forme des termes m_k et $m'_{k'}$, parce que, d'après le lemme 19, nous avons $P_{\text{3bis}}(m_k) = accept$ et $P_{\text{3bis}}(m'_{k'}) = accept$. Soit $i \in \{0, \ldots, n\}$ tel que $SITE_i(m_k)$ est le premier site corrompu.

Dans le cas particulier où $i = 0$, il est toujours vrai que $j \in \{0, \ldots, n\}$ est plus grand ou égal à i. Soit donc $i \in \{1, \ldots, n\}$. Les sites $SITE_l(m_k)$, $0 \leq l < i$, ne sont pas corrompus. Selon le lemme 20, il n'y aucun état global (Sec, H) de tr, tel que $Msg(H) \cup Sec \vdash \mathsf{sk}(SITE_l(m_k))$.

Nous montrons la propriété par récurrence.

– Les maillons o_0 et o'_0 doivent être de la forme $[[s_1, \ldots, s_n]]_{\mathsf{sk}(s)}$ et $[[s'_1, \ldots, s'_n]]_{\mathsf{sk}(s')}$, pour que les termes m_k et $m'_{k'}$ soient finalement acceptées par le programme P_{3bis}. Nous devons montrer que $o_0 = o'_0$. Par la condition que nous avons imposée sur m_k et $m'_{k'}$ dans la définition 43, nous avons $SITE_0(m_k) = SITE_0(m_{k'})$, c'est-à-dire $s = s'$. Il reste à démontrer que $[s_1, \ldots, s_n] = [s'_1, \ldots, s'_n]$. Cela est le cas, parce que les termes m_k et $m'_{k'}$ sont acceptés par le programme P_{3bis} qui teste si ces listes contiennent les sites qui ont signé les maillons respectivement. Comme nous avons $SITE_l(m_k) = SITE_l(m'_{k'})$ pour $0 \leq l \leq n$, nous avons aussi l'égalité entre les deux listes $[s_1, \ldots, s_n]$ et $[s'_1, \ldots, s'_n]$. Nous avons donc $o_0 = o'_0$.

– Soit maintenant $q < i$, tel que $o_{q-1} = o'_{q-1}$. Nous voulons démontrer que $o_q = o'_q$. Les maillons o_q et o'_q ont les formes $[x, hash(o_{q-1})]_{\mathsf{sk}(s)}$ et $[x', hash(o'_{q-1})]_{\mathsf{sk}(s')}$, parce que les termes m_k et $m'_{k'}$ sont acceptés par le programme P_{3bis}. Nous avons $o_{q-1} = o'_{q-1}$ par hypothèse d'induction et $s = s'$, car $SITE_q(m_k) = SITE_q(m'_{k'})$. Il faut encore démontrer que $x = x'$.

  Étant donné que les clefs $\mathsf{sk}(s)$ et $\mathsf{sk}(s')$ ne sont pas déductibles et qu'il n'y a pas de règle de déduction qui permet de signer sans déduire $\mathsf{sk}(s)$ et $\mathsf{sk}(s')$, le programme P_{2bis} doit être utilisé pour signer les maillons o_q et o'_q. Les programmes P_{1bis} et P_{3bis} ne peuvent pas être utilisés, parce que le premier renverrait des maillons qui ne sont pas acceptés par le programme P_{3bis}, sauf en tant que premiers maillons, et le deuxième ne renvoie que les constantes *accept* et *not_accept*. Il doit donc y avoir deux transitions $(Sec_1, H_1) \xrightarrow{\textbf{advance}(h_1)} (Sec'_1, H'_1)$ et $(Sec_2, H_2) \xrightarrow{\textbf{advance}(h_2)} (Sec'_2, H'_2)$ avec $h_1 = (p_1, m_1) \ldots (p'_1, m'_1) \in H_1$ et $h_2 = (p_2, m_2) \ldots (p'_2, m'_2) \in H_2$ où p'_1 et $p'_2 \in S_{P_{\text{2bis}}}$ et les termes m'_1 et m'_2 ont la forme des listes acceptées par le programme P_{2bis} en tant qu'entrées avec les maillons o_{q-1}, respectivement o'_{q-1}, en dernière position pour assurer que les termes de hachage dans les maillons o_q et o'_q sont corrects. Nous avons $H'_1 = H_1 \setminus \{h_1\} \cup \{h'_1\}$ et $H'_2 = H_2 \setminus \{h_2\} \cup \{h'_2\}$ avec $h'_1 = step_{\text{P1bis}}(h_1, H_1) = (p_1, m_1) \ldots (p'_1, m'_1)(p''_1, m''_1)$

et $h'_2 = step_{\text{P1bis}}(h_2, H_2) = (p_2, m_2)\dots(p'_2, m'_2)(p''_2, m''_2)$. Les maillons o_q et o'_q sont les derniers maillons des termes m''_1 et m''_2, renvoyés par la fonction $exec_{\text{P1bis}}$ avec, respectivement, p'_1 et m'_1 pour m''_1 et p'_2 et m'_2 pour m''_2 comme entrées.
Quant aux offres créées par le programme P_{2bis} et mises à la place de x et x', ils ont la forme $\mathit{off}(s, r)$ et $\mathit{off}(s', r)$, parce que les maillons o_q et o'_q sont signés par le programme P_{2bis} avec les clefs $\mathsf{sk}(s)$ et $\mathsf{sk}(s')$. Étant donné que $s = s'$, nous avons $x = x'$.
Nous concluons donc que $o_q = o'_q$. □

7.3.2 Deuxième modification : le protocole P1ter

Le protocole P1ter a été conçu pour obtenir des offres qui dépendent à la fois du site qui les propose et du site qui les demande. D'autre côté, elles ne dépendent pas d'une requête. Pour modéliser qu'une offre dépende de deux sites, nous utilisons une algèbre T'_x légèrement modifiée : nous remplaçons le symbole fonctionnel $\mathit{off}(s, r)$ dans T_x où $s \in S$ et $r \in R$ par le symbole $\mathit{off}(s, s')$ avec s et $s' \in S$. Cette modification revient donc implicitement à associer une seule requête à chaque site. Nous obtenons alors un raffinement du protocole P1bis.

Le protocole P1ter consiste en trois programmes, appelés P_{1ter}, P_{2ter} et P_{3ter}. Le premier programme est utilisé par le site qui commence la session. Le deuxième programme est exécuté par les sites qui ajoutent leurs offres. Finalement, le site qui a commencé la session se sert du troisième programme pour vérifier la liste des offres reçue.

Les programmes et conditions

Le programme P_{1ter} prend une paire $\langle u, [s_1, \dots, s_n]\rangle$ comme entrée où u représente le site qui initialise l'exécution du protocole et $[s_1, \dots, s_n]$ est la liste des sites dont il veut obtenir les offres. Le programme retourne alors le premier maillon. Le programme P_{1ter} est donné par

Fonction P_{1ter}

Entrées : $\langle u, [s_1, \dots, s_n]\rangle$, où $u, s_1, \dots, s_n \in S$ et $[s_1, \dots, s_n]$ n'est pas vide
Sorties : $[[u, [s_1, \dots, s_n]]_{\mathsf{sk}(u)}]$ ou *not_accept* si l'entrée n'a pas la forme attendue

Le programme P_{2ter} est censé être exécuté par les sites qui ajoutent leurs offres. Il prend comme entrée une liste d'offres et ajoute l'offre du site actuel. Pour trouver ce site, le programme compare la longueur de la liste des offres avec la longueur de l'itinéraire $[s_1, \dots, s_n]$ contenu dans le premier maillon. Lorsque le programme ajoute une offre, il suppose que c'est le site figurant au milieu du dernier maillon de la liste reçue qui demande l'offre. Si la liste des offres ne consiste qu'en un seul maillon, ce site doit être en première position. Le programme P_{2ter} est donné par

Fonction P_{2ter}

Entrées : $o_0 :: \dots :: o_k :: [\,]$ où o_0 a la forme $[u, [s_1, \dots, s_n]]_{\mathsf{sk}(u)}$ avec $n > k$
Sorties : m ou *not_accept* si l'entrée n'a pas la forme attendue
begin
  si o_k *s'unifie avec* $[x, y]_{\mathsf{sk}(x)}$ **alors**
    $u = x$
  si o_k *s'unifie avec* $[x, y, z]_{\mathsf{sk}(v)}$ **alors**
    $u = y$
  $m = o_0 :: \dots :: o_k :: [\mathit{off}(s_{k+1}, u), u, \mathit{hash}(o_k)]_{\mathsf{sk}(s_{k+1})} :: [\,]$
  retourner m
end

Le programme $P_{3\text{ter}}$ vérifie en premier lieu que les maillons d'une liste d'offres sont signés par les sites qui figurent dans la liste $[s_1, \ldots, s_n]$ du premier maillon. Deuxièmement, il vérifie que la relation de chaînage est valide. C'est le cas, si tous les maillons, sauf le premier, contiennent un terme de hachage, dans lequel apparaît le maillon précédant. Notons que le programme ne s'occupe pas des termes dans la positions des offres, ni, à l'exception du premier maillon, des termes qui figurent dans la position du site qui a initialisé la session. Le programme retourne *accept*, si la vérification finit avec succès, sinon *not_accept*. Le programme $P_{3\text{ter}}$ est donné par

Fonction $P_{3\text{ter}}$

Entrées : $o_0 :: o_1 :: \ldots :: o_n :: [\,]$
Sorties : $m \in \{accept, not\_accept\}$ ou not_accept si l'entrée n'a pas la forme attendue
begin
  $m = accept$
  si o_0 *ne s'unifie pas avec le terme* $[u, [s_1, \ldots, s_n]]_{\mathsf{sk}(u)}$
    ou *il existe un* o_i *qui n'est pas signé avec la clef* $\mathsf{sk}(s_i)$, $1 \leq i \leq n$, **alors**
    $m = not\_accept$
  pour $i = 1$ **à** n **faire**
    si o_i *ne s'unifie pas avec* $[x, y, hash(o_{i-1})]_{\mathsf{sk}(s_i)}$ **alors**
      $m = not\_accept$
  retourner m
end

Nous spécifions maintenant les conditions qui permettent de changer entres les exécutions des programmes $P_{1\text{ter}}$, $P_{2\text{ter}}$ et $P_{1\text{ter}}$. Soit m un terme de la forme $o_0 :: \ldots :: o_k :: [\,]$ et soit $[s_1, \ldots, s_n]$ l'itinéraire contenu dans le maillon o_0. En général, toutes les conditions $C_{i,j\text{ter}}$, $1 \leq i, j \leq 3$, sont fausses, sauf les suivantes :

- $C_{1,2\text{ter}}(m') = 1$ si et seulement si m' a la forme $\langle u, [s_1, \ldots, s_n]\rangle$,
- $C_{2,2\text{ter}}(m) = 1$ si et seulement si $k < n - 1$,
- $C_{2,3\text{ter}}(m) = 1$ si et seulement si $k = n - 1$.

Le protocole P1ter est donc spécifié par

$$\text{P1ter} = ((P_{1\text{ter}}, P_{2\text{ter}}, P_{3\text{ter}}), ((C_{1,1\text{ter}}, C_{1,2\text{ter}}, C_{1,3\text{ter}}), (C_{2,1\text{ter}}, C_{2,2\text{ter}}, C_{2,3\text{ter}}), (C_{3,1\text{ter}}, C_{3,2\text{ter}}, C_{3,3\text{ter}}))).$$

La preuve de l'intégrité forte en avant

Nous prouvons maintenant que le protocole P1ter satisfait la propriété de l'intégrité forte en avant de la définition 43. La preuve suit essentiellement la preuve donnée pour le protocole P1bis dans le paragraphe 7.3.1. Nous donnons d'abord les mêmes lemmes auxiliaires. Leur preuve se fait de manière analogue aux preuves des lemmes prouvés pour le protocole P1bis. Finalement, nous prouvons par récurrence la propriété souhaitée pour le protocole P1ter.

Lemme 21. *Soit* $tr \in \mathsf{Exec}(\text{P1ter})$ *une trace d'exécution. Elle contient un état global* (Sec, H) *avec* $h = (p_1, m_1) \ldots (p_k, m_k)(\perp, accept) \in H$ *si et seulement si elle contient un état global* (Sec', H'), *tel que* $h' = (p_1, m_1) \ldots (p_k, m_k) \in H'$ *et une transition* **advance**(h') *et nous avons* $p_k \in S_{P_{3\text{ter}}}$ *et* $P_{3\text{ter}}(m_k) = accept$.

Preuve. La preuve est analogue à la preuve du lemme 18. □

Lemme 22. *Soit* $tr \in \mathsf{Exec}(\text{P1ter})$ *une trace d'exécution. Si un site* $s \in S$ *n'apparaît pas dans une transition* **corrupt** *de tr, alors il n'existe pas d'état global* (Sec, H) *dans cette trace, tel que* $Msg(H) \cup Sec \vdash \mathsf{sk}(s)$.

Preuve. La preuve est analogue à la preuve du lemme 20. □

Théorème 5. *Le protocole* P1ter *satisfait la propriété de l'intégrité forte en avant de la définition 43.*

Preuve. Soit $tr \in \mathsf{Exec}(\text{P1ter})$ une trace, telle qu'elle contient les deux termes $m_k = [o_0, \ldots, o_n]$ et $m'_{k'} = [o'_0, \ldots, o'_n]$ de la définition 43 pour le protocole P1ter. La forme de ces deux termes est connue, parce qu'ils sont acceptés par le programme $P_{3\text{ter}}$, d'après le lemme 21. Soit $i \in \{0, \ldots, n\}$ tel que le site $SITE_i(m_k)$ est le premier site corrompu.

Dans le cas où $i = 0$, $j \in \{0, \ldots, n\}$ est toujours plus grand ou égal à i. Soit donc $i \in \{1, \ldots, n\}$. Les agents $SITE_l(m_k)$, $0 \leq l < i$, ne sont pas corrompus. D'après le lemme 22, il n'y aucun état global (Sec, H) dans tr, tel que $Msg(H) \cup Sec \vdash \mathsf{sk}(SITE_l(m_k))$.

Nous démontrons la propriété par récurrence :

- Soient les maillons o_0 et o'_0 de la forme $[s, [s_1, \ldots, s_n]]_{\mathsf{sk}(s)}$ et $[s', [s'_1, \ldots, s'_n]]_{\mathsf{sk}(s')}$. Les maillons ont cette forme, parce qu'ils sont acceptés par le programme $P_{3\text{ter}}$. Nous montrons que $o_0 = o'_0$. Nous avons déjà $SITE_0(m_k) = SITE_0(m'_{k'})$ et donc $s = s'$. Il reste à montrer que $[s_1, \ldots, s_n] = [s'_1, \ldots, s'_n]$. Étant donné que les termes m_k et $m'_{k'}$ sont acceptés par le programme $P_{3\text{ter}}$, les clefs, avec lesquelles sont respectivement signées les maillons, correspondent aux aux sites qui figurent dans les listes. Comme nous avons $SITE_l(m_k) = SITE_l(m'_{k'})$, $0 \leq l \leq n$, nous concluons que $s_l = s'_l$ et, par conséquent, que les deux listes sont égales. Nous avons donc $o_0 = o'_0$.
- Soit $q < i$, tel que $o_{q-1} = o'_{q-1}$. Nous prouvons que $o_q = o'_q$. Étant donné que m_k et $m'_{k'}$ sont des termes acceptés par le programme $P_{3\text{ter}}$, les maillons o_q et o_q sont respectivement de la forme $[x, y, hash(o_{q-1})]_{\mathsf{sk}(s)}$ et $[x', y', hash(o'_{q-1})]_{\mathsf{sk}(s')}$. Nous avons $o_{q-1} = o'_{q-1}$ par la hypothèse d'induction et $s = s'$, parce que $SITE_q(m_k) = SITE_q(m'_{k'})$. Il reste à montrer que $x = x'$ et $y = y'$.

  Étant donné que les clefs $\mathsf{sk}(s)$ et $\mathsf{sk}(s')$ ne sont pas déductibles et qu'il n'y a aucune règle de déduction qui permet de signer sans déduire la clef, le programme $P_{2\text{ter}}$ doit être utilisé pour signer les offres o_q et o'_q. Les programmes $P_{1\text{ter}}$ et $P_{3\text{ter}}$ ne peuvent pas être utilisés, car les termes renvoyés ne sont pas acceptés comme maillons par le programme $P_{3\text{ter}}$. Il doit donc y avoir deux transitions $(Sec_1, H_1) \xrightarrow{\mathbf{advance}(h_1)} (Sec'_1, H'_1)$ et $(Sec_2, H_2) \xrightarrow{\mathbf{advance}(h_2)} (Sec'_2, H'_2)$ avec $h_1 = (p_1, m_1) \ldots (p'_1, m'_1) \in H_1$ et $h_2 = (p_2, m_2) \ldots (p'_2, m'_2) \in H_2$ où p'_1 et $p'_2 \in S_{P_{2\text{ter}}}$ et les termes m'_1 et m'_2 ont la forme des listes acceptées par le programme $P_{2\text{ter}}$ comme entrées avec les maillons o_{q-1}, respectivement o'_{q-1}, en dernière position pour assurer que les termes de hachage dans les maillons o_q et o'_q sont corrects. Les maillons o_{q-1} et o'_{q-1} sont de la forme $[v, u, h]_{\mathsf{sk}(t)}$ et $[v', u', h']_{\mathsf{sk}(t')}$ ou de la forme $[u, h]_{\mathsf{sk}(t)}$ et $[u', h']_{\mathsf{sk}(t')}$, si $q - 1 = 0$. Nous avons $H'_1 = H_1 \setminus \{h_1\} \cup \{h'_1\}$ et $H'_2 = H_2 \setminus \{h_2\} \cup \{h'_2\}$ avec $h'_1 = step_{\text{P1ter}}(h_1, H_1) = (p_1, m_1) \ldots (p'_1, m'_1)(p''_1, m''_1)$ et $h'_2 = step_{\text{P1ter}}(h_2, H_2) = (p_2, m_2) \ldots (p'_2, m'_2)(p''_2, m''_2)$. Les maillons o_q et o'_q sont les derniers maillons des termes m''_1 et m''_2, renvoyés par la fonction $exec_{\text{P1ter}}$ avec, respectivement, p'_1 et m'_1 pour m''_1 et p'_2 et m'_2 pour m''_2 comme entrées.

  Les termes u et u' sont mis à la place de y et y' dans les maillons o_q et o'_q. Comme nous avons $o_{q-1} = o_{q-1}$ par hypothèse d'induction, nous avons $u = u'$ et, par conséquent, $y = y'$. Quant aux offres créées par le programme $P_{2\text{ter}}$ et mises à la place de x et x', ils ont la forme $off(s, u)$ et $off(s', u')$. Nous avons déjà montré que $s = s'$ et $u = u'$. Les deux offres sont donc égales et nous avons $x = x'$.

  Nous concluons donc que $o_q = o'_q$. □

7.3.3 Troisième modification : le protocole P1quater

Nous gardons l'algèbre modifiée T'_x que nous venons de définir dans le paragraphe 7.3.2, c'est-à-dire que le symbole fonctionnel *off* dépend de deux sites au lieu d'un site et d'une requête. La différence principale entre les protocoles P1ter et P1quater est le fait que la constante qui représente le site initialisant la session n'est plus répétée dans tous les maillons de la liste des offres. Elle se trouve seulement dans le premier. Le protocole P1quater consiste en trois programmes que nous appelons $P_{1\text{quater}}$, $P_{2\text{quater}}$ et $P_{3\text{quater}}$. Ces programmes remplissent les mêmes fonctions que ceux décrites lors de la présentation du protocole P1ter du paragraphe 7.3.2.

Le fait que nous ne répétons pas dans chaque maillon le site qui commence la session rend la relation de chaînage insignifiante en ce qui concerne la propriété de l'intégrité forte en avant pour le protocole P1quater. En effet, nous ne pouvons plus garantir cette propriété et montrer une attaque. Nous exhibons d'une manière informelle un tel scénario. Bien que le protocole P1quater ne satisfasse pas la propriété souhaitée, il nous permet néanmoins de mieux comprendre le fonctionnement des protocoles P1bis et P1ter et de justifier les décisions de design que nous avons prises à l'égard de ces derniers.

Les programmes et conditions

Le programme $P_{1\text{quater}}$ ne diffère pas du programme $P_{1\text{ter}}$. Il est donné par

Fonction $P_{1\text{quater}}$

Entrées : $\langle u, [s_1, \ldots, s_n] \rangle$, où $u, s_1, \ldots, s_n \in S$ et $[s_1, \ldots, s_n]$ n'est pas vide
Sorties : $[[u, [s_1, \ldots, s_n]]_{\mathsf{sk}(u)}]$ ou *not_accept* si l'entrée n'a pas la forme attendue

Le programme $P_{2\text{quater}}$ diffère du programme $P_{2\text{ter}}$ par le fait que les maillons ajoutés ne comprennent plus la constante représentant le site qui a initialisé la session. Par conséquent, il prend cette information du premier maillon o_0 de la liste. Le programme $P_{2\text{quater}}$ est donné par

Fonction $P_{2\text{quater}}$

Entrées : $o_0 :: \ldots :: o_k :: [\,]$ où o_0 a la forme $[u, [s_1, \ldots, s_n]]_{\mathsf{sk}(u)}$ avec $n > k$
Sorties : m ou *not_accept* si l'entrée n'a pas la forme attendue
begin
  $m = o_0 :: \ldots :: o_k :: [off(s_{k+1}, u), hash(o_k)]_{\mathsf{sk}(s_{k+1})} :: [\,]$
  retourner m
end

Le programme $P_{3\text{quater}}$ vérifie que les maillons sont respectivement signés par les sites qui figurent dans la liste $[s_1, \ldots, s_n]$ contenu dans le maillon o_0. En outre, il vérifie que la relation de chaînage est valide, c'est-à-dire que tous les termes de hachage contiennent le maillon qui précède directement dans la liste. Notons que le programme ne s'occupe pas des termes qui figurent dans dans la position des offres de chaque maillon. Le programme renvoie *accept*, si la vérification se termine avec succès, sinon *not_accept*. Le programme $P_{3\text{quater}}$ est donné par

Fonction $P_{3\text{quater}}$

Entrées : $o_0 :: o_1 :: \ldots :: o_n :: [\,]$
Sorties : $m \in \{accept, not\_accept\}$ ou not_accept si l'entrée n'a pas la forme attendue
begin
  $m = accept$
  si o_0 *ne s'unifie pas avec le terme* $[u, [s_1, \ldots, s_n]]_{\mathsf{sk}(u)}$
    ou *il existe un* o_i *qui n'est pas signé avec la clef* $\mathsf{sk}(s_i)$, $1 \leq i \leq n$, **alors**
    $m = not\_accept$
  pour $i = 1$ **à** n **faire**
    si o_i *ne s'unifie pas avec* $[x, hash(o_{i-1})]_{\mathsf{sk}(y)}$ **alors**
      $m = not\_accept$
  retourner m
end

Nous spécifions maintenant les conditions qui définissent les changements entres les exécutions des programmes $P_{1\text{quater}}$, $P_{2\text{quater}}$ et $P_{1\text{quater}}$. Soit m un terme de la forme $o_0 :: \ldots :: o_k :: [\,]$ où le maillon o_0 contient la liste $[s_1, \ldots, s_n]$ des sites à visiter. En général, toutes les conditions $C_{i,j\text{quater}}$, $1 \leq i, j \leq 3$, sont fausses, sauf les suivantes :

- $C_{1,2\text{quater}}(m') = 1$ si et seulement si m' a la forme $\langle u, [s_1, \ldots, s_n]\rangle$
- $C_{2,2\text{quater}}(m) = 1$ si et seulement si $k < n - 1$.
- $C_{2,3\text{quater}}(m) = 1$ si et seulement si $k = n - 1$.

Le protocole P1quater est spécifié par

$$\text{P1quater} = ((P_{1\text{quater}}, P_{2\text{quater}}, P_{3\text{quater}}),$$
$$((C_{1,1\text{quater}}, C_{1,2\text{quater}}, C_{1,3\text{quater}}), (C_{2,1\text{quater}}, C_{2,2\text{quater}}, C_{2,3\text{quater}}), (C_{3,1\text{quater}}, C_{3,2\text{quater}}, C_{3,3\text{quater}}))).$$

Une attaque contre le protocole P1quater

Au lieu de donner une trace d'exécution, nous donnons une description informelle d'une attaque qui existe pour le protocole P1quater contre la propriété de l'intégrité forte en avant donnée dans la définition 43. Soit $h_1 = \ldots (p_1, m_1)$ avec $p_1 \in S_{P_{2\text{quater}}}$ et $m_1 = o_0 :: \ldots o_k :: [\,]$ un historique d'exécution. Soit le maillon o_0 de la forme $[u, [s_1, \ldots, s_n]]_{\mathsf{sk}(u)}$, tel que $n > k + 1$ et aucun des sites s_i, $1 \leq i \leq k$, n'est corrompu, et soient les maillons o_i signés par les clefs $\mathsf{sk}(s_i)$. Un intrus peut commencer une nouvelle session pour un site $u' \neq u$ non corrompu. Au cours de la session, le site u' veut obtenir les offres des sites $s_1, \ldots, s_n$. Il obtient alors un premier maillon o'_0 de la forme $[u', [s_1, \ldots, s_n]]_{\mathsf{sk}(u')}$. À l'aide de ce maillon, il peut construire une liste d'offres $m'_1 = o'_0 :: o_1 :: \ldots :: o_k :: [\,]$, pour laquelle il crée un historique d'exécution $h'_1 = (p'_1, m'_1)$ avec $p'_1 \in S_{P_{2\text{quater}}}$. Lors des transitions **advance**(h_1) et **advance**(h'_1), le programme $P_{2\text{quater}}$ ajoute le maillon o_{k+1} à la liste m_1, respectivement o'_{k+1} à la liste m'_1. L'adversaire obtient donc deux historiques d'exécution $h_2 = \ldots (p_1, m_1)(p_2, m_2)$ et $h'_2 = (p'_1, m'_1)(p'_2, m'_2)$ avec $m_2 = o_0 :: \ldots o_k :: o_{k+1} :: [\,]$ et $m'_2 = o'_0 :: \ldots o_k :: o'_{k+1} :: [\,]$ où les maillons o_{k+1} et o'_{k+1} sont de la forme $[off(u, s_k), hash(o_k)]_{\mathsf{sk}(s_k)}$ et $[off(u', s_k), hash(o_k)]_{\mathsf{sk}(s_k)}$. Ensuite, l'intrus remplace le maillon o'_0 de la liste m'_2 par le maillon o_0. Il obtient donc une liste $m''_2 = o_0 :: \ldots o_k :: o'_{k+1} :: [\,]$ pour lequel il peut créer un historique d'exécution $h''_1 = (p, m''_2)$ avec $p \in S_{P_{2\text{quater}}}$.

Soient maintenant $m_3 = o_0 :: \ldots :: o_n :: [\,]$ et $m'_3 = o_0 :: o_1 :: \ldots :: o_k :: o'_{k+1} :: \ldots :: o'_n :: [\,]$ les listes obtenues en finissant l'exécution du protocole pour les historiques d'exécution h_1 et h''_1. Ces listes sont acceptées par le programme $P_{3\text{quater}}$, parce que les maillons sont respectivement signés par les clefs des sites qui figurent dans la liste $[s_1, \ldots, s_n]$ du maillons o_0 et la relation de

chaînage est valide. Nous avons $length(m_3) = length(m'_3)$ et $SITE_l(m_3) = SITE_l(m'_3)$, $0 \leq l \leq length(m_3)$. Étant donné que les sites $s_1, \ldots, s_n$ ne sont pas corrompus, il ne doit pas exister un $j \in \{0, \ldots, length(m_3)\}$, tel que $OFF_j(m_3) \neq OFF_j(m'_3)$, mais avec $j = k+1$ nous avons trouvé des offres telles que $OFF_j(m_3) \neq OFF_j(m'_3)$, parce que nous avons choisi dans la deuxième session un site $u' \neq u$. Nous concluons que le protocole P1quater ne satisfait pas la propriété de l'intégrité forte en avant.

7.4 Conclusion

Nous avons d'abord modélisé le protocole original P1 [KAG98] à l'aide du modèle proposé dans le chapitre 6. La modélisation nous a permis de retrouver l'attaque contre la propriété de l'intégrité forte en avant décrite dans [Rot01]. Le modèle est donc expressif en ce qui concerne la recherche des attaques contre cette propriété.

Les auteurs du protocole P1 ont affirmé que le protocole P1 satisfait la propriété de l'intégrité forte en avant. Cette affirmation est basée sur la croyance qu'une relation de chaînage valide implique qu'un intrus ne peut pas changer les maillons ajouté par des sites non corrompus. Cette croyance s'est avérée fautive avec la découverte de l'attaque mentionnée. Néanmoins, il reste une question intéressante : comment pouvons-nous nous servir d'une relation de chaînage pour garantir des propriétés d'un protocole, si possible efficacement ? En proposant trois protocoles à l'instar du protocole P1, nous avons abordé cette question d'une manière pratique. Nous avons montré que notre modèle formel peut être utilisé à cet égard avec profit.

Nous voulons d'abord analyser les raisons, pour lesquelles le protocole P1ter satisfait la propriété de l'intégrité en avant, tandis qu'il existe une attaque contre le protocole P1quater. La seule différence est le fait que nous répétons dans chaque maillon du premier protocole la constante u qui représente le site qui a commencé la session, ce que nous ne faisons pas dans les maillons utilisés pour le deuxième protocole. Le fait de vérifier la relation de chaînage dans le protocole P1ter revient alors à la vérification que tous les maillons contiennent le site u et les offres proposées pour ce site, sans que nous devions explicitement vérifier les offres et le site. Utilisant que le premier maillon est aussi signé par le site u, nous parvenons à prouver la propriété souhaitée. Le protocole P1quater ne nous permet pas cette conclusion. Nous avons vu que nous pouvons temporairement changer le site u dans le premier maillon, sans que cela affecte la validité de la relation de chaînage.

Le protocole P1bis ne permet qu'une seule requête. Elle a été codée dans les programmes. Nous pouvons nous demander, si ce protocole satisfait la propriété désirée dans le cas où nous permettons plusieurs requêtes. Nous pourrions par exemple modifier le protocole P1ter en remplaçant dans chaque maillon le site u par une requête r. Cependant, cette modification ne garantit pas l'intégrité forte en avant comme nous l'avons définie, parce que nous pouvons construire deux listes dans deux exécutions normales, telles que les sites visités sont les mêmes dans le même ordre. Pourtant, la requête contenue dans les maillons de la première liste peut différer de la requête contenue dans les maillons de la deuxième. Le protocole P1ter satisfait l'intégrité forte en avant, parce que nous avons fixé pour chaque site une seul requête qui est, en effet, identique avec son nom. Il faudrait donc trouver une autre définition de la propriété de l'intégrité forte en avant pour permettre à un site d'utiliser plusieurs requêtes.

Nous pouvons aussi poser la question du rapport entre le coût de la vérification d'une structure de données et les garanties que nous pouvons donner pour un protocole. Si nous avions par exemple pris soin que le programme $P_{3\text{quater}}$ vérifie que les offres *off* dépendent du site qui a signé le premier maillon, nous aurions pu espérer que le protocole P1quater satisfait la propriété

souhaitée. Cela reviendrait en effet à une vérification explicite de ce que nous avons vérifié implicitement dans le protocole P1ter en testant si la relation de chaînage est valide. Le rapport qui existe entre le coût de vérifier la structure des messages et les propriétés d'un protocole n'est pas clair.

Conclusion et perspectives

1 Première partie

Conclusion

Nous avons montré dans la première partie que les propriétés d'une certaine classe peuvent être transférées d'un modèle symbolique peu riche en fonctionnalités vers des modèles plus riches sous la condition que les symboles fonctionnels manquants soient codés d'une manière bien définie. Ces codages sont notamment le remplacement des primitives probabilistes par leurs pendants déterministes, des signatures par le chiffrement asymétrique à clef privée et des fonctions de hachage par le chiffrement asymétrique à clef publique, sous l'hypothèse que les agents à qui les clefs publiques appartiennent ne participent pas dans l'exécution du protocole. Dans ce cas, nul agent ne peut déduire le contenu d'un tel terme, ce qui revient à une valeur de hachage. Nous avons prouvé des théorèmes de correction pour ces trois codages.

La certitude que les codages sont corrects permet aux vérificateurs des protocoles cryptographiques d'analyser des protocoles dans un outil automatique, même si cet outil n'offre pas toutes les fonctionnalités dont le protocole a besoin, tout en obtenant des garanties aussi fortes que pour une vérification dans un modèle qui contient toutes les fonctionnalités nécessaires. Nous avons utilisé dans un deuxième temps cette connaissance pour étendre le domaine d'application de la plate-forme AVISPA, outil de vérification automatique. Le nouveau module qui est intégré désormais dans cet outil est basé principalement sur le théorème de correction qui nous permet de remplacer les primitives probabilistes par leurs pendants déterministes, ainsi qu'un théorème de la littérature [CKKW06] qui permet le transfert de propriétés d'un modèle symbolique contenant des primitives probabilistes vers un modèle calculatoire. Une vérification dans un modèle à primitives déterministes nous mène alors automatiquement à une vérification dans un modèle symbolique à primitives probabilistes, puis à une vérification pour un modèle calculatoire. Pour tester l'efficacité de notre méthode, nous avons analysé la bibliothèque de protocoles d'AVISPA dont nous avons pu vérifier 9 parmi les 84 protocoles pour le modèle calculatoire.

Perspectives

Un problème majeur pour transférer des résultats d'une analyse symbolique vers un modèle calculatoire était le fait que nous ne pouvions pas prendre en compte la signature, parce qu'elle n'est pas prévue dans l'article [CKKW06]. Or, un grand nombre de protocoles en a besoin. Nous avons vu qu'une méthode existe pour vérifier aussi ces protocoles. Cette méthode consiste en l'application du théorème de correction qui nous permet de transférer des propriété prouvées pour un modèle où la signature est codée par un chiffrement asymétrique à clef privée vers le modèle avec signature. Une automatisation de cette méthode se heurte notamment à l'absence d'une distinction claire entre les clefs publiques et privées dans la plate-forme AVISPA. Celle-ci

ne permet que la définition des clefs et leurs inverses, sans pour autant prescrire à un vérificateur lesquelles il doit utiliser comme clefs publiques et lesquelles comme privées. À l'extrême, on pourrait même mélanger les interprétations : on pourrait, dans un protocole, utiliser quelques clefs inverses comme clefs publiques et d'autres comme clefs privées. Pour cette raison, les codages des signatures ne peuvent pas être syntaxiquement reconnus en tant que tels. Deux possibilités se présentent alors pour remédier à ce problème et automatiser la vérification : premièrement, on pourrait introduire des moyens syntaxiques qui permettent la distinction entre clefs publiques et privées. Deuxièmement, on pourrait tout de suite munir la plate-forme AVISPA d'un symbole fonctionnel pour exprimer la signature. À notre avis, la deuxième possibilité est préférable, car nous la jugeons plus nette. Il reste néanmoins possible pour un vérificateur rigoureux de bien garder en tête la signification des clefs et de leurs inverses et d'appliquer à la main le théorème qui permet le transfert du modèle sans signature vers le modèle avec signature, puis vers le modèle avec primitives probabilistes pour finalement obtenir des garanties pour le modèle calculatoire.

Un résultat souhaitable serait aussi une combinaison des résultats de [CW05] et de [CKKW06], donc d'un théorème qui permet le transfert vers un modèle calculatoire pour des protocoles avec chiffrement asymétrique et signature et d'un théorème qui permet le transfert pour des protocoles avec chiffrement asymétrique et fonctions de hachage, pour obtenir un théorème qui justifie le transfert de propriétés entre un modèle symbolique qui contient à la fois le chiffrement asymétrique, la signature et les fonctions de hachage vers un modèle calculatoire. Un tel résultat en combinaison avec un adaptation de la syntaxe d'AVISPA qui rend possible la reconnaissance des signatures permettrait la vérification automatique de nombreux protocoles pour un modèle calculatoire.

2 Deuxième partie

Conclusion

Nous avons commencé la deuxième partie de cette thèse avec une classification des types de récursivité qui peuvent être trouvés dans des protocoles récursifs. Premièrement, on peut y compter les protocoles qui ont besoin des structures de données de taille non bornée. Deuxièmement, le nombre de participants peut être non borné et, finalement, le nombre de messages échangés dans une exécution peut être non borné. Toutes ces incertitudes nécessitent une description récursive, soit dans les actions effectuées par les participants, soit dans la description globale d'un protocole. Nous avons cité des exemples pour chacun de ces types de récursivité. Ce premier chapitre sur les protocoles récursifs se termine avec un bref aperçu sur la littérature existante.

Nous nous avons ensuite proposé l'étude formelle d'un protocole particulier présenté dans la littérature [KAG98]. Il s'agit d'un protocole de commerce électronique qui permet à un participant d'envoyer un agent collecter des offres sur le réseau pour un certain produit. Pour étudier ce protocole, nous avons donc dans un premier temps défini un modèle symbolique. Malgré l'étude de cas que nous avions en tête, nous pensons que ce modèle est assez flexible pour pouvoir facilement être adapté pour servir à l'étude d'autres protocoles récursifs. Nous pensons notamment qu'il existe des exemples de protocoles pour chacun des types de récursivité mentionnés que nous pouvons modéliser dans notre formalisme.

Ensuite, nous avons retrouvé à l'aide de notre modèle symbolique une attaque connue contre une propriété du protocole en question. Cela montre bien que notre modèle est utile. À la suite de cette analyse, nous nous sommes rendus compte que la propriété examinée est en effet très particulière, parce que certains états de l'exécution qui semblent intuitivement être des attaques

contre cette propriété ne le sont pas. C'est notamment le cas d'une exécution où la liste des offres est tronquée et reconstruite. Nous avons donc modifié cette propriété pour exclure ces particularités. Ensuite, nous avons proposé deux modifications du protocole original avec comme but d'obtenir un protocole qui satisfasse cette nouvelle propriété. La première variante nous permet de collecter des offres qui dépendent du site qui les propose, tandis que la deuxième version permet de les faire aussi dépendre du site qui les demande. À l'aide de notre modèle symbolique nous avons prouvé que les variantes proposées satisfont en effet la propriété en question.

Perspectives

A notre avis, un futur travail sur les protocoles récursifs doit se concentrer sur la recherche des classes décidables. Étant donné qu'à ce jour, peu de résultats de décidabilité sont connus dans ce domaine de recherche, cette direction nous semble être prometteuse, d'autant plus que la récursivité est très répandue dans des protocoles utilisés en réalité. On peut donc espérer tirer un avantage pratique des résultats théoriques.

Bibliographie

[Aba00] M. Abadi. Security protocols and their properties. In *Foundations of Secure Computation*. IOS Press, 2000. 20th Int. Summer School, Marktoberdorf, Germany.

[ABB+05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In *Proc. of the 17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[AC04] A. Armando and L. Compagna. SATMC : A SAT-based model checker for security protocols. In *Logics in Artificial Intelligence*, volume 3229 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[AES01] Advanced encryption standard. National Bureau of Standards, Processing Standards Publication No. 197, 2001.

[AR00] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proc. of the 2000 International Conference IFIP on Theoretical Computer Science (TCS00*, volume 1872. Springer-Verlag, 2000.

[AST00] G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4), 2000.

[BDPR98] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Proc. of the 18th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO98)*. Springer-Verlag, 1998.

[BG85] M. Blum and S. Goldwasser. An efficient probabilistic public key encryption scheme which hides all partial information. In *Proc. of CRYPTO 84 on Advances in cryptology*. Springer-Verlag, 1985.

[BHKO04] Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay technique to automatically verify security protocols. In *Proc. of the 2004 International Workshop on Automated Verification of Infinite-State Systems (AVIS04)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2004.

[BJ03] M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. of the 20th International Symposium on Theoretical Aspects of Computer Science, STACS '03*, volume 2607 of *Lecture Notes in Computer Science*, 2003.

[Bla01] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press, 2001.

[BM82] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo random bits. In *Proc. of the 23rd Symposium on Foundations of Computer Science (FOCS82)*. IEEE Computer Society Press, 1982.

[BMV05] D. Basin, S. Mödersheim, and L. Viganò. OFMC : A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3), 2005.

[BO86] R. V. Book and F. Otto. The verifiability of two-party protocols. In *Advances in Cryptology : Proc. of a Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT85)*, Lecture Notes in Computer Science. Springer-Verlag, 1986.

[BO97] J. A. Bull and D. J. Otway. The authentification protocol. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defence Research Agency, Malvern, UK, 1997.

[BP05] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *Proc. of the 2005 IEEE Symposium on Security and Privacy*, 00, 2005.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical : a paradigm for designing efficient protocols. In *Proc. of the 1st ACM conference on computer and communications security (CCS93)*. ACM Press, 1993.

[BS97] J. Bryans and S. Schneider. CSP, PVS, and a recursive authentication protocol. In *Proc. of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[CD99] K. J. Compton and S. Dexter. Proof techniques for cryptographic protocols. In *Proc. of the 26th International Colloquium on Automata, Languages and Programming (ICALP99)*. Springer-Verlag, 1999.

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on : `http ://www.grappa.univ-lille3.fr/tata`, 1997.

[CDL+99] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proc. of the 12th IEEE workshop on Computer Security Foundations (CSFW99)*. IEEE Computer Society Press, 1999.

[CDL06] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 1(1), 2006.

[CGH04] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4), 2004.

[CHW06a] V. Cortier, H. Hördegen, and B. Warinschi. Explicit randomness is not necessary when modeling probabilistic encryption. In *Workshop on Information and Computer Security (ICS 2006)*, Electronic Notes in Theoretical Computer Science. Elsevier, 2006.

[CHW06b] V. Cortier, H. Hördegen, and B. Warinschi. Explicit randomness is not necessary when modeling probabilistic encryption. Research Report 5928, INRIA, 2006.

[CKKW06] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Proc. of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science*

(FSTTCS'06), volume 4337 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.

[CLC03] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proc. 14th International Conference on Rewriting Techniques and Applications (RTA03)*, volume 2706 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[CM96] A. Chavez and P. Maes. Kasbah : An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*. Practical Application Company, 1996.

[CMP05] I. Cervesato, C. Meadows, and D. Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In *Proc. of the 18th IEEE workshop on Computer Security Foundations (CSFW05)*. IEEE Computer Society Press, 2005.

[CV02] Y. Chevalier and L. Vigneron. Automated unbounded verification of security protocols. In *Proc. of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[CW05] V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *Proc. the 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[DEK82] D. Dolev, S. Even, and R. M. Karp. On the security of ping-pong protocols. In *Proc of CRYPTO 82*. Plenum Press, 1982.

[DES77] Data encryption standard. National Bureau of Standards, Processing Standards Publication No. 46, 1977.

[DEW97] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the world-wide web. In *Proc. of the 1st International Conference on Autonomous Agents (Agents'97)*. ACM Press, 1997.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6), 1976.

[DK97] Z. Dang and R. Kemmerer. Using the ASTRAL model checker for cryptographic protocol analysis, 1997.

[DM00] G. Denker and J. Millen. CAPSL integrated protocol environment. In *Proc. of the DARPA Information Survivability Conference (DISCEX00)*. IEEE Computer Society Press, 2000.

[DVOW92] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2(2), 1992.

[DY81] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. of the 22nd IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 1981.

[EG85] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proc. of CRYPTO 84 on Advances in cryptology*. Springer-Verlag, 1985.

[ElG85] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. In *Proc. of CRYPTO 84 on Advances in Cryptology*. Springer-Verlag, 1985.

[FS87] A. Fiat and A. Shamir. How to prove yourself : practical solutions to identification and signature problems. In *Proc. on Advances in cryptology—CRYPTO '86*. Springer-Verlag, 1987.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28, 1984.

[Gol93] O. Goldreich. A uniform-complexity treatment of encryption and zero-knowledge. *Journal of Cryptology*, 6(1), 1993.

[HC98] D. Harkins and D. Carrel. *The Internet Key Exchange (IKE)*. The Internet Society, 1998. RFC 2409.

[Her05] J. Herzog. A computational interpretation of Dolev-Yao adversaries. *Theoretical Computer Science*, 340(1), 2005.

[JRV00] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Logic for Programming and Automated Reasoning (LPAR'00)*, volume 1955 of *Lecture Notes in Computer Science*, 2000.

[KAG98] G. Karjoth, N. Asokan, and C. Gülcü. Protecting the computation results of free-roaming agents. In *Proc. the 2nd International Workshop on Mobile Agents (MA'98)*, volume 1477 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[KT07] R. Küsters and T. Truderung. On the automatic analysis of recursive security protocols with XOR. In *Proc. of the 24th Symposium on Theoretical Aspects of Computer Science (STACS 2007)*, volume 4393 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[Küs05] R. Küsters. On the decidability of cryptographic protocols with open-ended data structures, 2005.

[KW03] R. Küsters and T. Wilke. Automata-based Analysis of Recursive Cryptographic Protocols. Technical Report 0311, Institut für Informatik und Praktische Mathematik, CAU Kiel, Germany, 2003.

[KW04] R. Küsters and T. Wilke. Automata-based analysis of recursive cryptographic protocols. In *21st Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, volume 2996 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[Kür06] K. O. Kürtz. Automatic analysis of recursive cryptographic protocols. Master's thesis, Christian-Albrechts-Universität zu Kiel, 2006.

[Low95] G. Lowe. An attack on the needham-schroeder public-key authentication protocol. *Information Processing Letters*, 56(3), 1995.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. of Tools and algoritms for the construction and analysis of systems (TACAS'96)*. Springer-Verlag, 1996.

[Low97a] Lowe. Casper : A compiler for the analysis of security protocols. In *Proc. of The 10th Computer Security Foundations Workshop (PCSFW97)*. IEEE Computer Society Press, 1997.

[Low97b] G. Lowe. Casper : A compiler for the analysis of security protocols. In *Proc. of the 10th Computer Security Foundations Workshop (CSFW'97)*. IEEE Computer Society Press, 1997.

[Low97c] G. Lowe. A hierarchy of authentication specifications. In *Proc. of the 10th Computer Security Foundations Workshop (CSFW'97)*. IEEE Computer Society Press, 1997.

[Mea96] C. Meadows. The NRL protocol analyzer : An overview. *Journal of Logic Programming*, 26(2), 1996.

[Mea00] C. Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In *First Workshop on Issues in the Theory of Security WITS'00*, 2000.

[Mea01] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. *Lecture Notes in Computer Science*, 2052, 2001.

[MS01] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, 2001.

[MSST98] D. Maughan, M. Schneider, M. Schertler, and J. Turner. *Security Association and Key Management Protocol (ISAKMP)*. The Internet Society, 1998. RFC 2408.

[MvOV96] A. J. Menezes, P. C. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[MW04] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of cryptography conference - Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer-Verlag, 2004.

[NS78] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commununication of the ACM*, 21(12), 1978.

[NY90] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proc. of the 22nd annual ACM symposium on Theory of computing (STOC90)*. ACM Press, 1990.

[Pau97a] L. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. of the 10th Computer Security Foundations Workshop (CSFW'97)*. IEEE Computer Society Press, 1997.

[Pau97b] L. C. Paulson. Proving properties of security protocols by induction. In *10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[Pau98] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6, 1998.

[PCTS00] A. Perrig, R. Canetti, J. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proc. of the 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2000.

[PQ01] O. Pereira and J.-J. Quisquater. Security analysis of the cliques protocols suites : First results. In *Proc. of the IFIP TC11 16th Annual Working Conference on Information Security*, volume 193. Kluwer, B.V., 2001.

[PS00] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signature. *Journal of Cryptology*, 13(3), 2000.

[Rot01] V. Roth. On the robustness of some cryptographic protocols for mobile agent protection. In *Mobile Agents*, volume 2240 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[RS98] P. Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol. a cautionary tale. *Information Processing Letters*, 65(1), 1998.

[RSA78a] R. L. Rivest, A. Shamir, and L. M Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2), 1978.

[RSA78b] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communication of the ACM*, 21(2), 1978.

[RT01] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW01)*. IEEE Computer Society Press, 2001.

[Son99] D. X. Song. Athena : A new efficient automatic checker for security protocol analysis. In *Proc. of the 12th Computer Security Foundations Workshop (CSFW'99)*. IEEE Computer Society Press, 1999.

[STW96] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *ACM Conference on Computer and Communications Security*. ACM Press, 1996.

[STW98] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES : A new approach to group key agreement. In *Proc. of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*. IEEE Computer Society Press, 1998.

[THG99] J. Thayer, J. Herzog, and J. Guttman. Strand spaces : Proving security protocols correct. *IEEE Journal of Computer Security*, 7, 1999.

[Tru05] T. Truderung. Selecting theories and recursive protocols. In *CONCUR 2005 — Concurrency Theory*, Lecture Notes in Computer Science. Springer-Verlag, 2005.

[WC04] Y. Wang and B.-T. Chu. sSCADA : Securing SCADA Infrastructure Communications. Cryptology ePrint Archive, Report 2004/265, 2004.

[Yao82] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. of the 23rd Annual Symposium on the Foundations of Computer Science (FOCS82)*. IEEE Computer Society Press, 1982.

[YCCC+04] Y. Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drieslma, J. Mantovani, S. Mödersheim, and L. Vigneron. A high level protocol specification language for industrial security-sensitive protocols (SAPS 2004). In *Proc. of the Workshop on Specification and Automated Processing of Security Requirements*. Austrian Computer Society, 2004.

[Yee97] B. S. Yee. A sanctuary for mobile agents. Technical Report CS97-537, UC San Diego, Department of Computer Science and Engineering, 1997.