Vérification formelle de protocoles basés sur de courtes chaines authentifiées

Ludovic Robin

▶ **To cite this version:**

Ludovic Robin. Vérification formelle de protocoles basés sur de courtes chaines authentifiées. Cryptographie et sécurité [cs.CR]. Université de Lorraine, 2018. Français. NNT : 2018LORR0019 . tel-01767989

HAL Id: tel-01767989
https://theses.hal.science/tel-01767989

Submitted on 16 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Vérification formelle de protocoles basés sur de courtes chaines authentifiées

THÈSE

présentée et soutenue publiquement le 15 février 2018

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Ludovic Robin

Composition du jury

Rapporteurs :	Yannick Chevalier	Maitre de conférence, Université de Toulouse III
	Tayssir Touili	Directrice de recherche, CNRS, Paris
Examinateurs :	Stéphanie Delaune	Directrice de recherche, CNRS, Rennes
	Steve Kremer	Directeur de recherche, Inria, Nancy
	Pierre-Yves Strub	Maitre de conférence, École polytechnique
	Emmanuel Thomé	Directeur de recherche, Inria, Nancy

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

Mis en page avec la classe thesul.

Résumé

Les protocoles de sécurité modernes peuvent impliquer un participant humain de façon à ce qu'il compare ou copie de courtes chaines de caractères faisant le pont entre différents appareils. C'est par exemple le cas des protocoles basés sur une authentification à facteur multiples comme les protocoles *Google 2 factor* ou *3D-Secure*. Cependant, de telles chaines de caractères peuvent être sujettes à des attaques par force brute. Dans cette thèse nous proposons un modèle symbolique qui inclut les capacités de l'attaquant à deviner des secrets faibles et à produire des collisions avec des fonctions de hachage dont de l'application résulte une courte chaine de caractères. Nous proposons une nouvelle procédure de décision pour analyser un protocole (restreint à un nombre borné de sessions) qui se base sur de courtes chaines de caractères. Cette procédure a été intégré dans l'outil AKISS et testé sur les protocoles du standard ISO/IEC 9798-6 :2010.

Mots-clés: Méthodes formelles, Protocoles, Sécurité, Vérification

Abstract

Modern security protocols may involve humans in order to compare or copy short strings between different devices. Multi-factor authentication protocols, such as *Google 2-factor* or *3D-Secure* are typical examples of such protocols. However, such short strings may be subject to brute force attacks. In this thesis we propose a symbolic model which includes attacker capabilities for both guessing short strings, and producing collisions when short strings result from an application of weak hash functions. We propose a new decision procedure for analyzing (a bounded number of sessions of) protocols that rely on short strings. The procedure has been integrated in the AKISS tool and tested protocols from the ISO/IEC 9798-6 :2010 standard.

Keywords: Formal methods, Protocols, Security, Verification

Remerciements

«Chaque découverte, chaque progrès, chaque augmentation de la richesse de l'humanité a son origine dans l'ensemble du travail manuel et cérébral du passé et du présent. Alors, de quel droit quiconque pourrait-il s'approprier la moindre parcelle de cet immense tout, et dire : ceci est à moi, non à vous ?» La conquête du pain, Pierre Kropotkine

J'aimerai en premier lieu remercier tous les membres du jury. Merci à Yannick Chevalier et Tayssir Touili pour avoir accepté d'être rapporteurs et pour la relecture attentive de mon manuscrit. Je remercie Pierre-Yves Strub d'avoir accepté de participer à mon jury. Je remercie Emmanuel Thomé pour avoir été un référent sans faille et m'avoir conseillé pour l'après thèse. Enfin je remercie Steve Kremer de m'avoir fait découvrir ce domaine de recherche passionnant en stage, puis en thèse, en partageant les durs efforts de mon encadrement avec Stéphanie Delaune que je remercie tout autant. Merci à vous de m'avoir accordé du temps, des conseils, de m'avoir formé au monde de la recherche. Cette expérience enrichissante tant sur le plan scientifique que sur le plan humain, n'aurait pas été possible sans votre patience et votre engagement, même dans ces périodes où l'on aimerait que son thésard soit plus efficient.

Je remercie tous mes collègues de l'équipe Pesto. Merci à tous ceux qui ont partagé mon bureau et avec qui nous avons passé de très bons moments : comme ces discussions sur les meilleures recettes de gâteaux avec Cyrille, celles interminables sur l'Histoire avec Éric, nos parties d'échecs endiablées avec Walid, ou de jeux de société avec Catalin, nos conversations sans queue ni tête avec Joseph et Alicia, ou celles à perdre la tête avec Ivan. Merci à Sergiu, Younes pour leur soutien tout particulier pendant la rédaction de ma thèse. Et bien sûr tous mes autres collègues de l'équipe. Je remercie chaleureusement tous les habitués du pique-nique des doctorants, dont encore Éric sans qui cette thèse aurait été bien moins joviale. Son récit sur l'histoire de la fourchette à travers les siècles restera sans doute dans les annales du pique-nique. Un grand merci à Laurent qui a su développer des connaissances administratives et les faire partager à un point proche de l'aide humanitaire. Je remercie aussi Svyat qui a réussi à élever le débat pendant ces midis animés. Et merci à Simon, avec qui le *troll* devient une discipline artistique.

Merci à tous mes amis proches mais distants par les kilomètres qui m'ont apportés leur soutien quand ils le pouvaient, merci Jason, Harmony, Benjamin, Rozenn, Hélène, Xavier, Stéphanie, Guilhem, Thomas, Yannick, Jonas, ... pour en citer quelques uns. Merci à ceux moins éloignés qui m'ont soutenu presque quotidiennement. Un grand merci à Oriane qui, plus que les autres peut-être, a dû me supporter pendant l'écriture de mon manuscrit. Merci à mes parents, ma sœur, mon neveu, qui assurent les bons moments et parfois les moins bons. Je remercie tous ceux qui se reconnaitront avec qui j'ai pu discuter autour d'un café, de philosophie, de littérature et même de politique. Pendant ces 3 ans, j'ai pu nouer des amitiés qui vont au delà de simples relations de travail.

Et bien sûr merci à tous ceux qui ont contribué à ma thèse car ils font partie intégrante de la vie au laboratoire. Merci à Martine et Emmanuelle au secrétariat, à Caroline et Angélique à la cafétéria, merci à Isabelle et au personnel du restaurant, au service informatique, et au personnel d'entretien. Sans toutes ces personnes, et bien d'autres que j'ai certainement oublié, ce travail n'aurait pu aboutir.

Table des matières

Introduction

1 Contexte 1
2 Les protocoles cryptographiques 2
2.1 Primitives cryptographiques 2
2.2 Propriétés de sécurité 3
3 La conception de protocoles 3
3.1 Limites dans la conception 3
3.2 Failles logiques 4
4 La vérification dans le modèle symbolique 6
4.1 Les outils de vérification automatique 7
5 Protocoles cryptographiques et secrets faibles 7
5.1 Les protocoles à mots de passe 8
5.2 Les protocoles hors-bande 9
5.3 Vérification 12
6 Contribution 13
7 Résumé 13

Partie I Modèle formel **15**

Chapitre 1
Messages

1.1 Termes 17
1.2 Fonctions de hachage faibles 19
1.3 Connaissance de l'attaquant 21
1.4 Noms devinables 21

Chapitre 2
Modélisation des protocoles

2.1 Syntaxe de processus 23
2.2 Sémantique de processus 25
2.3 Modélisation des canaux hors-bande 26
2.4 Propriétés de sécurité 28

Chapitre 3
Compilation vers un ensemble de traces

3.1 Syntaxe de traces 31
3.2 Sémantique de traces 33
3.3 Traduction depuis une algèbre de processus 34
3.4 Propriétés de sécurité sur les traces 37

Partie II Vérification 41

Chapitre 4
Procédure de décision

4.1 Propriété des variants finis 44
4.2 Modélisation des protocoles 45
4.2.1 Prédicats 45
4.2.2 Statements de la graine 48
4.2.3 Correction et complétude 50
4.2.4 Preuve de la correction et la complétude de la graine 51
4.3 Saturation 56
4.3.1 Procédure de saturation 56
4.3.2 Correction et complétude de la saturation 59
4.3.3 Preuve de la correction et de la complétude de la saturation 60
4.4 Procédure 65
4.4.1 Effectivité de la procédure de saturation 65
4.4.2 Description, correction et complétude de l'algorithme 66
4.4.3 Preuve de la correction et de la complétude de l'algorithme . 67

Chapitre 5
Implémentation

5.1 Fonctionnement d'AKISS original 78
5.1.1 Modélisation 79
5.1.2 Vérification 81

5.2 Adaptation de l'outil AKISS à un attaquant qui devine les noms faibles 83
5.2.1 Modélisation avec des noms faibles 83
5.2.2 Vérification en présence de noms faibles 84

Chapitre 6
Études de cas

6.1 Protocoles utilisant des noms faibles 88
6.2 Protocoles utilisant des fonctions de hachage faibles 92
6.3 Protocoles utilisant des noms faibles et des fonctions de hachage faibles 96
6.4 Autres protocoles 98

Conclusion

Bibliographie **107**

Introduction

1 Contexte

Un protocole est une série d'étapes qui a pour but d'accomplir une action spécifique. Des protocoles sont utilisés dans de multiples domaines, qui vont du protocole administratif contraignant, aux indispensables formules de politesse qui introduisent une conversation. Sur internet, nous prêtons aux protocoles une importance toute particulière. En effet, toutes les communications y sont encadrées par des protocoles, mais à la différence d'une conversation classique, sur internet nous passons par de nombreux relais plus ou moins honnêtes. Nos informations passent donc de mains en mains jusqu'à arriver à notre interlocuteur final. Il devient risqué de transmettre des informations confidentielles, il n'est même pas évident de justifier son identité à notre interlocuteur. D'autres considérations peuvent être prises en compte notamment concernant notre anonymat. Ainsi, les protocoles sur internet sont spécifiquement conçus pour répondre aux besoins de nos activités, tout en nous protégeant. Des considérations de confidentialité et d'authenticité se posent tout particulièrement dans le commerce électronique. Un paiement nécessite de transmettre ses données bancaires au marchand. Ainsi l'on voudra s'assurer qu'un attaquant ne peut voler nos coordonnées bancaires ou usurper notre identité. Pour cela on utilise des protocoles comme 3D-Secure [3DS]. De la même manière, dans les réseaux mobiles, le protocole Bluetooth [BTH01] répond aux mêmes problématiques en permettant d'échanger des données entre deux téléphones proches tout en s'assurant que personne n'intercepte ces données. L'anonymat est nécessaire dans le cas, par exemple, du vote électronique. La nécessité pratique de faire voter des électeurs à distance demande de mettre en place des protocoles pour préserver les conditions requises au vote dans une démocratie. De nombreux protocoles sont conçus dans ce but [GRCC15, RT09, CGGI14, RT09].

Ces protocoles qui ont pour but de protéger les messages qui transitent sur le réseau sont appelés des *protocoles de sécurité*. Cependant les concevoir est une démarche complexe et de nombreuses failles sont régulièrement découvertes [ACC+08, Hbd13, ABD+15, VP17]. Les exploitations de telles failles peuvent être critiques pour les utilisateurs, fuite de données bancaires, récupération de mots de passe, usurpation d'identité pour effectuer des achats frauduleux, perte de l'anonymat d'électeurs, etc...

Malgré les dispositions évidemment mises en place afin d'éviter les failles de sécurité, les protocoles sont sujet à nombreuses attaques. Parmi celles-ci, nous avons la vulnérabilité *Heartbleed* [Hbd13] qui touchait la grande majorité des serveurs de l'Internet courant 2014, ou encore l'attaque découverte sur le processus d'authentification de Google, *Google Single Sign-On* [ACC+08]. Autre exemple, l'attaque *Logjam* [ABD+15] en 2015, qui consistait à forcer l'utilisation d'une version faible du protocole *OpenSSL*,

tout dernièrement, il y a eu aussi l'attaque *KRACK* [VP17] qui permettait de pirater n'importe quel réseau wifi utilisant *WPA2* [HM05].

2 Les protocoles cryptographiques

Afin d'assurer leurs fonctions de sécurité, les protocoles vont utiliser la cryptographie. Pour assurer la confidentialité des échanges, des algorithmes de chiffrement symétrique, c'est-à-dire que l'émetteur et le receveur ont la connaissance commune d'une même clé, vont permettre d'assurer la confidentialité des communications. Cependant, partager cette clé de manière sure est un challenge délicat. En 1976, la cryptographie et avec elle les protocoles cryptographiques vont progresser rapidement à partir de la publication des travaux de Diffie et Hellman [DH76] sur les fonctions à sens unique, ouvrant la voie à la cryptographie à clé publique (ou asymétrique). Avec la cryptographie à clé publique chaque participant va posséder deux clés. Une clé publique qui va permettre de chiffrer des messages et une clé confidentielle permettant de les déchiffrer. Ceci est le fonctionnement le plus courant, mais la cryptographie asymétrique et les fonctions à sens unique de manière générale, ont beaucoup d'autres applications : signature de messages, fonctions de hachage, *etc.*...

2.1 Primitives cryptographiques

Nous pouvons distinguer ces outils cryptographiques du fait des propriétés que leur utilisation va dégager et ainsi les définir en tant que primitives cryptographiques. Une *primitive cryptographique* est une fonction qui va avoir certaines propriétés, et peut donc être utilisée comme un outil pour concevoir un protocole. Les primitives cryptographiques les plus communes sont les suivantes :

- Le *chiffrement symétrique* permet de chiffrer et déchiffrer un message m, ces deux opérations s'effectuent à l'aide d'une même clé k. Un chiffrement symétrique sûr ne permet à personne de retrouver un message m chiffré par une clé k sans connaitre la clé k. En pratique ce type de chiffrement est très utilisé car peu couteux en temps de calcul à réaliser. *AES* [AES01] est aujourd'hui le standard pour faire du chiffrement symétrique.
- Le *chiffrement asymétrique*, permet de chiffrer un message m à l'aide d'une clé k_{pub} afin de faire en sorte que ce message ne puisse pas être déchiffré sans une seconde clé k_{prv}. Ce chiffrement présente l'avantage que la clé publique k_{pub} peut être connue de tous, et permettre ainsi des échanges avec le détenteur de la clé privée k_{prv} de manière confidentielle. En pratique de nombreux types de chiffrements asymétrique sont utilisés, le chiffrement le plus utilisé restant l'algorithme *RSA* [RSA78], mais d'autres méthodes sont utilisées comme des chiffrements basés sur *El-Gamal* [Gam84], sur les courbes elliptiques ou les réseaux euclidiens.
- *Les fonctions de hachage* présentent l'intérêt qu'à un message de grande taille nous pouvons, par application de cette fonction, générer un haché se voulant unique, qui pourra identifier ce message de manière sure. Ce haché peut avoir la propriété d'être plus petit que le message, dans ce cas l'unicité est remplacée par la difficulté à trouver un autre message avec la même image. En pratique on peut utiliser des algorithmes du *Secure Hash Standard (SHS)* [GD95] comme SHA-1 ou son successeur SHA-2.

Nous distinguons la fonctionnalité à laquelle répond la cryptographie et les méthodes afin d'obtenir de telles propriétés. Plusieurs méthodes existent pour vérifier que des outils cryptographiques assurent effectivement les propriétés qu'on peut leur prêter, ces méthodes sont davantage décrites dans la Section 3.

2.2 Propriétés de sécurité

Le puissant outil mathématique qu'est la cryptographie ne suffit pas à lui seul à réaliser un protocole de sécurité. Avant d'établir une session sure ou pour assurer une sécurité plus complexe, plusieurs échanges doivent avoir lieu entre les participants d'un protocole. Ainsi un protocole de sécurité doit pouvoir échanger des messages, s'aidant de la cryptographie, afin d'assurer la sécurité de ses participants. Les primitives cryptographiques ont donc des propriétés particulières qui vont être prises en compte pour répondre à un besoin dans la conception d'un protocole. Ce besoin est caractérisé par la fonction d'un protocole. Quand nous payons via l'Internet par exemple, nous ne voulons pas que nos informations bancaires soient interceptées par l'attaquant, nous ne voulons pas non plus qu'un attaquant puisse éventuellement se faire passer pour un vendeur légitime. Ces considérations dégagent autant de propriétés de sécurité, ici notre protocole devra prendre en compte le besoin de *confidentialité* des données bancaires tout en s'assurant de l'*authenticité* du vendeur. Les propriétés de sécurité que l'on retrouve le plus souvent dans les protocoles sont les suivantes :

— La *confidentialité*, peut concerner un message ou une clé. Cette propriété est respectée si un attaquant ne peut pas retrouver ce message ou cette clé au cours du protocole et de manière plus générale, ne pas obtenir d'information sur ce secret.

— L'*authenticité* ou *authentification*, statue qu'un attaquant ne doit pas être en mesure d'usurper l'identité d'un participant légitime du protocole. Nous pouvons aussi le formuler comme une propriété de correspondance, c'est-à-dire que si un participant légitime du protocole termine sa session en pensant avoir communiqué avec un autre participant légitime, alors c'est effectivement le cas.

— L'*anonymat* signifie qu'un attaquant ne peut distinguer qui de tous les participants éventuels, est effectivement en train d'exécuter le protocole.

Les notions de confidentialité et d'authenticité sont particulièrement utiles dans le paiement en ligne. Tandis que d'autres notions comme l'anonymat vont être nécessaires notamment dans les protocoles de vote électronique, qui vont de manière générale, également nécessiter de nombreuses autres notions, comme l'authenticité du vote par exemple.

3 La conception de protocoles

3.1 Limites dans la conception

La conception de protocoles reste une tâche difficile. Et de nombreuses failles sont découvertes régulièrement dans les protocoles utilisés en pratique. Nous distinguons plusieurs types d'attaques :

— Les attaques les plus fréquentes sont dues à des erreurs d'implémentation. Concevoir un protocole est d'ores et déjà un problème difficile mais les erreurs

d'implémentation sont un facteur encore plus important d'attaques. Ces erreurs peuvent se produire à l'implémentation de primitives cryptographiques, ou dans l'implémentation de sous-protocoles, cassant ainsi la logique de la conception générale. Pour pallier à ce problème, des bibliothèques sont développées permettant d'utiliser les primitives cryptographiques courantes, ou d'utiliser des protocoles entiers en boite noire, la bibliothèque *OpenSSL* [SSL] permet par exemple d'utiliser toutes les fonctionnalités du protocole SSL/TLS. Ces initiatives permettent de concentrer la recherche de failles d'implémentations sur ces bibliothèques et ainsi permettre des implémentations protégées de toutes les failles connues. L'attaque *Heartbleed* [Hbd13] est par exemple basée sur une erreur d'implémentation qui conduisait à la possibilité pour l'attaquant de lire la mémoire interne des systèmes protégés par des versions d'*OpenSSL* vulnérables.

— Les attaques sur la cryptographie proviennent d'erreur lors de la conception de primitives cryptographiques. Par exemple, le chiffrement par flots RC4 présentait un défaut révélé par son utilisation dans le protocole WEP permettant il y a quelques années de protéger nos réseaux wifi, cette attaque fut découverte notamment par Fluhrer *et al.* [FMS01]. Ainsi, la clé secrète ou clé WEP pouvait être retrouvée par un attaquant, qui écoutait les procédures d'initiation de connexions, en quelques minutes si le réseau était actif.

— Un autre type d'attaque correspond aux *failles logiques*. C'est-à-dire que c'est la logique profonde du protocole qui permet à un attaquant de violer les propriétés de sécurité qui auraient dues être assurées. On retrouve ces failles dans le protocole de Needham-Schroeder [NS78], un protocole d'authentification entre deux participants. Une vingtaine d'années après sa conception Lowe découvrit une attaque [Low96], celle-ci permet à un attaquant d'usurper l'identité d'un participant. Une autre faille logique a été découverte dans le protocole *Google Single Sign-On* par Armando *et al.* [ACC+08]. Nous avons encore l'attaque KRACK [VP17] qui, utilisant une faille logique d'attaque par rejeu, va permettre d'apprendre la clé utilisée pour sécuriser les échanges.

Dans cette thèse, nous nous concentrons sur ces failles logiques.

3.2 Failles logiques

Les failles logiques ne concernent donc pas les primitives cryptographiques qui assurent les propriétés qu'on leur prête, mais elles concernent l'entrelacement et la conception des messages échangés entre les participants.

Le protocole le plus célèbre témoignant de ces failles, est le protocole de Needham-Schroeder [NS78]. Ce protocole est décrit dans la Figure 1. Dans cette figure $pk(A)$ et $pk(B)$ représentent respectivement la clé publique de A et de B. La fonction de chiffrement à clé publique $\{m\}_{pk(A)}$ correspond au chiffrement du message m par la clé publique de A. L'information N_A est un *nonce*, c'est-à-dire un nombre aléatoire qui n'est pas susceptible d'être réutilisé dans d'autres instance du protocole (contrairement à un mot de passe). Ce nonce N_A est généré par A et N_B est un nonce généré par B. Suivant ce protocole, les deux participants se mettent d'accord sur la valeur des nonces N_A et N_B.

Il fonctionne selon le principe de *challenge/response*. Ainsi, A envoie à B un nonce connu d'elle seule, chiffré avec la clé publique de B. B va ensuite montrer à A qu'il

est capable de le déchiffrer. Comme preuve, il renvoie le nonce de A avec un nonce qu'il aura généré, ceci chiffré avec la clé de A. A termine le protocole montrant à B qu'il a réussi à déchiffrer son nonce. À la fin du protocole les deux participants sont convaincus de s'être échangé deux nonces, ils peuvent être utilisés par la suite pour communiquer.

$$\begin{array}{lcll} A & \rightarrow & B: & \{N_A, A\}_{pk(B)} \\ B & \rightarrow & A: & \{N_B, N_A\}_{pk(A)} \\ A & \rightarrow & B: & \{N_B\}_{pk(B)} \end{array}$$

FIGURE 1 – Protocole Needham-Schroeder

Le protocole de Needham-Schroeder a été conçu en 1978, il était sujet à une attaque qui fut démontrée par Lowe [Low96] une vingtaine d'années plus tard.

Attaque : L'attaque découverte par Lowe [Low96] consiste à rejouer une session entre un participant honnête A et un participant malhonnête I, pour démarrer une session frauduleuse entre A et B. Cette attaque est de type *homme du milieu*, c'est-à-dire que l'attaquant I va pouvoir se placer au milieu des échanges entre A et B afin d'agir sur le protocole pour violer sa sécurité. Nous décrivons cette attaque dans la Figure 2. Cette attaque va utiliser deux sessions différentes du protocole. Une première entre A et I, et une seconde entre I se faisant passer pour A et B. Le participant A va démarrer la première session avec I, ainsi il lui envoie un challenge (1). C'est là que I va commencer la deuxième session avec B, il doit donc lui fournir un challenge, il forge un challenge à partir de celui que A lui a envoyé dans la première session (2). Ce challenge contenant l'identité de A, B s'adresse à A pour envoyer sa réponse accompagnée son propre challenge (3). A pensant avoir reçu cette réponse de I, envoie à I le résultat du challenge (4). I peut ainsi, en se faisant passer pour A, envoyer la réponse au challenge de B. Ainsi, le protocole est sujet à une attaque. A a commencé une session avec B alors qu'il pensait l'avoir commencé avec I. Et I connait les nonces N_A et N_B utilisés.

$$\begin{array}{llcll} (1) & A & \rightarrow & I: & \{N_A, A\}_{pk(I)} \\ (2) & I(A) & \rightarrow & B: & \{N_A, A\}_{pk(B)} \\ (3) & B & \rightarrow & A: & \{N_B, N_A\}_{pk(A)} \\ (4) & A & \rightarrow & I: & \{N_B\}_{pk(I)} \\ (5) & I(A) & \rightarrow & B: & \{N_B\}_{pk(B)} \end{array}$$

FIGURE 2 – Attaque sur le protocole Needham-Schroeder

La faille présente dans le protocole de Needham-Schroeder est une faille logique, la cryptographie remplit son rôle mais c'est dans la conception même du protocole que l'attaque se niche. Ainsi, vérifier si des protocoles peuvent être sujets à des failles logiques représente un enjeu majeur.

4 La vérification dans le modèle symbolique

Les failles logiques posent un problème complexe à la conception de protocoles. Elles demandent un fort besoin de méthodes de vérification rigoureuses. La faillibilité de la logique humaine doit être compensée par une preuve de sécurité qui doit accompagner tout protocole.

Afin de vérifier ces protocoles, il nous faut représenter tout ce qui concernerait une attaque logique. Ainsi, nous pouvons *modéliser l'attaquant par ce qu'il est en mesure de réaliser*, il peut par exemple faire usage de cryptographie, ou la casser dans des cas particuliers ; *par ce qu'il sait et peut apprendre au cours du protocole*, ce qui définit sa connaissance ; *par ce qu'il contrôle dans le réseau*, peut-il intercepter des messages, en forger de nouveaux, sur quels canaux spécifiques ? Il nous faut également modéliser *ce que fait le protocole* et surtout *à quels objectifs il répond*, c'est-à-dire quelles propriétés de sécurité doivent être assurées. Ainsi, nous décrivons ci-dessous ce que nous entendons par modèle symbolique, quelles méthodes nous pouvons utiliser pour vérifier des protocoles, et comment nous procédons.

Le modèle symbolique : Nous nous concentrons ici sur les failles logiques, nous devons donc nous concentrer sur la logique des échanges entre les participants. Une façon effective de procéder est d'utiliser le modèle symbolique dit de Dolev-Yao [DY83]. Dans ce modèle, nous considérons un attaquant qui a tout contrôle sur le réseau, c'est-à-dire qu'il peut écouter, intercepter et forger des messages. Ces messages sont échangés entre les participants et vus comme des termes. Ces termes sont construits à partir de primitives cryptographiques dont les propriétés sont définies préalablement et sont intangiblement respectées.

Procédures de vérification : Vérifier en prenant en compte des failles logiques, si le protocole assure des propriétés de sécurité particulières peut vite devenir fastidieux. Ainsi, plusieurs travaux visent à automatiser ces procédures [Bla01, Bla16, SMCB12, BDS15, CcCK12]. Ces procédures vont considérer la modélisation du protocole, des capacités de l'attaquant, des propriétés de sécurité que le protocole doit respecter et vont permettre de décider si ces propriétés peuvent êtres violées par l'attaquant.

Les propriétés d'atteignabilité : Ces propriétés sont typiquement utilisées pour vérifier des notions de confidentialité ou d'authenticité. Dans le cas de la confidentialité, cela consistera à vérifier que pendant l'exécution du protocole, un attaquant ne peut pas déduire le message que l'on veut confidentiel. Dans le cas de l'authenticité, on s'attachera à vérifier qu'un participant légitime qui pense avoir terminé sa session avec un autre participant légitime l'a effectivement fait. Ainsi, nous nous assurerons que l'on ne peut atteindre un point où un des participants termine sa session en ayant été abusé sur son interlocuteur réel.

Les propriétés d'équivalence : Cette notion plus récente permet de vérifier d'autres types de propriétés de sécurité sur les protocoles. Intuitivement, une propriété d'équivalence est respectée si un attaquant ne peut pas distinguer deux exécutions différentes d'un même protocole, ou de deux protocoles différents. Ainsi, cette notion de distinguabilité nous permet de vérifier des propriétés de confidentialité forte ou d'anonymat.

4.1 Les outils de vérification automatique

Dans le cas général, vérifier si un protocole cryptographique respecte des propriétés, même simples, comme la confidentialité faible et l'authentification est indécidable [NDS99]. Cependant, dans le cas d'un nombre borné de sessions, vérifier la sécurité d'un protocole cryptographique devient un problème NP-complet [RT01]. Ainsi plusieurs outils vont tenter de vérifier des protocoles cryptographiques qui ne terminent que dans des cas particuliers ou vont se concentrer sur un nombre borné de sessions. Il existe deux grandes familles d'outils :

Nombre non borné de sessions : Comme nous l'avons précisé dans le cas d'un nombre non borné de sessions, vérifier la sécurité des protocoles cryptographiques est un problème indécidable. Cependant, des outils parviennent à vérifier certaines classes de protocoles en terminant la plupart du temps. Ainsi, ProVerif [Bla01, Bla16], développé en premier lieu afin de vérifier des propriétés d'atteignabilité, a été étendu par la suite pour prouver des propriétés d'équivalence particulières [BAF05]. Cet outil va représenter le protocole et les capacités de l'attaquant à l'aide de clauses de Horn. Il permet à l'utilisateur de définir de nombreux types de primitives cryptographiques afin de modéliser un protocole. Il pourra également utiliser des canaux privés permettant de représenter des échanges synchrones que l'attaquant ne peut pas espionner ou falsifier. ProVerif, considérant un nombre non borné de sessions, peut ne pas terminer, mais il peut également produire de faux-négatifs [CB13, DRS08]. Un autre outil, Tamarin [SMCB12, BDS15] utilise le *multiset rewriting* et peut soit exhiber des attaques ou prouver l'équivalence. Tamarin ne termine pas pour un nombre important de théories, cependant un mode interactif permet un guidage manuel améliorant ses performances.

Nombre borné de sessions : L'analyse de protocoles à nombre borné de sessions est décidable [RT01], ce qui en fait une sous-classe de l'analyse de protocoles très étudiée. Ce résultat a été étendu pour traiter d'autres primitives cryptographiques comme l'exponentiation modulaire ou le XOR [CKRT03, CS03, Shm04, KT11, DKP12, BDGK17]. De nombreux outils font suite au résultat de 2001. Notamment la suite d'outils AVISPA [ABB$^+$02, ABB$^+$05, AAA$^+$12] qui combine plusieurs techniques pour analyser les protocoles, ou l'outil Apte [CCD17], spécialisé dans l'équivalence de traces. Cette thèse sera portée sur un de ces outils, l'outil AKISS [CcCK12], qui se concentre également sur l'équivalence de traces. Il peut supporter un nombre important de primitives et fournit également des témoins de non-équivalence quand il ne parvient pas à prouver le contraire.

5 Protocoles cryptographiques et secrets faibles

Le modèle symbolique abstrait la taille réelle des messages. Ainsi, certains messages sont parfois de très petite taille (quelques digits), pendant qu'une clé cryptographique est de très grande taille afin d'assurer au mieux la sécurité des fonctions associées. Dans le modèle symbolique, le fait que des noms représentent des noms de petites ou grandes taille n'est pas une information conservée a priori. Cependant, une telle information peut être utile afin de considérer un attaquant qui puisse utiliser la petite taille des chaines pour attaquer le protocole. Un secret faible est un secret qui, de par la faible

entropie due à sa petite taille ou son petit espace de définition, va peut être pouvoir être deviné par l'attaquant sous certaines conditions. En effet, la faible entropie du secret se caractérise par le fait que l'attaquant peut itérer sur toutes les valeurs qu'il pourrait prendre. C'est la puissance de calcul de l'attaquant qui à terme définit si un secret est faible ou non.

5.1 Les protocoles à mots de passe

Les mots de passe sont souvent utilisés dans les protocoles afin d'assurer des propriétés d'authentification, comme dans EKE [BM92] ou [BMP00, ZDW06, AP05]. On peut les considérer comme des secrets faibles car quand ils sont définis par les utilisateurs d'un service, ils sont souvent de petite taille, ou alors certains reviennent de manière plus fréquente [Dup13]. Dans ces protocoles, la sécurité dépend de la robustesse du mot de passe. Les mots de passe comme clé de voute de ces protocoles sont régulièrement attaqués par des attaques dites d'*ingénierie sociale*, par exemple à l'aide de méthodes d'*hameçonnage*, mais ils peuvent également être victimes de failles logiques. En ce sens qu'un attaquant pourrait retrouver un mot de passe en agissant sur une ou plusieurs utilisations du protocole par les participants. Dans ce cas, le but de l'attaquant sera d'exploiter la faible entropie du mot de passe et d'effectuer une *attaque par force brute*. C'est-à-dire que l'attaquant va itérer sur de nombreuses valeurs possibles du mots de passe, jusqu'à trouver la valeur correcte. Ces attaques par force brute peuvent être effectuées *en-ligne*, et donc en prenant part de manière active à plusieurs sessions du protocole, testant pour chaque session un nouveau mot de passe. Mais cette approche est très vite discriminée en ajoutant des contre-mesures comme définir 3 essais maximum avant bannissement de l'utilisateur et donc du mot de passe de la procédure protocolaire. Une autre approche consiste à essayer de deviner le mot de passe par une attaque *hors-ligne*, c'est-à-dire que l'attaquant va, depuis la connaissance qu'il aura obtenu en intervenant dans une ou plusieurs sessions du protocole, essayer de deviner localement, donc sans intervenir davantage avec le protocole, le mot de passe. Ces attaques sont plus gênantes car plus difficile à éviter, il faut donc veiller à ne pas laisser de faille logique permettant à l'attaquant d'obtenir suffisamment de connaissance pour effectuer une telle attaque.

Les attaques hors-ligne par force brute ou attaque par devinette : Si l'attaquant a une puissance de calcul suffisante en sa possession, il est en capacité d'éventuellement *deviner* un mot de passe ou toute autre chaine de caractères de petite taille. Pour cela, il pourra itérer sur toutes les valeurs possibles de cette courte chaine, ou essayer tous les mots de passe possibles. Ainsi, dans le protocole de la Figure 3 ci-dessous, nous détaillons un protocole jouet utilisant un secret faible s pouvant être un mot de passe ou tout autre chaine de caractères ou valeur ayant une faible entropie, ainsi qu'une fonction de hachage hash.

$$1.\quad A \;\rightarrow\; B : \;\; \mathsf{hash}(s)$$

FIGURE 3 – Protocole jouet utilisant un secret faible

Ici, un attaquant peut effectuer une attaque hors-ligne par force brute. Il lui suffit

d'écouter le message que l'agent A envoie à B : $\mathsf{hash}(s)$. Il peut ensuite essayer toutes les valeurs possibles s' de s et comparer à chaque fois si $\mathsf{hash}(s') = \mathsf{hash}(s)$. De cette manière, l'attaquant sait quand il a trouvé la bonne valeur pour s', c'est-à-dire quand il a deviné s.

5.2 Les protocoles hors-bande

La particularité de ces protocoles vient du fait qu'ils utilisent une intervention physique d'un ou plusieurs participants du protocole afin de satisfaire des propriétés plus fortes d'authentification. Parfois, c'est une simple sécurité supplémentaire, on parle alors d'authentification multi-facteurs [Dis13]. D'autres fois l'authentification dépend uniquement de leur utilisation. Cette intervention physique d'un participant à donc un fort avantage, mais présente aussi l'inconvénient que l'action doit être à la portée d'un humain. Ainsi, il n'est pas question de le forcer à recopier une clé de plusieurs dizaines de caractères. À la différence d'un canal classique, qui correspondrait à l'établissement d'une communication via Internet, un *canal hors-bande* définit les échanges qui utilisent une interaction physique d'un humain afin d'être menés à bien.

Canal hors-bande : On appellera l'axe de communication entre deux participants qui vont utiliser une interaction humaine pour échanger des données un *canal hors-bande*. Par exemple, cet échange peut s'illustrer de la manière suivante : disons que O_A représente l'ordinateur de l'humain H_A et O_B, l'ordinateur du l'humain H_B. O_A dans le cadre d'un protocole peut envoyer un message sur le téléphone de l'humain H_B. H_B va ensuite retranscrire le contenu de ce message à O_B. On représentera dans les figures suivantes les canaux hors-bande par $\rightarrow_e$. Ces canaux sont considérés ici authentiques, nous verrons par la suite les différentes hypothèses que l'on peut considérer.

Des protocoles hors-bande sont utilisés dans de nombreux domaines. Par exemple, dans le paiement en ligne avec 3D-Secure [3DS], dans les systèmes d'échange sur appareils embarqués avec le Bluetooth [BTH01], ou encore dans le vote électronique avec notamment le protocole Norvégien [Gjø11] ou DU-vote [NR11]. On s'intéresse dans cette thèse à la vérification du standard ISO9798-6 :2010 [ISO10] qui décrit des mécanismes d'authentification utilisant une interaction manuelle sur des appareils spécifiques.

L'interaction humaine vient avec son lot de contraintes. Comme nous l'avons énoncé précédemment, un canal hors-bande ne peut faire transiter des grandes chaines de chiffres. Un utilisateur humain ne pouvant recopier ces digits sans erreurs. Ainsi, les messages sur ces canaux doivent être de petite taille. Nous présentons maintenant deux types d'utilisation de ces canaux. Les protocoles qui vont utiliser des nonces de petite taille et ceux qui vont utiliser des fonctions de hachage qui renvoie un entier de petite taille.

Les protocoles utilisant des nonces faibles : Un premier type de message que l'on peut confier à un participant humain, ce sont des nonces faibles, c'est-à-dire des entiers aléatoires, de quelques digits. Ainsi, la faible entropie de ces nonces en fait nécessairement des secrets faibles. Ces secrets faibles sont soumis, tout comme les mots de passe, à des *attaques par devinette*, mais contrairement aux protocoles à mots de passe, un nonce faible deviné par un attaquant ne peut pas être utilisé dans une

nouvelle session du protocole, parce que ce nonce est généré aléatoirement à chaque début de session et ne peut donc à lui seul permettre de casser toutes les prochaines sessions d'un protocole. C'est pourquoi deviner un nonce faible ne peut pas être l'enjeu pour un attaquant souhaitant casser la sécurité du protocole, mais il peut avoir besoin de deviner ce nonce faible afin de s'en servir pour violer une propriété de sécurité du protocole.

Dans la Figure 4, nous présentons un protocole hors-bande qui utilise un secret faible. Ce protocole est une dérivation volontairement simplifiée affaiblie du *mécanisme d'authentification manuelle 4* que nous présentons dans la Section 6.1 et qui est aussi décrit dans le standard ISO9798-6 [ISO10]. La version simplifiée mais vérifiée sure de ce protocole est détaillée dans la Figure 5. Nous considérons dans cette figure un nonce de petite taille, r_A, généré par A, et $info_A$ une information détenue par A, que A veut transmettre à B de manière authentique. Pour s'assurer d'avoir effectivement reçu $info_A$ de A, B va vérifier si la partie droite du premier message qu'il a reçu : $md_1 = \mathsf{hash}(info_A, r_A)$, est égale à l'application de la fonction de hachage hash, sur la partie gauche du premier message reçu $mg_1 = info_A$, et le deuxième message reçu $m_2 = r_A$. Il teste donc $\mathsf{hash}(mg_1, m_2) \stackrel{?}{=} md_1$.

$$
\begin{array}{llll}
1. & A & \rightarrow & B : \; info_A, \mathsf{hash}(info_A, r_A) \\
2. & B & \rightarrow_e & A : \; \mathsf{ack} \\
4. & A & \rightarrow_e & B : \; r_A
\end{array}
$$

FIGURE 4 – Version affaiblie et simplifiée d'un protocole du standard ISO9798-6

Dans ce dernier protocole, A doit d'abord fournir à B la valeur de sa donnée $info_A$ et il va s'engager sur la valeur de r_A. Quand B aura reçu ce message, A va pouvoir révéler r_A. Ainsi, A s'est engagé sur la valeur de r_A avant de le révéler. Ce procédé s'appelle *l'engagement avant connaissance* ou *mise en gage*.

Sécurité induite de l'engagement avant connaissance (ou mise en gage) : Dans les protocoles des Figures 4 et 5, la sécurité vient du fait que A doit dans un premier temps, s'engager sur la valeur du haché, md_1 (donc sur ses paramètres), auprès de B. Une fois que A est sure que B a reçu le message mg_1, md_1, grâce à la réception de l'acquittement authentique de B, A va révéler le secret qui, plus tôt, aurait permis à l'attaquant de construire un autre haché pour B, vérifiant une autre donnée et trompant B. Ce mécanisme de sécurité se nomme *engagement avant connaissance* ou *mise en gage*.

Attaque sur le protocole de la Figure 4 : L'attaquant peut effectuer une attaque hors-ligne par force brute. En effet, après le premier échange de A vers B, l'attaquant connait $info_A$ et $\mathsf{hash}(info_A, r_A)$, il peut ainsi itérer sur les valeurs possibles de r_A, symbolisons ce candidat par r', puis vérifier si il a deviné r_A en testant $\mathsf{hash}(mg_1, r') \stackrel{?}{=} md_1$.

La Figure 5, détaille une version sécurisée du protocole de la Figure 4. Le nonce faible r_A est généré par A au début d'une session du protocole. Le symbole hash représente

une fonction de hachage, s'appliquant cette fois-ci à 3 paramètres. Le nonce n_A est un nonce de grande taille généré par A au début de chaque session, il est impossible pour l'attaquant de deviner un nonce de grande taille par force brute. Dans cette version montrée sure par l'outil AKISS, la sécurité vient du fait que l'attaquant ne peut se contenter des parties gauche et droite du premier message mg_1 et md_1 afin de deviner r_A. Pour s'assurer de l'authenticité d'$info_A$, B va considérer son second message reçu $m_2 = n_A$ et le troisième $m_3 = r_A$. Il va vérifier si $\mathsf{hash}(mg_1, m_2, m_3) \stackrel{?}{=} md_1$.

1. $A \rightarrow B : info_A, \mathsf{hash}(info_A, n_A, r_A)$
2. $B \rightarrow_e A : \mathsf{ack}$
3. $A \rightarrow B : n_A$
4. $A \rightarrow_e B : r_A$

FIGURE 5 – Version épurée d'un protocole du standard ISO9798-6

Les protocoles qui utilisent des fonctions de hachage faible : Un autre type de message que l'on peut envoyer correspond au résultat d'une fonction de hachage qui comprime fortement ses paramètres. Une telle fonction devient bien plus sensible aux attaques par collision. Intuitivement, il est facile pour un attaquant de trouver d'autres paramètres de cette fonction de hachage donnant le même résultat après application. Nous présentons dans les figures 6 et 7 des protocoles basés sur *le mécanisme d'authentification 3* détaillé dans le standard ISO 9798-6 [ISO10]. De nombreux autres types de protocoles utilisant des fonctions de hachage faibles existent [NR11, ISO10].

Attaque de collisions : Une attaque de collisions a pour but de trouver deux paramètres, ou séquences de paramètres, d'une fonction de hachage qui produisent le même résultat après application de la fonction. Ainsi, si un attaquant observe $\mathsf{hash}(t)$, son but sera de trouver un paramètre t' tel que $\mathsf{hash}(t) = \mathsf{hash}(t')$. Les fonctions de hachage sujettes à ce type d'attaque seront notées sh.

La Figure 6 détaille un protocole sujet aux attaques de collisions. Ici, A va générer pour chaque session un nonce de grande taille : k. Le but de ce protocole est pour A d'envoyer une donnée $info_A$. A et B doivent être convaincus qu'ils utilisent la même donnée $info_A$, ainsi B doit se convaincre que cette donnée est authentique, c'est-à-dire qu'elle vient bien de A. Pour s'assurer que la donnée $info_A$ est authentique B va comparer le haché qu'il a reçu comme deuxième message : $m_2 = \mathsf{sh}(k, info_A)$ au haché qu'il va reconstruire à partir du premier message $m_1 = info_A$ et du troisième message reçu $m_3 = k$. Il va donc tester $m_2 \stackrel{?}{=} \mathsf{sh}(m_3, m_1)$.

1. $A \rightarrow B : info_A$
2. $B \rightarrow_e A : \mathsf{ack}$
3. $A \rightarrow_e B : \mathsf{sh}(k, info_A)$
4. $A \rightarrow B : k$

FIGURE 6 – Version simplifié et affaiblie du mécanisme 3 de [ISO10]

Attaque sur le protocole de la Figure 6 : Ce protocole est sujet à une attaque de collisions. L'attaquant observe (ou calcule) le haché $\mathsf{sh}(k, info_A)$, comme cette fonction est faible l'attaquant va pouvoir itérer sur les valeurs possibles de k, appelons le sujet de l'itération k', jusqu'à trouver une collision telle que $\mathsf{sh}(k', info_I) = \mathsf{sh}(k, info_A)$ et ainsi B reconnaitra la donnée fournie par l'attaquant comme la donnée authentique fournie par A.

La Figure 7 détaille une version sécurisé du protocole de la Figure 6. Dans cette figure, le symbole sh est une fonction de hachage faible, et hash représente une fonction de hachage classique. Le nonce k est de grande taille et généré par A. Concernant les messages reçus de B nous pouvons distinguer la partie gauche du premier message $mg_1 = info_A$ et sa partie droite $md_1 = \mathsf{hash}(k)$, le deuxième message $m_2 = \mathsf{sh}(k, info_A)$ et le troisième message $m_3 = k$. Pour s'assurer de l'authenticité de la donnée $info_A$, B vérifiera $m_2 \stackrel{?}{=} \mathsf{sh}(m_3, mg_1)$ et $\mathsf{hash}(m_3) \stackrel{?}{=} md_1$.

$$
\begin{array}{llll}
1. & A & \rightarrow & B : \ info_A, \mathsf{hash}(k) \\
2. & B & \rightarrow_e & A : \ \mathsf{ack} \\
3. & A & \rightarrow_e & B : \ \mathsf{sh}(k, info_A) \\
4. & A & \rightarrow & B : \ k
\end{array}
$$

FIGURE 7 – Version épurée du mécanisme d'authentification manuelle 3 de [ISO10]

Les protocoles hors-bande se distinguent des protocoles à mots de passe car la sécurité du protocole ne repose pas tant sur le fait que l'attaquant peut deviner un secret faible mais sur le moment précis où l'attaquant peut le faire. Si cela n'est possible que tardivement, cela ne peut avoir aucun impact sur la sécurité du protocole.

5.3 Vérification

Il est nécessaire de vérifier ces protocoles utilisant de courtes chaines. Depuis le papier de Gavin Lowe [Low04], plusieurs travaux ont été effectués pour caractériser les attaques par devinette sur des protocoles à mots de passe et proposent des méthodes d'analyse [CMAFE03, CDE05, DJ06, DKR08, CDK11]. De nombreux outils permettent de vérifier qu'il n'y a pas attaques par devinette dans ces protocoles, les outils ProVerif [Bla16] ou AKISS [CcCK12] utilisent pour cela des propriétés d'équivalence.

La confidentialité du mot de passe est une condition sine qua non à la sécurité des protocoles basés sur ce mécanisme. Quand nous considérons un mot de passe faible, le protocole n'est évidemment plus sûr si le mot de passe peut être deviné, il pourra être utilisé par l'attaquant dans une deuxième session du protocole. Dans le cas des protocoles hors-bande, l'enjeu est différent : ici les noms faibles sont spécifiques à l'interaction avec l'utilisateur, et sont donc périmés d'une session à l'autre. On peut même voir dans les protocoles montrés en exemple que l'on révèle ces secrets faibles au cours d'une exécution normale du protocole. Il faut donc considérer la capacité de l'attaquant à deviner ces secrets faibles et à l'ajouter à sa connaissance afin d'en tirer possiblement avantage pour violer la sécurité du protocole. Dans ce sens, plusieurs travaux ont

été réalisés comme ceux de Roscoe *et al.* [RSN12], de Delaune et Jacquemard [DJ06] et de Chothia *et al.* [CS10]. Dans [RSN12], ils proposent l'utilisation de l'algèbre de processus CSP et le *model-checker* FDR afin d'analyser des protocoles se basant sur des secrets faibles. Cependant, leur outil est restreint à des messages de taille bornée et des primitives cryptographiques particulières, analysant un système à états finis et n'autorisant pas des théories équationnelles définies par l'utilisateur. Dans [DJ06], une procédure de décision est proposée pour résoudre un problème similaire à celui que nous étudions ici, mais en se basant sur une configuration plus simple. En effet, les tests de dis-égalité ne sont pas considérés, les auteurs s'attachent à un ensemble fixé de primitives ne permettant donc pas de modéliser des fonctions de hachage faibles. De plus, aucune implémentation dans un outil n'est fournie et la procédure est loin de pouvoir être intégrée en l'état. Les travaux de [CS10] utilisent l'outil ProVerif [Bla16] pour analyser des protocoles sur la base de la mise en gage. Leur travail a inspiré notre codage des fonctions de hachage faibles dans la théorie équationnelle, permettant ainsi à l'attaquant de trouver des collisions. Mais ils n'offrent pas de support pour les secrets faibles qui pourraient être devinés et appris par un attaquant au cours de l'exécution.

6 Contribution

La vérification de propriété d'authentification en présence de noms faibles : Peu de travaux prennent en considération l'attaque par force brute que peut utiliser l'attaquant, non pas comme un but mais comme un moyen de casser la propriété d'authentification de nos protocoles. Les travaux en ce sens sont limités et ne permettent pas notamment de s'accompagner d'une théorie équationnelle (représentant les comportements des primitives cryptographiques) pouvant être définie par l'utilisateur avec des libertés conséquentes ou se bornent à des tailles de messages finies. Nous avons proposé des définitions permettant d'exprimer les capacités d'un attaquant à exploiter la faiblesse de courtes chaines. Nous avons pu exprimer notre problème de vérification de tels protocoles en nous concentrant sur des propriétés d'authentification. Nous avons également mis au point une procédure de décision permettant de vérifier ces propriétés, notamment en représentant le protocole avec des clauses de Horn et appliquant une procédure de résolution dédiée.

Une implémentation dans l'outil AKISS : Nous avons fourni une implémentation de cette procédure dans l'outil AKISS [AkG]. Nous avons également proposé de nombreuses optimisations notamment concernant le nombre de traces générées avant l'application de la procédure de décision. Avec cette implémentation, nous avons vérifié le standard de mécanismes à transfert manuel de données ISO 9798-6 [ISO10].

Ce travail a donné lieu à une publication à CSF 2017 intitulée *Formal Verification of Protocols Based on Short Authenticated Strings* [DKR17].

7 Résumé

Nous décrivons un modèle et définissons les notions de noms devinables et des fonctions de hachage faible dans le Chapitre 1. Dans le Chapitre 2, nous décrivons les

protocoles comme des processus et nous expliquons comment exprimer des propriétés d'authentification sur ces processus. Le Chapitre 3 traite de la traduction de ces processus en ensemble de traces. Les traces de cet ensemble vont permettre ensuite d'appliquer notre procédure de décision décrite dans le Chapitre 4 afin de décider de la sécurité de notre protocole. Dans le Chapitre 5 nous expliquons comment cette procédure a été intégrée dans l'outil AKISS. Pour ce faire, des optimisations ont été réalisées, elles sont décrites dans ce chapitre. Dans le Chapitre 6, nous utilisons cette nouvelle version de l'outil AKISS afin de vérifier les protocoles du standard ISO 9798-6 [ISO10].

Première partie

Modèle formel

Chapitre 1

Messages

Sommaire

1.1 Termes 17
1.2 Fonctions de hachage faibles 19
1.3 Connaissance de l'attaquant 21
1.4 Noms devinables 21

Pour modéliser un protocole il nous faut nécessairement modéliser les messages que les participants du protocole s'échangent. Nous montrons donc tout d'abord comment nous modélisons des messages sous la forme de termes. Ces termes sont accompagnés d'une théorie équationnelle permettant de représenter les propriétés des fonctions cryptographiques. La liberté donnée par la théorie équationnelle nous permet notamment de modéliser les fonctions de hachage faible. Puis, nous nous intéressons à l'attaquant et comment il stocke les messages qui correspondent à sa connaissance. De cette connaissance, l'attaquant pourra, sous des contraintes réalistes, deviner des noms faibles.

1.1 Termes

Dans le modèle symbolique [DY83], on représente couramment les messages par des termes. On peut distinguer différents types de *noms*. Des noms *publics* représentent typiquement des identifiants connus à la fois de l'attaquant et des participants. Ils représentent également les nonces (nombres aléatoires à usage unique) générés par l'attaquant. Des noms *privés* sont utilisés pour représenter des nonces ou des clés générés par des participants honnêtes. Ces noms ont trait à modéliser le caractère secret de ces valeurs générées par les participants. Les noms privés sont d'une taille suffisamment conséquente pour qu'un attaquant ne puisse pas raisonnablement deviner leur valeur en faisant plusieurs essais aléatoires. Au contraire, nous définissons un autre type de noms, les noms *devinables* qui peuvent potentiellement être devinés par un attaquant sous certaines conditions. Ils représentent donc des données à faible entropie comme les mots de passe courts ou les codes PIN.

Ainsi, on considère trois ensembles infinis et disjoints de *noms* :
— $\mathcal{N}_{\mathsf{prv}}$, l'ensemble des noms *privés* ;
— $\mathcal{N}_{\mathsf{pub}}$, l'ensemble des noms *publics* ;

— $\mathcal{N}_{\mathsf{guess}}$, l'ensemble des noms *devinables*.

L'ensemble des noms, noté $\mathcal{N}$, est défini tel que $\mathcal{N} = \mathcal{N}_{\mathsf{prv}} \uplus \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}}$. On utilise des symboles de fonctions pour représenter des primitives cryptographiques. On considère une *signature* Σ comme un ensemble fini de symboles de fonctions couplés avec leur arité. On appelle des *constantes* les symboles de fonction d'arité 0. Les variables et les noms sont des *atomes*.

Exemple 1 *Une signature typique pour le protocole présenté en Figure 5 serait :*

$$\Sigma_{\mathsf{hash}} = \{\mathsf{ack}/0, \mathsf{hash}/1, \langle ., . \rangle/2, \mathsf{proj}_1/1, \mathsf{proj}_2/1\}$$

ack *est une constante,* hash *représente la fonction de hachage,* $\langle ., . \rangle$ *représente la fonction d'appariement et* proj_1 *(resp.* proj_2*) permet d'accéder au premier (resp. deuxième) élément de cet appariement.*

On considère également un ensemble de *variables* contenant notamment un ensemble infini $\mathcal{X}$. Les variables dans $\mathcal{X} = \{x, y, \ldots\}$ permettent de représenter les parties inconnues des messages, nous verrons dans le Chapitre 2 où celles-ci sont utilisées pour modéliser des protocoles. On peut donc construire des termes à l'aide de noms dans $\mathcal{N}$, de fonctions dans Σ, et de variables dans $\mathcal{X}$.

Définition 1 (Termes) *Soit $\mathcal{A}$ un ensemble d'atomes et Σ une signature. On note $\mathcal{T}(\Sigma, \mathcal{A})$ le plus petit ensemble qui contient $\mathcal{A}$ et est clos par application des symboles de fonction dans Σ en respectant l'arité associée.*

Ainsi, ces termes nous permettent de représenter des messages. Un terme est lui-même composé de sous-termes. Soit p une séquence d'entiers avec ϵ la séquence vide, le *sous-terme* à la *position* p d'un terme t, est notée $t|_p$. Formellement $t|_\epsilon = t$ et $\mathsf{f}(t_1, t_2, \ldots, t_n)|_{ip} = t_i|_p$. Une *substitution* est un remplacement de variables vers des termes. L'application d'une substitution σ à un terme u est écrite $u\sigma$, et on note $\mathrm{dom}(\sigma)$ son *domaine*, i.e. $\mathrm{dom}(\sigma) = \{x \mid \sigma(x) \neq x\}$. On note la substitution identité, qui a comme domaine l'ensemble vide, $\emptyset$.

Définition 2 (Unification) *On dit que deux termes s et t sont* unifiables *(syntaxiquement) s'il existe une substitution σ telle que $s\sigma = t\sigma$.*

L'unificateur le plus général entre deux termes unifiables s et t est noté $\mathrm{mgu}(s, t)$.

Cet unificateur le plus général est unique à un renommage de variable près [BS01].

Il nous faut maintenant, pour représenter les primitives cryptographiques de nos protocoles, exprimer le lien entre ces symboles. Typiquement, une signature est suffisante pour modéliser le comportement d'une fonction de hachage classique. Cependant, pour des fonctions ayant un lien entre elles nous avons besoin d'accompagner la signature d'une théorie équationnelle pour représenter ce lien.

Exemple 2 *Prenons comme exemple le cas d'un protocole qui utilise du chiffrement symétrique. On considère la signature suivante :*

$$\Sigma = \{\mathsf{enc}/2, \mathsf{dec}/2\}$$

Ainsi, on peut construire un terme qui représente le chiffrement du message m par la clé k en écrivant $\mathsf{enc}(m, k)$. *Pour modéliser le comportement d'une fonction de déchiffrement il nous faut y associer une théorie équationnelle capturant le résultat d'un déchiffrement sur un message chiffré.*

$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x$$

Cette théorie équationnelle signifie que pour tout message x chiffré par la clé y on peut le déchiffrer en utilisant cette même clé y et obtenir de nouveau le message x.

La *théorie équationnelle* E sur la signature Σ est donc définie par un ensemble d'équations de la forme $u = v$ où $u, v \in \mathcal{T}(\Sigma, \mathcal{X})$, et induit une relation d'équivalence sur les termes : $=_{\mathsf{E}}$, celle-ci est la plus petite relation de congruence sur les termes qui contient toutes les équations $u = v$ de E, et qui est close par substitution des variables par des termes. On remarque que les équations ne contiennent pas de noms, mais uniquement des termes construits à partir des variables et des symboles de la signature.

Exemple 3 *On représente le protocole de la Figure 4 en considérant la signature définie dans l'Exemple 1 et la théorie équationnelle* $\mathsf{E}_{\mathsf{hash}}$ *définie par les équations :*

$$\mathsf{proj}_1(\langle x, y\rangle) \;=\; x \;\textit{et}\; \mathsf{proj}_2(\langle x, y\rangle) \;=\; y.$$

Le symbole de fonction $\langle\cdot,\cdot\rangle$ modélise les paires, alors que les fonctions de projection sont notées proj_1 *et* proj_2. *On utilise le symbole unaire* hash *pour modéliser la fonction cryptographique de hachage classique. Un message haché par une fonction de hachage sure ne pouvant pas être retrouvé depuis le résultat de cette application, aucune équation n'est nécessaire dans la théorie équationnelle.*

Soit $m_A = \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle$ avec $info_A \in \mathcal{N}_{\mathsf{pub}}$ et $r_A \in \mathcal{N}_{\mathsf{guess}}$. On a que :

$$\mathsf{hash}(\langle \mathsf{proj}_1(m_A), r_A\rangle) =_{\mathsf{E}_{\mathsf{hash}}} \mathsf{hash}(\langle info_A, r_A\rangle)$$

$$\mathsf{proj}_2(m_A) =_{\mathsf{E}_{\mathsf{hash}}} \mathsf{hash}(\langle info_A, r_A\rangle)$$

$$\mathsf{hash}(\langle \mathsf{proj}_1(m_A), r_A\rangle) =_{\mathsf{E}_{\mathsf{hash}}} \mathsf{proj}_2(m_A).$$

1.2 Fonctions de hachage faibles

Des codes de vérification de petite taille, sous forme de courtes chaines de caractères, ont parfois besoin d'être transmis entre les participants, le plus couramment dans un but authentifiant. Une façon de générer ces codes est d'utiliser des fonctions de hachage *faibles*. Ces codes sont alors comparés ou recopiés par un participant humain. Cependant, le caractère énumérable des valeurs de retour d'une telle fonction de hachage permet à un attaquant de trouver efficacement des *collisions*. C'est-à-dire de trouver des paramètres différents sur lesquels va s'appliquer la fonction de hachage pour obtenir une même valeur de retour. Si la fonction s'applique sur plusieurs paramètres, on peut même considérer que l'attaquant peut choisir certains paramètres en itérant sur les paramètres restants jusqu'à trouver une collision.

Une façon de modéliser ces fonctions de hachage faibles est de représenter cette faiblesse dans la théorie équationnelle. Ce type d'approche a déjà été utilisé par Chothia *et al.* dans [CSS15] pour vérifier des protocoles d'engagement avec l'outil de vérification ProVerif. Alors que les fonctions de hachage classiques sont représentées uniquement par un symbole libre, le fait qu'une collision peut être trouvée efficacement par un attaquant sur une fonction de hachage faible à clé est modélisé par les équations suivantes :

$$\begin{aligned} \mathsf{sh}(\mathsf{bf}(k', \mathsf{sh}(k, m)), k') &= \mathsf{sh}(m, k) \\ \mathsf{sh}(m', \mathsf{bf}(m', \mathsf{sh}(k, m))) &= \mathsf{sh}(m, k) \end{aligned}$$

On considère une application de la fonction de hachage faible sh à une clé k et un message m, et on notera cette application $\mathsf{sh}(m, k)$. Dans la première équation, la notion de force brute modélisée par $\mathsf{bf}(k', \mathsf{sh}(m, k))$ représente le fait qu'un attaquant peut choisir une nouvelle clé pour sh, disons k'. Considérant cette nouvelle clé, il peut trouver efficacement, en essayant de nombreuses valeurs possibles pour le deuxième paramètre, une nouvelle valeur $r = \mathsf{bf}(k', \mathsf{sh}(m, k))$ telle qu'une collision se produit entre $\mathsf{sh}(m, k)$ et une nouvelle application de la fonction de hachage à k' et r, $\mathsf{sh}(r, k')$. Comme il y a collision nous obtenons que $\mathsf{sh}(r, k') = \mathsf{sh}(m, k)$, ce qui est exactement ce qui est modélisé par la première équation. De la même manière, dans la deuxième équation, on peut laisser l'attaquant choisir cette fois-ci le deuxième paramètre et utiliser la force brute pour calculer le premier paramètre et parvenir à une collision.

Exemple 4 *Dans l'exemple de protocole présenté en Figure 7 le message* $\mathsf{sh}(\langle info_A, k\rangle)$ *est transmis de A vers B. Cette fonction de hachage est faible, c'est pour le signifier que l'on représente ce terme par* $\mathsf{sh}(.,.)$*. Ainsi, l'attaquant pourrait trouver de nouveaux paramètres présentant une collision avec la précédente application. La théorie équationnelle l'autorise à trouver une telle collision s'il lui est possible de choisir un nouveau message* $info_I$ *et de construire un haché* $\mathsf{sh}(info_I, k')$ *où* k' *représente la valeur obtenue par l'attaquant en énumérant les valeurs possibles pour le deuxième terme de ce* sh *jusqu'à obtenir une collision, c'est-à-dire* $\mathsf{bf}(info_I, \mathsf{sh}(info_A, k))$*. Pour obtenir cette valeur* k' *et donc construire* $\mathsf{bf}(info_I, \mathsf{sh}(info_A, k))$ *l'attaquant doit pouvoir vérifier s'il a effectivement trouvé une collision et doit donc connaitre* $\mathsf{sh}(info_A, k)$*.*

Système de réécriture. Travailler modulo E est difficile, une façon plus opérationnelle de procéder est d'utiliser un système de réécriture. Un *système de réécriture* $\mathcal{R}$ est un ensemble de règles de la forme $\ell \to r$ où $\ell, r \in \mathcal{T}(\Sigma, \mathcal{X})$, et $\mathrm{vars}(r) \subseteq \mathrm{vars}(\ell)$. Un terme t peut être réécrit en u, noté $t \to_{\mathcal{R}} u$, s'il existe une position p dans t, une règle $\ell \to r$ dans $\mathcal{R}$ et une substitution σ telles que :

— $t|_p = \ell\sigma$; et
— $u = t[r\sigma]_p$, i.e. u est le terme obtenu en remplaçant $t|_p$ par $r\sigma$ dans t.

La relation $\to_{\mathcal{R}}^*$ représente la clôture transitive et réflexive de $\to_{\mathcal{R}}$. Un système de réécriture $\mathcal{R}$ est *convergent* s'il :

— est *confluent* : pour t, t_1, t_2 tels que $t \to_{\mathcal{R}}^* t_1$ et $t \to_{\mathcal{R}}^* t_2$, il existe u tel que $t_1 \to_{\mathcal{R}}^* u$ et $t_2 \to_{\mathcal{R}}^* u$; et
— *termine* : c'est-à-dire qu'il n'admet pas de séquence infinie $t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$

On note $t{\downarrow_{\mathcal{R}}}$ (ou plus simplement $t{\downarrow}$) la forme normale d'un terme t.

On considèrera uniquement dans la suite de ce document une théorie équationnelle E qui peut être représentée par un système de réécriture convergent $\mathcal{R}$, i.e. :

$$u =_{\mathsf{E}} v \;\Leftrightarrow\; u{\downarrow_{\mathcal{R}}} = v{\downarrow_{\mathcal{R}}}$$

Exemple 5 *En reprenant l'Exemple 3, on considère maintenant le système de réécriture $\mathcal{R}_{\mathsf{hash}}$ tel que :*

$$\mathsf{proj}_1(\langle x, y\rangle) \to x \qquad \mathsf{proj}_2(\langle x, y\rangle) \to y$$

$\mathcal{R}_{\mathsf{hash}}$ est un système de réécriture convergent et il représente la théorie équationnelle $\mathsf{E}_{\mathsf{hash}}$ définie dans l'Exemple 3.

1.3 Connaissance de l'attaquant

Notre attaquant va espionner tous les messages que s'échangent, sur des canaux publics, les participants d'un protocole. Ainsi, il va engranger de la connaissance qu'il va pouvoir utiliser ensuite pour construire de nouveaux messages. Cette connaissance est représentée par une séquence de messages $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{N})$ que l'on appelle une *frame*. On considère un deuxième ensemble infini de variables disjoint de $\mathcal{X}$, l'ensemble $\mathcal{W}$, tel que $\mathcal{W} = \{\mathsf{w}_1, \mathsf{w}_2, \ldots\}$. Une frame associe les termes $t_1, \ldots, t_n$ observés par l'attaquant à des variables de $\mathcal{W}$. Ainsi, une frame, nommons la ϕ, respecte le format suivant :

$$\phi = \{\mathsf{w}_1 \mapsto t_1, \ldots, \mathsf{w}_n \mapsto t_n\}.$$

Cette frame est donc une substitution avec comme *domaine* $\mathrm{dom}(\phi) = \{\mathsf{w}_1, \ldots, \mathsf{w}_n\}$, comme *image* $\mathrm{Im}(\phi) = \{\phi(x) \mid x \in \mathrm{dom}(\phi)\}$ et comme *taille* $|\phi| = n$.

Un attaquant peut déduire de nouveaux messages en appliquant des symboles de fonctions (dans Σ) à des noms publics (dans $\mathcal{N}_{\mathsf{pub}}$) et à des messages qu'il connait déjà (i.e. les termes dans $\mathrm{Im}(\phi)$).

Définition 3 (Déduction forte) *Soit ϕ une frame et $t \in \mathcal{T}(\Sigma, \mathcal{N})$.*

On dit que t est fortement déductible *de ϕ (par R), noté $\phi \vdash^{-}_{R} t$ quand $R\phi =_{\mathsf{E}} t$ pour $R \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathrm{dom}(\phi))$. Le terme R est appelé une* recette.

Exemple 6 *Considérons la frame suivante :*

$$\{\mathsf{w}_1 \mapsto \langle info_A, \mathsf{hash}(\langle info_A, \langle n_A, r_A\rangle\rangle)\rangle\}$$

qui est obtenue après l'envoie du premier message dans le protocole présenté en Figure 5. Elle est accompagnée de la théorie équationnelle $\mathsf{E}_{\mathsf{hash}}$. L'attaquant peut par exemple ici déduire fortement $info_A$, i.e. $\phi \vdash^{-}_{\mathsf{proj}_1(\mathsf{w}_1)} info_A$.

1.4 Noms devinables

Un attaquant peut deviner un nom s'il peut itérer sur toutes les valeurs possibles que peut prendre ce nom et s'il peut vérifier quand il a trouvé la valeur correcte. Dans la Section 1.1 nous avons introduit l'ensemble $\mathcal{N}_{\mathsf{guess}}$ contenant ces noms faibles que l'attaquant pourrait éventuellement deviner. Cependant, tout nom faible ne peut être

deviné. Il nous faut d'abord vérifier si l'attaquant peut distinguer quand il a réussi à deviner correctement de quand sa tentative échoue, et donc vérifier s'il a réussi à deviner la bonne valeur représentée par ce nom faible. Pour modéliser cette capacité de l'attaquant à deviner des noms faibles on va considérer qu'il a accès aux noms dans $\mathcal{N}_{\mathsf{guess}}$. Ainsi, l'attaquant va pouvoir construire davantage de termes pour construire des tests. Ces tests vont lui permettre de vérifier ses tentatives. La définition suivante permet d'exprimer sous quelles conditions on peut considérer qu'un attaquant peut deviner un nom faible. L'intuition est que si l'attaquant peut construire un test entre deux termes contenant au moins un nom dans $\mathcal{N}_{\mathsf{guess}}$, et que ce test n'est plus vrai quand ce nom faible est remplacé par un nom frais, alors l'attaquant a un moyen de vérifier quand il a trouvé la bonne valeur pour ce nom. C'est-à-dire que l'attaquant est en mesure de construire un test qui lui permet de distinguer quand il a effectivement trouvé la bonne valeur associée au nom faible.

Définition 4 (données devinables [CDK11, CDE05]) *Soit ϕ une frame et $g \in \mathcal{N}_{\mathsf{guess}}$. On dit que g est* devinable *de ϕ s'il existe $R_1, R_2 \in T(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathrm{dom}(\phi))$ tels que :*

- *$R_1\phi =_{\mathsf{E}} R_2\phi$, et*
- *$R'_1\phi \neq_{\mathsf{E}} R'_2\phi$ où $R'_i = R_i\{g \mapsto g'\}$ pour un nom frais g' ($i \in \{1, 2\}$).*

Ici nous considérons que les noms dans $\mathcal{N}_{\mathsf{guess}}$ peuvent être devinés ensemble et sont donc *conjointement* faibles. C'est un choix de modélisation qui pourrait conduire à de fausses attaques. Par exemple, considérons que dans un protocole les nonces considérés forts (donc non-devinables) ont une taille de 64 bits, et que ce protocole utilise deux nonces n_1 et n_2 qui eux sont considérés faibles et on une taille de 32 bits. Supposons que le message $\mathsf{hash}(\langle n_1, n_2\rangle)$ est envoyé sur le réseau. Dans notre modèle, l'attaquant va pouvoir deviner n_1 et n_2, pourtant la paire de n_1 et n_2 représente une concaténation classique et aurait ici une taille de 64 bits. Elle serait donc de taille suffisamment grande pour éviter qu'un attaquant puisse deviner ces nonces.

Exemple 7 *Soit $\phi = \{\mathsf{w}_1 \mapsto \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle\}$ avec $r_A \in \mathcal{N}_{\mathsf{guess}}$. Le nom r_A est devinable de ϕ. Intuitivement, comme l'attaquant connait $info_A$, il peut construire le terme $\mathsf{hash}(\langle info_A, _\rangle)$ et itérer sur toutes les valeurs possibles jusqu'à retrouver la valeur $\mathsf{hash}(\langle info_A, r_A\rangle)$ qu'il connait également. En considérant les recettes R_1, R_2 telles que $R_1 = \mathsf{proj}_2(\mathsf{w}_1)$ et $R_2 = \mathsf{hash}(\langle \mathsf{proj}_1(\mathsf{w}_1), r_A\rangle)$, nous avons que :*

- *$R_1\phi =_{\mathsf{E}_{\mathsf{hash}}} \mathsf{hash}(\langle info_A, r_A\rangle) =_{\mathsf{E}_{\mathsf{hash}}} R_2\phi$, alors que*
- *$(R_1\{r_A \mapsto r'_A\})\varphi \neq_{\mathsf{E}_{\mathsf{hash}}} R_2\{r_A \mapsto r'_A\}\varphi$.*

Cependant, on remarque que r_A n'est pas devinable de ϕ' :

$$\phi' = \{\mathsf{w}_1 \mapsto \langle info_A, \mathsf{hash}(\langle info_A, n_A, r_A\rangle)\rangle\}$$

où $n_A \in \mathcal{N}_{\mathsf{prv}}$.

On peut voir intuitivement que, sans connaitre n_A, il n'y a aucun moyen pour l'attaquant de vérifier s'il a deviné correctement la valeur r_A.

Chapitre 2

Modélisation des protocoles

Sommaire

2.1 Syntaxe de processus 23

2.2 Sémantique de processus 25

2.3 Modélisation des canaux hors-bande 26

2.4 Propriétés de sécurité 28

Dans le Chapitre 1, nous avons introduit comment représenter les messages que les participants d'un protocole vont s'échanger. Pour modéliser un protocole, nous avons besoin de décrire précisément les actions des différents rôles qui le composent. Ici, nous présentons comment modéliser les échanges de ces participants en présence d'un attaquant qui contrôle le réseau. Pour cela, nous décomposons ces échanges, distinguant émission et réception de messages et branchement conditionnel, tout en considérant l'exécution concurrente de ces actions afin de représenter toutes les interactions possibles. Nous modélisons ces échanges à l'aide d'une algèbre de processus qui est très proche du pi-calcul appliqué [AF01]. En particulier, celle-ci nous permet de modéliser les canaux hors-bande qui sont caractéristiques des protocoles manipulant des données à faible entropie. Nous adaptons également cette algèbre en ajoutant la notion d'évènements afin de vérifier aisément des propriétés d'authentification.

2.1 Syntaxe de processus

Les messages sont envoyés et reçus sur des canaux de communication. Ainsi nous définissons $\mathcal{C}h$ comme l'ensemble représentant ces canaux. Nous pouvons voir un protocole comme un processus clos où les différents participants agissent parallèlement comme des sous-processus. Ainsi, au sein d'un processus, on peut émettre et recevoir des messages, tester des conditions et exécuter des sous-processus. On modélise un protocole comme un processus en suivant la grammaire suivante :

P, Q	$::=$		
		$\mathbf{0}$	processus nul
.		$\mathbf{in}(c, x).P$	réception d'un message
		$\mathbf{out}(c, t).P$	émission d'un message
		$\mathbf{if}\ s = t\ \mathbf{then}\ P\ \mathbf{else}\ Q$	branchement conditionnel
		$P \| \| Q$	composition parallèle

FIGURE 2.1 – Syntaxe des processus

L'action de réception de message $\mathbf{in}(c, x)$ s'applique sur $c \in \mathcal{C}h$ et $x \in \mathcal{X}$. Elle modélise l'attente d'un message sur le canal c. Le processus $\mathbf{in}(c, x).P$ représente donc un agent en attente d'un message sur le canal c. À la réception d'un message m, celui-ci poursuivra son exécution comme indiqué par le processus P dans lequel la variable x sera remplacée par m. L'action d'émission de message $\mathbf{out}(c, t)$ considère un canal $c \in \mathcal{C}h$ et un terme $t \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ et modélise l'émission du message représenté par le terme t sur le canal c. Le branchement conditionnel $\mathbf{if}\ s = t\ \mathbf{then}\ P\ \mathbf{else}\ Q$ nous permet de modéliser une vérification d'égalité entre deux termes s et $t \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ effectuée par un des participants, selon le résultat du test on exécutera le processus P ou Q. La composition parallèle $P||Q$ nous permet de modéliser deux processus P et Q qui peuvent s'exécuter de façon concurrente, cela nous permettra par exemple de représenter plusieurs participants pouvant évoluer indépendamment.

On considère les définitions habituelles de variables *libres* et *liées* pour les processus. On considère également qu'une variable ne peut être liée qu'une seule fois. Cette condition est aisément satisfaite en renommant si besoin. Un processus est clos si et seulement s'il ne contient aucune variable libre. Dans un souci de concision, il nous arrivera d'omettre le processus nul. Aussi, nous noterons $\mathbf{if}\ s = t\ \mathbf{then}\ P$ au lieu de $\mathbf{if}\ s = t\ \mathbf{then}\ P\ \mathbf{else}\ \mathbf{0}$. De la même manière nous noterons $\mathbf{if}\ s \neq t\ \mathbf{then}\ P$ au lieu de $\mathbf{if}\ s = t\ \mathbf{then}\ \mathbf{0}\ \mathbf{else}\ P$.

Nous distinguons des échanges publics, qui pourront être lus et modifiés par l'attaquant dans la mesure de ses possibilités, et des échanges privés devant lesquels l'attaquant reste impuissant. Un échange est public si le canal de transmission est lui aussi public, c'est le type d'échange que l'on effectue via Internet par exemple. Les messages transitant sur de tels canaux peuvent être lus, interceptés et forgés par l'attaquant. Un échange est privé s'il a lieu sur un canal privé, un attaquant ne peut interagir avec un tel canal, en ce sens, l'attaquant n'a aucun pouvoir sinon de retarder l'échange. Nous définissons donc deux ensembles de canaux distincts, les canaux publics et les canaux privés, utilisés pour représenter les échanges publics ou privés de messages. Ainsi $\mathcal{C}h_{\mathsf{pub}}$ dénote l'ensemble des noms correspondants à des *canaux publics*. L'ensemble des noms correspondants à des *canaux privés* sera noté $\mathcal{C}h_{\mathsf{prv}}$. Naturellement nous avons que $\mathcal{C}h_{\mathsf{pub}} \uplus \mathcal{C}h_{\mathsf{prv}} = \mathcal{C}h$.

Exemple 8 *Reprenons le protocole affaibli présenté en Figure 4 :*

$$\begin{array}{llll} 1. & A & \rightarrow & B : \ info_A, \mathsf{hash}(info_A, r_A) \\ 2. & B & \rightarrow_e & A : \ \mathsf{ack} \\ 4. & A & \rightarrow_e & B : \ r_A \end{array}$$

avec la signature Σ_{hash} (Exemple 1) et la théorie équationnelle $\mathsf{E}_{\mathsf{hash}}$ (Exemple 3). Supposons également un canal public $c \in \mathcal{C}h_{\mathsf{pub}}$ et deux canaux privés $e_1, e_2 \in \mathcal{C}h_{priv}$, un

nom faible $r_A \in \mathcal{N}_{\mathsf{guess}}$ et une donnée $info_A \in \mathcal{N}_{\mathsf{pub}}$. En guise de simplification, nous encodons ici le canal hors-bande comme un canal privé, les différentes manières d'encoder les canaux hors-bande seront présentées ultérieurement dans la Section 2.3. Le protocole peut se modéliser de la façon suivante :

$$\begin{array}{rcl} P & = & A||B \\ A & = & \mathbf{out}(c, \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle).\mathbf{in}(e_2, x).\mathbf{out}(e_1, r_A) \\ B & = & \mathbf{in}(c, y).\mathbf{out}(e_2, \mathsf{ack}).\mathbf{in}(e_1.z).\mathbf{if}\ \mathsf{hash}(\langle \mathsf{proj}_1(y), z\rangle) = \mathsf{proj}_2(y)\ \mathbf{then}\ P' \end{array}$$

2.2 Sémantique de processus

Nous définissons maintenant la sémantique opérationnelle de notre algèbre de processus à l'aide d'une relation de transition étiquetée sur des configurations. Une *configuration* est une paire $(\mathcal{P}, \phi)$ où $\mathcal{P}$ est un multi-ensemble fini de processus clos, et ϕ est une frame utilisée pour stocker les messages que les participants ont précédemment envoyés sur des canaux publics, et qui sont donc connus de l'attaquant.

La sémantique est décrite par une relation dénotée $\xrightarrow{\ell}$ sur les configurations où $\ell \in \{\mathbf{out}(c), \mathbf{in}(c,t), \mathbf{test}^{=}, \mathbf{test}^{\neq}, \mathbf{guess}(g)\}$. Cette étiquette nous indiquera l'action qui a été exécutée.

Nul $\quad (\{\mathbf{0}\} \cup \mathcal{P}, \phi) \to (\mathcal{P}, \phi)$

Para $\quad (\{P||Q\} \cup \mathcal{P}, \phi) \to (\{P, Q\} \cup \mathcal{P}, \phi)$

Recept $\quad (\{\mathbf{in}(c, x).P\} \cup \mathcal{P}, \phi) \xrightarrow{\mathbf{in}(c,t)} (P\{x \mapsto t{\downarrow}\} \cup \mathcal{P}, \phi)$ si $\phi \vdash_R^- t$ et $c \in \mathcal{C}h_{\mathsf{pub}}$

Envoi $\quad (\{\mathbf{out}(c, t).P\} \cup \mathcal{P}, \phi) \xrightarrow{\mathbf{out}(c)} (\{P\} \cup \mathcal{P}, \phi \cup \{\mathsf{w}_{|\phi|+1} \mapsto t{\downarrow}\})$ si $c \in \mathcal{C}h_{\mathsf{pub}}$

ComPriv $\quad (\{\mathbf{in}(c, x).P, \mathbf{out}(c, t).Q\} \cup \mathcal{P}', \phi) \to (\{P\{x \mapsto t{\downarrow}\}, Q\} \cup \mathcal{P}, \phi)$ si $c \in \mathcal{C}h_{\mathsf{prv}}$

Test$^{=}$ $\quad (\{\mathbf{if}\ s = t\ \mathbf{then}\ P\ \mathbf{else}\ Q\} \cup \mathcal{P}', \phi) \xrightarrow{\mathbf{test}^{=}} (\{P\} \cup \mathcal{P}', \phi)$ si $s{\downarrow} = t{\downarrow}$

Test$^{\neq}$ $\quad (\{\mathbf{if}\ s = t\ \mathbf{then}\ P\ \mathbf{else}\ Q\} \cup \mathcal{P}', \phi) \xrightarrow{\mathbf{test}^{\neq}} (\{Q\} \cup \mathcal{P}', \phi)$ si $s{\downarrow} \neq t{\downarrow}$

Devine $\quad (\mathcal{P}, \phi) \xrightarrow{\mathbf{guess}(g)} (\mathcal{P}, \phi \cup \{\mathsf{w}_{|\phi|+1} \mapsto g\})$ si g est devinable de ϕ et $g \notin \mathrm{Im}(\phi)$

Figure 2.2 – Sémantique des processus

Dans la Figure 2.2, la règle Nul permet de terminer l'exécution d'un processus de $\mathcal{P}$ quand celui-ci n'a plus d'actions à exécuter. La règle Para distingue l'exécution de processus parallèle en les ajoutant à $\mathcal{P}$. La règle Recept modélise la réception d'un message : la variable, ici x, sera remplacée par le terme reçu dans les actions suivantes du processus. La règle Envoi représente l'envoi d'un message sur un canal public :

le message envoyé est ajouté à la frame. La règle ComPriv représente l'envoi d'un message sur un canal privé. Cet envoi considère une paire d'actions, réception et envoi, aucune information n'est ajoutée à la frame puisque, comme dans le cas de la règle Recept, nous allons substituer à la variable utilisée pour la réception le terme envoyé. Les règles Test$^{=}$ et Test$^{\neq}$ représente les deux cas possibles lors d'un branchement conditionnel. Enfin la règle Devine modélise le fait qu'un attaquant devine un nom faible, auquel cas l'attaquant peut l'ajouter à sa connaissance.

Exemple 9 *Considérons le processus P défini dans l'Exemple 8. Une exécution de la configuration $(\{P\}, \emptyset)$ pourrait se dérouler comme suit :*

$$
\begin{aligned}
(\{P\},\emptyset) \to & \quad (\{\mathbf{out}(c,M).A_1, \mathbf{in}(c,y).B_1\},\emptyset)\\
\xrightarrow{\mathbf{out}(c)} & \quad (\{A_1, \mathbf{in}(c,y).B_1\}, \{\mathsf{w}_1 \mapsto M\})\\
\xrightarrow{\mathbf{guess}(r_A)} & \quad (\{A_1, \mathbf{in}(c,y).B_1\}, \{\mathsf{w}_1 \mapsto M, \mathsf{w}_2 \mapsto r_A\})\\
\xrightarrow{\mathbf{in}(c,M)} & \quad \left(\begin{array}{l}\{\mathbf{in}(e_2,x).\mathbf{out}(e_1,r_A), B_1\{y \mapsto M\}\},\\ \{\mathsf{w}_1 \mapsto M, \mathsf{w}_2 \mapsto r_A\}\end{array}\right)\\
\to & \quad \left(\begin{array}{l}\left\{\begin{array}{l}\mathbf{out}(e_1,r_A), (\mathbf{in}(e_1.z).\\ \quad \mathbf{if}\ \mathsf{hash}(\langle \mathsf{proj}_1(M), z\rangle) = \mathsf{proj}_2(M)\ \mathbf{then}\ P')\end{array}\right\},\\ \{\mathsf{w}_1 \mapsto M, \mathsf{w}_2 \mapsto r_A\}\end{array}\right)\\
\to & \quad \left(\begin{array}{l}\{\mathbf{0}, (\mathbf{if}\ \mathsf{hash}(\langle \mathsf{proj}_1(M), r_A\rangle) = \mathsf{proj}_2(M)\ \mathbf{then}\ P')\},\\ \{\mathsf{w}_1 \mapsto M, \mathsf{w}_2 \mapsto r_A\}\end{array}\right)\\
\xrightarrow{\mathbf{test}^{=}} & \quad (\{\mathbf{0},\mathbf{0}\}, \{\mathsf{w}_1 \mapsto M, \mathsf{w}_2 \mapsto r_A\})\\
\to\to & \quad (\emptyset, \{\mathsf{w}_1 \mapsto M, \mathsf{w}_2 \mapsto r_A\})
\end{aligned}
$$

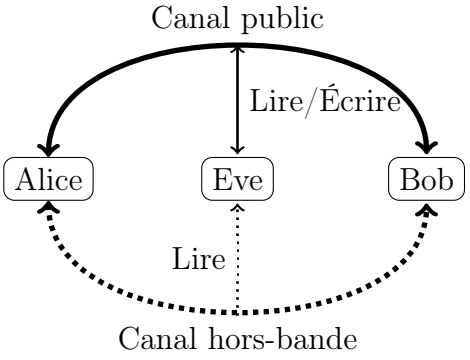
où :

- $M = \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle$;
- $A_1 = \mathbf{in}(e_2,x).\mathbf{out}(e_1,r_A)$; *et*
- $B_1 = (\mathbf{out}(e_2,\mathsf{ack}).\mathbf{in}(e_1.z).\mathbf{if}\ \mathsf{hash}(\langle \mathsf{proj}_1(y), z\rangle) = \mathsf{proj}_2(y)\ \mathbf{then}\ P')$.

2.3 Modélisation des canaux hors-bande

Le protocole *commitment before scheme* présenté dans [RSN12] ainsi que les protocoles que nous avons introduits dans les Figures 5 et 7 utilisent des canaux *empiriques*, que l'on considère comme étant authentiques et publics. Dans la pratique, ce sont des canaux hors-bande qui sont utilisés pour obtenir ces propriétés. L'utilisation des canaux hors-bande demande de faire intervenir un utilisateur humain comme receveur ou émetteur du message envoyé. Il va devoir par exemple écrire le message reçu ou l'information déduite sur son ordinateur ou son boitier dédié après une réception sur un canal hors-bande. Parfois, on lui demandera d'écrire sur un boitier dédié une courte chaine de caractère lue sur son ordinateur. Ces canaux peuvent prendre des formes

très diverses : un message envoyé sur le téléphone portable d'un des participants du protocole par un autre participant [3DS], un affichage que l'humain devra lire puis retranscrire sur un autre appareil [ISO10], [GMN04], [GRCC15] , ou encore deux affichages sur des boitiers que l'humain devra comparer [BTH01]. Ces canaux permettent une communication avec des propriétés de sécurité accrues, et jouent un rôle important pour empêcher des attaques exploitant des lacunes d'authentification des participants du type *homme du milieu*.



Eve, en tant qu'attaquant, ne peut qu'écouter sur le canal hors-bande ce qui permet à A et B d'échanger des messages authentifiés.

FIGURE 2.3 – Schéma d'un canal hors-bande

De par cette intervention humaine il est impératif de garantir que *les messages transmis sur ces canaux soient courts*. En effet, un humain aura sinon des difficultés à recopier une telle chaine de caractères sans erreurs. Il faut également considérer une autre contrainte liée à l'utilisation de ce type de canal. Une émission d'un message à l'attention d'un humain servant de vecteur entre les participants d'un protocole va être de nature *asynchrone*. La réception du message par le participant humain dépendra nécessairement de la réactivité de celui-ci et s'assurer que le message a effectivement été reçu demandera donc un échange supplémentaire, si besoin. Ces contraintes sont accompagnées d'avantages plus ou moins forts selon la forme exacte que ces canaux vont prendre en pratique. La plus faible hypothèse considérée généralement est qu'*un attaquant ne peut pas écrire* sur ce type de canal. Cette propriété d'authenticité est celle recherchée lorsque l'on met en place ce type de canal. Une schématisation d'un tel canal est présentée dans la Figure 2.3. On considère aussi parfois que ce type de canal permet de garantir la confidentialité des messages.

Afin de représenter l'envoi d'un message t sur un canal hors-bande, il nous faut prendre en compte les propriétés de ce canal. En effet, nous ne modéliserons pas de la même manière ces canaux suivant la forme qu'ils prennent en pratique. Considérons un canal privé $e \in \mathcal{C}h_{\mathsf{prv}}$, un canal public $c \in \mathcal{C}h_{\mathsf{pub}}$, et 2 processus concurrents $P||Q$ qui communiquent sur un canal hors-bande tels que P envoie un message t à Q sur ce canal. Nous présentons plusieurs encodages de canaux hors-bande.

Dans l'Exemple 8 nous avons utilisé un canal hors-bande authentique, asynchrone et confidentiel. C'est ce type de canaux qui est considéré dans le protocole 3D-Secure [3DS]. Nous codons ce canal hors-bande de la façon suivante :

$$\mathbf{out}(e,t)||P||\mathbf{in}(e,x).Q$$

Comme $e \in \mathcal{C}h_{\mathsf{prv}}$, t est transmis de manière confidentielle et authentique. Les canaux privés étant synchrones, $\mathbf{out}(e,t)$ s'exécutant de manière concurrente avec P permet d'exécuter des actions de P avant l'émission sur le canal privé, ainsi nous obtenons la propriété d'asynchronicité.

Dans les protocoles du standard [ISO10] les canaux hors-bande considérés sont authentiques et asynchrones mais pas nécessairement confidentiels. On modélisera ce type de canaux de la façon suivante :

$$\mathbf{out}(e,t)||\mathbf{out}(c,t).P||\mathbf{in}(e,x).Q$$

La différence ici est que l'on envoie le message t, à la fois sur le canal privé e de manière parallèle au reste de l'exécution du protocole pour obtenir comme précédemment les propriétés d'authenticité et d'asynchronicité, et sur le canal public afin de perdre la propriété de confidentialité que nous ne voulons pas ici.

Exemple 10 *Pour modéliser l'exemple de la Figure 4 :*

$$\begin{array}{llll} 1. & A & \rightarrow & B: \ info_A, \mathsf{hash}(info_A, r_A) \\ 2. & B & \rightarrow_e & A: \ \mathsf{ack} \\ 4. & A & \rightarrow_e & B: \ r_A \end{array}$$

on considérera deux canaux hors-bande authentiques et asynchrones (e_1, e_2). En suivant le codage précédent on obtient :

$$\begin{array}{lll} P & = & A||B \\ A & = & \mathbf{out}(c, \langle info_A, \mathsf{hash}(info_A, r_A)\rangle).\mathbf{in}(e_2, x).(\mathbf{out}(e_1, r_A)||\mathbf{out}(c, r_A)) \\ B & = & \mathbf{in}(c, y).(\mathbf{out}(e_2, \mathsf{ack})|| \\ & & \quad \mathbf{out}(c, \mathsf{ack}).\mathbf{in}(e_1.z).\mathbf{if}\ \mathsf{hash}(\mathsf{proj}_1(y), z) = \mathsf{proj}_2(y)\ \mathbf{then}\ 0) \end{array}$$

2.4 Propriétés de sécurité

Nous avons vu comment modéliser des protocoles et nous avons également donné une sémantique permettant de considérer un protocole s'exécutant dans un environnement hostile contrôlé par un attaquant. Nous définissons maintenant un langage qui permet de modéliser les propriétés de sécurité que le protocole doit satisfaire. Les protocoles que nous avons décrit en Figures 5 et 7 ont pour but d'authentifier l'échange d'une donnée $info_A$. La manière la plus courante pour exprimer des propriétés d'authentification est d'utiliser des relations de correspondance, comme fait dans [Low97],[Bla09]. Ces relations permettent en particulier de vérifier que lorsqu'un participant A termine une session en pensant avoir parlé à B, il y a bien une session joué par B avec le participant A qui lui correspond.

Afin d'utiliser ces relations de correspondance pour exprimer notre propriété de sécurité nous introduisons la notion d'évènements. L'ensemble $\Sigma\mathsf{v}$ contient les symboles d'évènements et les évènements sont des annotations de la forme $\mathsf{ev}(t)$ où $\mathsf{ev} \in \Sigma\mathsf{v}$ et $t \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$. L'ensemble des évènements est noté $\mathcal{E}v$. Nous étendons la syntaxe des processus (Figure 2.1) en y ajoutant la possibilité d'utiliser ces évènements :

$$\begin{array}{lll} P, Q & ::= & \\ & & \cdots \\ & & \mathsf{ev}(t).P \qquad \text{évènement} \\ & & \cdots \end{array}$$

où $\mathsf{ev}(t) \in \mathcal{E}v$. De la même façon, nous étendons la sémantique des processus en ajoutant à la sémantique classique (Figure 2.2) la règle :

$$\text{ÉVÈNEMENT} \quad (\{\mathsf{ev}(t).P\} \cup \mathcal{P}, \phi) \xrightarrow{\mathsf{ev}(t)} (P \cup \mathcal{P}, \phi) \text{ si } \mathsf{ev}(t) \in \mathcal{E}v$$

On notera $[\mathsf{ev}_2(x) \Rightarrow \mathsf{ev}_1(x)]$ la propriété de correspondance entre un évènement de fin de session $\mathsf{ev}_2(x) \in \mathcal{E}v$ et un évènement de début de session $\mathsf{ev}_1(x) \in \mathcal{E}v$, considérant x universellement quantifié.

Définition 5 *Soient P un processus et $\varphi = [\mathsf{ev}_2(x) \Rightarrow \mathsf{ev}_1(x)]$ une propriété de correspondance. Nous disons que P satisfait la propriété φ, noté $P \models \varphi$, si pour toute exécution $(\{P\}, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (\mathcal{P}, \phi)$ si il existe $i, 1 \leq i \leq n$, tel que $\ell_i = \mathsf{ev}_2(t_2)$ alors il existe j, $1 \leq j < i$ tel que $\ell_j = \mathsf{ev}_1(t_1)$, et $t_2{\downarrow} =_{\mathsf{E}} t_1{\downarrow}$.*

Nous définissons le problème de correspondance de la façon suivante :

Problème de correspondance

— *Entrée :* Un processus clos P, une propriété $\varphi = [\mathsf{ev}_2(x) \Rightarrow \mathsf{ev}_1(x)]$.

— *Sortie :* Est ce que $P \models \varphi$?

Exemple 11 *On peut écrire une relation de correspondance exprimant une propriété d'authentification sur le processus P décrit dans l'Exemple 8 de la manière suivante :*

$$\begin{aligned}
P &= A||B \\
A &= \mathbf{out}(c, \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle).\mathbf{in}(e_2, x).\mathbf{begin}(info_A). \\
&\qquad \mathbf{out}(e_1, r_A) \\
B &= \mathbf{in}(c, y).\mathbf{out}(e_2, \mathsf{ack}).\mathbf{in}(e_1.z). \\
&\qquad \mathbf{if}\ \mathsf{hash}(\mathsf{proj}_1(y), z) = \mathsf{proj}_2(y)\ \mathbf{then}\ \mathbf{end}(\mathsf{proj}_1(y))
\end{aligned}$$

Ici, on vérifiera si $P \models [\mathbf{end}(x)) \Rightarrow \mathbf{begin}(x)]$. C'est-à-dire que si B termine une session avec succès, alors A a commencé une session avec B, et A et B sont d'accord sur la donnée qui a été échangée.

On remarque dans l'Exemple 11 que le placement de l'évènement **begin** doit être choisi soigneusement parce que la propriété en dépend. On placera généralement l'évènement **begin** avant la dernière action d'émission de message du participant concerné.

Chapitre 3

Compilation vers un ensemble de traces

Sommaire

3.1 Syntaxe de traces 31
3.2 Sémantique de traces 33
3.3 Traduction depuis une algèbre de processus 34
3.4 Propriétés de sécurité sur les traces 37

Nous avons décrit dans le chapitre précédent comment représenter les protocoles, et les propriétés de sécurité que l'on veut vérifier, à l'aide d'une algèbre de processus. Pour des raisons algorithmiques, nous faisons le choix ici, comme dans [CcCK12], de nous ramener à une syntaxe de traces. Cette syntaxe est toute aussi expressive que la syntaxe de processus (Chapitre 2) et nous montrerons comment cette dernière se traduit dans cette nouvelle syntaxe de traces.

Nous introduirons tout d'abord la syntaxe et la sémantique de ces traces qui sont représentées par des séquences d'actions. Nous détaillerons ensuite la traduction vers un ensemble de traces de notre protocole et de la propriété de sécurité exprimés dans notre syntaxe de processus. Nous montrerons ensuite comment, vérifier des propriétés d'authentification dans ce contexte, revient à considérer un problème d'atteignabilité sur notre ensemble de traces.

3.1 Syntaxe de traces

Afin d'avoir une représentation mieux adaptée aux algorithmes que nous présenterons dans le Chapitre 4, nous introduisons ici une nouvelle syntaxe pour représenter les protocoles. Dans cette syntaxe de traces, l'entrelacement des différentes actions a déjà été fixé.

Une trace T est représentée par une séquence d'*actions* de l'ensemble des actions

noté $\mathcal{A}c$ et est générée par la grammaire suivante :

$$\begin{array}{rll} T = & \mathbf{in}(c,x).T & \text{réception d'un message} \\ & \mathbf{out}(c,t).T & \text{émission d'un message} \\ & [s=t].T & \text{tests positifs} \\ & [s\neq t].T & \text{tests négatifs} \\ & \mathbf{guess}(g).T & \text{deviner un nom faible} \\ & \mathbf{0} & \text{trace vide} \end{array}$$

où $c \in \mathcal{C}h_{\mathsf{prv}} \cup \mathcal{C}h_{\mathsf{pub}}$, $x \in \mathcal{X}$, $s,t \in \mathcal{T}(\Sigma, \mathcal{N} \uplus \mathcal{X})$ et $g \in \mathcal{N}_{\mathsf{guess}}$.

L'action $\mathbf{in}(c,x)$ représente la réception d'un message sur le canal c et agit, comme dans le chapitre précédent, telle une structure contraignante qui va appliquer une substitution sur la variable x. L'émission d'un message t sur un canal c est modélisée par $\mathbf{out}(c,t)$. Les tests d'égalité et de différence entre deux termes s et t ne sont plus représentés par un branchement conditionnel comme précédemment, mais sont respectivement représentés par les actions $[s=t]$ et $[s\neq t]$. L'action $\mathbf{guess}(g)$ détermine quand le nom g doit être deviné, c'est la seule action qui se place dans la perspective d'un attaquant, car c'est à l'attaquant que doit revenir le choix de deviner un nom, nous détaillons dans la Section 3.3 comment cette action est utilisée dans le cadre de la vérification de nos protocoles.

Un processus P sera représenté dans cette syntaxe comme un *ensemble fini de traces*, un ensemble de traces vide est noté $\emptyset$. Une trace peut être vue comme un des entrelacements possibles de l'enchainement des actions d'un protocole, ceci signifiant qu'une telle séquence d'actions peut être réalisée par le protocole jusqu'à un éventuel blocage. Une trace est dite *close* quand toutes ses variables sont liées par une action de réception de la forme $\mathbf{in}(c,x)$. Par soucis de concision, il nous arrivera d'omettre $\mathbf{0}$ (la trace vide) à la fin d'une trace. Avant de détailler la sémantique (Section 3.2) et comment nous traduisons un processus vers un ensemble de traces (Section 3.3), nous donnons ci-dessous un exemple d'une trace possible de notre protocole de la Figure 4.

Exemple 12 *Reprenons l'Exemple 8 :*

$$\begin{array}{rcl} P & = & A||B \\ A & = & \mathbf{out}(c, \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle).\mathbf{in}(e_2,x).\mathbf{out}(e_1,r_A) \\ B & = & \mathbf{in}(c,y).\mathbf{out}(e_2,\mathsf{ack}).\mathbf{in}(e_1.z).\mathbf{if}\ \mathsf{hash}(\langle \mathsf{proj}_1(y), z\rangle) = \mathsf{proj}_2(y)\ \mathbf{then}\ P' \end{array}$$

Ici on va prendre en compte une seule trace T_{weak} de P, c'est-à-dire une des séquences d'actions possibles représentant le processus P.

$$T_{\mathsf{weak}} \quad = \quad \mathbf{out}(c,m_A).\mathbf{guess}(r_A).\mathbf{in}(c,y).[\mathsf{hash}(\langle \mathsf{proj}_1(y), r_A\rangle) = \mathsf{proj}_2(y)]$$

où $m_A = \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle$, $c \in \mathcal{C}h$, $r_A \in \mathcal{N}_{\mathsf{guess}}$ et $y \in \mathcal{X}$. On suppose arbitrairement que le nom faible r_A doit être deviné juste après l'émission du message m_A. Cette trace représente bien un enchainement plausible des actions du processus P. En effet, dans l'Exemple 8 on a que $P = A||B$. La première action que peut faire A est $\mathbf{out}(c, \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle)$ cette action pourrait être suivie de $\mathbf{in}(c,y)$ pour B. On résout les deux envois sur un même canal privé, c'est-à-dire que pour chaque paire d'actions émission/réception nous remplaçons dans la trace les occurrences de la variable de l'action de réception par le terme de l'action d'émission. Il nous reste

à vérifier si $[\mathsf{hash}(\langle \mathsf{proj}_1(y), r_A\rangle) = \mathsf{proj}_2(y)]$*, notre trace considère ici uniquement le cas où ce test est vrai. Nous avons également ajouté une action* **guess** *qui elle aussi peut ne pas être exécutable, ceci étant dépendant de la sémantique. Ainsi, il est possible que le protocole ne puisse jamais attendre la fin de cette séquence d'actions, mais cette séquence reste un enchainement plausible.*

3.2 Sémantique de traces

Nous définissons maintenant la sémantique de notre algèbre de traces à l'aide d'une relation de transition étiquetée s'appliquant à des configurations. Une *configuration* est une paire (T, ϕ) où T est une trace close, et ϕ est une frame utilisée pour stocker les messages que les participants ont précédemment envoyés, et qui sont connus de l'attaquant.

Ainsi, nous définissons la relation $\xrightarrow{\ell}$ ci-dessous où l représente soit : une réception ou une émission de message, un test ou la tentative de deviner un nom faible.

$$\textsc{Recept} \quad (\mathbf{in}(c,x).T, \phi) \xrightarrow{\mathbf{in}(c,t)} (T\{x \mapsto t\downarrow\}, \phi) \quad \text{si } \phi \vdash^{+}_{R} t$$

$$\textsc{Envoi} \quad (\mathbf{out}(c,t).T, \phi) \xrightarrow{\mathbf{out}(c)} (T, \phi \cup \{\mathsf{w}_{|\phi|+1} \mapsto t\downarrow\})$$

$$\textsc{Test}^{=} \quad ([s = t].T, \phi) \xrightarrow{\mathbf{test}^{=}} (T, \phi) \quad \text{si } s\downarrow = t\downarrow$$

$$\textsc{Test}^{\neq} \quad ([s \neq t].T, \phi) \xrightarrow{\mathbf{test}^{\neq}} (T, \phi) \quad \text{si } s\downarrow \neq t\downarrow$$

$$\textsc{Devine} \quad (\mathbf{guess}(g).T, \phi) \xrightarrow{\mathbf{guess}(g)} (T, \phi \cup \{\mathsf{w}_{|\phi|+1} \mapsto g\}) \quad \text{si } g \text{ est devinable de } \phi \text{ et } g \notin \mathrm{Im}(\phi)$$

Cette sémantique cherche à modéliser les mêmes comportements que notre sémantique de processus du Chapitre 2, cependant, les tests et la capacité de l'attaquant à deviner des noms faibles sont ici décrits différemment pour épouser notre syntaxe de trace. Comme dans la sémantique de processus du Chapitre 2, l'étiquette $\mathbf{in}(c,t)$ représente la réception d'un message t envoyé par l'attaquant via le canal c et le label $\mathbf{out}(c)$ indique un message envoyé via le canal c, la règle de transition Envoi enregistre le message stocké dans la frame. Le branchement conditionnel étant aplani pour scier à une définition d'un protocole comme un ensemble de traces, deux règles de test sont à considérer. Les règles $\textsc{Test}^{=}$ et $\textsc{Test}^{\neq}$ vérifient respectivement l'égalité et la différence des termes s et t et sont étiquetées par $\mathbf{test}^{=}$ et $\mathbf{test}^{\neq}$. Quant à l'étiquette $\mathbf{guess}(g)$, elle exprime le fait que la valeur représentée par le nom g a été devinée, dans ce cas g est ajouté à la frame. Il est à noter que dans cette sémantique, l'action $\mathbf{guess}(g)$ doit être présente dans la trace pour que l'attaquant puisse deviner un nom faible g.

Exemple 13 *On peut exécuter la trace* T_{weak} *donnée dans l'Exemple 12 :*

$$T_{\mathsf{weak}} \;=\; \mathbf{out}(c, m_A).\mathbf{guess}(r_A).\mathbf{in}(c,y).[\mathsf{hash}(\langle \mathsf{proj}_1(y), r_A\rangle) = \mathsf{proj}_2(y)]$$

où $m_A = \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle$*, de la façon suivante :*

$$\begin{aligned}(T_{\mathsf{weak}}, \emptyset) &\xrightarrow{\mathbf{out}(c)}\xrightarrow{\mathsf{guess}(r_A)} && (\mathbf{in}(c,y).T'_{\mathsf{weak}}, \{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A\})\\ &\xrightarrow{\mathbf{in}(c,\langle info_A, \mathsf{hash}(\langle info_A, \mathsf{w}_2\rangle)\rangle)} && (T'_{\mathsf{weak}}, \{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A\})\\ &\xrightarrow{\mathbf{test}^{=}} && (\mathbf{0}, \{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A, \mathsf{w}_3 \mapsto r_A\})\end{aligned}$$

où $T'_{\mathsf{weak}} = [\mathsf{hash}(\langle \mathsf{proj}_1(y), r_A\rangle) = \mathsf{proj}_2(y)].\mathbf{0}$. *Le fait que* $\mathsf{guess}(r_A)$ *peut être exécuté est une conséquence directe du fait que* r_A *est devinable de* ϕ *(voir Exemple 7).*

3.3 Traduction depuis une algèbre de processus

Nous donnons ici l'intuition de la traduction depuis un processus comme spécifié par notre syntaxe de processus définie dans la Section 2.1 vers un ensemble de traces (Section 3.1).

Le processus nul et les actions d'émission et de réception sur des canaux publics se traduisent directement dans notre syntaxe minimaliste. Cependant l'émission et la réception sur un canal privé, le branchement conditionnel et la composition parallèle demandent d'être détaillés.

Pour exprimer la traduction vers un ensemble de traces nous définissons tout d'abord la fonction trad qui va traduire un ensemble de processus vers un ensemble de traces avec évènements. Les traces avec évènements sont des traces telles quelles ont été définies précédemment mais pour lesquelles nous autorisons les évènements comme actions valides dans la trace. Nous traduirons ensuite ces traces avec évènements vers nos traces classiques.

Pour simplifier les notations, définissons tout d'abord un opérateur séquençant une action, a, et un ensemble de traces, T. Ainsi nous avons :

$$a :: \mathsf{T} = \begin{cases} \{a.\mathbf{0}\} & \text{si } \mathsf{T} = \emptyset \\ \{a.T \mid \forall T \in \mathsf{T}\} & \text{sinon} \end{cases}$$

La forme normale d'un ensemble de processus $\mathcal{P}$, notée $\mathcal{P}{\downarrow}$, est obtenue en appliquant les règles Vide et Para autant que possible.

$$\begin{array}{ll} \textsc{Vide} & \mathcal{P} = \{\mathbf{0}\} \cup \mathcal{P}' \to \mathcal{P}' \\ \textsc{Para} & \mathcal{P} = \{P||Q\} \cup \mathcal{P}' \to \{P, Q\} \cup \mathcal{P}' \end{array}$$

Nous définissons trad, la fonction de traduction d'un ensemble de processus $\mathcal{P}$ vers un ensemble de traces avec évènements comme l'union des applications des fonctions trad_{int} et trad_{priv} sur la forme normale de $\mathcal{P}$. Nous avons donc que $\mathsf{trad}(\mathcal{P}) = \mathsf{trad}_{priv}(\mathcal{P}{\downarrow}) \cup \mathsf{trad}_{int}(\mathcal{P}{\downarrow})$, avec les définitions de trad_{priv} et de trad_{int} suivantes :

$$\mathsf{trad}_{priv}(\mathcal{P}) = \left\{ T \;\middle|\; \begin{array}{l} T \in \mathsf{trad}(\{P, Q\{x \mapsto t\}\} \cup \mathcal{P}'), e \in \mathcal{C}h_{\mathsf{prv}}, \\ \mathcal{P} = \{\mathbf{out}(e,t).P, \mathbf{in}(e,x).Q\} \cup \mathcal{P}' \end{array} \right\} \bigcup$$
$$\left\{ T \;\middle|\; \begin{array}{l} T \in \mathsf{trad}(\mathcal{P}'), e \in \mathcal{C}h_{\mathsf{prv}}, \\ \mathcal{P} = \mathbf{out}(e,t).P \cup \mathcal{P}' \text{ ou } \mathcal{P} = \mathbf{in}(e,x).Q \cup \mathcal{P}' \end{array} \right\}$$

$$\mathsf{trad}_{int}(\mathcal{P} = \{P_1, \ldots, P_n\}) = \left\{ T \;\middle|\; \begin{array}{l} T \in a_i :: \mathsf{trad}(\{P_1, \ldots, P'_i, \ldots, P_n\}), \\ P_i = a_i.P'_i, \\ \text{où } a_i \in \left\{ \mathbf{out}(c,t), \mathbf{in}(c,x), \mathsf{ev}(t) \right\}, \\ c \in \mathcal{C}h_{\mathsf{pub}} \end{array} \right\} \bigcup$$
$$\left\{ T \;\middle|\; \begin{array}{l} T \in [s = t] :: \mathsf{trad}(\{P_1, \ldots, P'_i, \ldots, P_n\}), \\ P_i = \mathbf{if}\ s = t\ \mathbf{then}\ P'_i\ \mathbf{else}\ Q \end{array} \right\} \bigcup$$
$$\left\{ T \;\middle|\; \begin{array}{l} T \in [s \neq t] :: \mathsf{trad}(\{P_1, \ldots, P'_i, \ldots, P_n\}), \\ P_i = \mathbf{if}\ s = t\ \mathbf{then}\ Q\ \mathbf{else}\ P'_i \end{array} \right\}$$

La fonction de traduction trad_{priv} va traduire les canaux privés vers notre syntaxe de trace. Le premier des 2 ensembles générés va considérer toutes les paires possibles d'émission/réception sur des canaux privés, ceci afin de générer les traces où cette paire émission/réception sera résolue i.e. on substituera la variable de l'action de réception par le terme émis. Le second ensemble considère le cas où l'on n'a pas réussi à émettre le message, dans ce cas une émission ou réception sur un canal privé est bloquante. La fonction trad_{int} s'intéresse à la traduction de deux autres cas : les entrelacements d'actions entre plusieurs processus concurrents, comme générés dans le premier des 3 ensembles, et la traduction des actions de test. Pour cela nous allons générer les traces qui vont considérer la branche *then* de la condition et les traces qui vont considérer la branche *else*.

Nous illustrons ci-dessous cette traduction à travers plusieurs exemples. Nous détaillons ensuite l'ajout d'actions **guess** dans nos traces.

Pour traduire l'émission/réception sur un canal privé e, il nous faut considérer toutes les paires possibles de $\mathbf{in}(e,x)/\mathbf{out}(e,t)$ et substituer la variable x par t dans le reste de la trace, c'est ce que fait la fonction de traduction trad_{priv}.

Exemple 14 *Soit un canal privé e et un canal public c.*

$$\begin{aligned} P &= \mathbf{in}(e,x).\mathbf{out}(c,\mathsf{hash}(x)) \\ Q &= \mathbf{out}(e,t) \\ A &= P||Q \end{aligned}$$

Après l'application de trad_{priv}, on obtient l'ensemble de traces Z tel que :

$$\begin{aligned} Z = \mathsf{trad}(\{A\}\downarrow) &= \mathsf{trad}_{int}(\{P,Q\}) \cup \mathsf{trad}_{priv}(\{P,Q\}) \\ &= \mathsf{trad}_{priv}(\{P,Q\}) \\ &= \mathsf{trad}(\{\mathbf{out}(c,\mathsf{hash}(t))\}) \cup \mathsf{trad}(\{Q\}) \cup \mathsf{trad}(\{P\}) \\ &= \{\mathbf{out}(c,\mathsf{hash}(t))\} \end{aligned}$$

Ceci parce que l'échange $\mathbf{in}(e,x)$ et $\mathbf{out}(e,t)$ a été résolu.

Un autre exemple plus complexe serait, considérant $c_1, c_2, c_3 \in \mathcal{C}h_{\mathsf{pub}}$:

$$\begin{aligned} P &= \mathbf{in}(e,x).\mathbf{out}(c_1,\mathsf{hash}(x)) \\ Q &= \mathbf{out}(e,t).\mathbf{out}(c_2,\mathsf{hash}(t)) \\ Q' &= \mathbf{out}(e,t').\mathbf{out}(c_3,\mathsf{hash}(t')) \end{aligned}$$

La traduction de $P||Q||Q'$ serait $Z = \mathsf{trad}_{priv}(\{P,Q,Q'\})$:

$$\begin{aligned} Z = \ \{ & \\ & \mathbf{out}(c_1,\mathsf{hash}(t)).\mathbf{out}(c_2,\mathsf{hash}(t)) \\ & \mathbf{out}(c_2,\mathsf{hash}(t)).\mathbf{out}(c_1,\mathsf{hash}(t)) \\ & \mathbf{out}(c_1,\mathsf{hash}(t')).\mathbf{out}(c_3,\mathsf{hash}(t')) \\ & \mathbf{out}(c_3,\mathsf{hash}(t')).\mathbf{out}(c_1,\mathsf{hash}(t')) \\ \} & \end{aligned}$$

Ici plusieurs résolutions de canaux sont possibles. Il nous faut les considérer toutes. Si un échange ne peut aboutir du fait de nos sélections de paires alors les actions suivantes du processus ne peuvent être exécutées. Les deux premières traces correspondent à l'appariement de P et Q tandis que les deux dernières correspondent à l'appariement de P et Q'.

La traduction du branchement conditionnel de la syntaxe de processus **if** $s = t$ **then** P **else** Q est fait par la fonction trad_{int}, cette traduction revient à considérer les deux traces $[s = t].P$ et $[s \neq t].Q$.

Exemple 15 *La traduction de* **if** $s = t$ **then out**(c, ok) **else out**(c, ko)*, notée Z sera la suivante :*

$$\begin{aligned} Z = \ \{ & \\ & [s = t].\mathbf{out}(c, \mathsf{ok}) \\ & [s \neq t].\mathbf{out}(c, \mathsf{ko}) \\ \} & \end{aligned}$$

La traduction de la composition parallèle $P||Q$ revient à représenter tous les entrelacements d'actions de la traduction de P et de Q, cette traduction est effectuée par la fonction trad_{int}.

Exemple 16

$$\begin{aligned} P &= \mathbf{out}(c, t).\mathbf{in}(c, x) \\ Q &= \mathbf{in}(c, y).\mathbf{out}(c, t') \end{aligned}$$

La traduction de $P||Q$, notée Z, sera la suivante :

$$\begin{aligned} Z = \ \{ & \\ & \mathbf{out}(c, t).\mathbf{in}(c, y).\mathbf{in}(c, x).\mathbf{out}(c, t') \\ & \mathbf{out}(c, t).\mathbf{in}(c, y).\mathbf{out}(c, t').\mathbf{in}(c, x) \\ & \mathbf{out}(c, t).\mathbf{in}(c, x).\mathbf{in}(c, y).\mathbf{out}(c, t') \\ & \mathbf{in}(c, y).\mathbf{out}(c, t).\mathbf{in}(c, x).\mathbf{out}(c, t') \\ & \mathbf{in}(c, y).\mathbf{out}(c, t).\mathbf{out}(c, t').\mathbf{in}(c, x) \\ & \mathbf{in}(c, y).\mathbf{out}(c, t').\mathbf{out}(c, t).\mathbf{in}(c, x) \\ \} & \end{aligned}$$

Les actions **guess** sont des actions qui représentent le moment où un attaquant doit deviner un nom faible. Évidemment un attaquant doit avoir la possibilité d'essayer de deviner un nom faible à tout moment. Nous devons donc considérer le cas où l'attaquant devine un nom faible, et celui où il ne devine pas de nom faible à chaque point de notre trace avec pour seule contrainte que l'attaquant ne va pas deviner un nom faible qu'il a déjà deviné. Nous définissons donc la fonction $\mathsf{posGuess}$ qui, appliquée sur un ensemble de traces avec évènements, T_{ev}, et un ensemble de noms faibles, $\mathcal{G}$, va générer l'ensemble de traces où toutes les possibilités de placement des actions **guess** sont considérées.

$$\mathsf{posGuess}(T_{\mathsf{ev}}, \mathcal{G}) = \begin{cases} \emptyset & \text{si } T_{\mathsf{ev}} = \mathbf{0} \\ \bigcup_{g \in \mathcal{G}} \mathbf{guess}(g) :: (\mathsf{posGuess}(T_{\mathsf{ev}}, \mathcal{G} \setminus \{g\}) & \\ \bigcup a :: \mathsf{posGuess}(T', \mathcal{G}) & \text{si } T_{\mathsf{ev}} = a.T' \end{cases}$$

Nous surchargeons $\mathsf{posGuess}$ pour considérer un ensemble de traces T_{ev} tel que :

$$\mathsf{posGuess}(\mathsf{T}_{\mathsf{ev}}, \mathcal{G}) = \bigcup_{T_{\mathsf{ev}} \in \mathsf{T}_{\mathsf{ev}}} \mathsf{posGuess}(T_{\mathsf{ev}}, \mathcal{G})$$

Afin d'illustrer cette définition nous proposons l'exemple ci-dessous.

Exemple 17 *Soit g_1, g_2 deux noms faibles, la trace $T = \mathbf{in}(c, x).\mathbf{out}(c, \mathsf{ok})$ devra être enrichie d'actions* **guess***, générant l'ensemble de traces suivant :*

$$
\begin{aligned}
P = \ \{ & \\
& \mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{guess}(g_1).\mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{guess}(g_1).\mathbf{guess}(g_2).\mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{guess}(g_1).\mathbf{in}(c,x).\mathbf{guess}(g_2).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{guess}(g_1).\mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}).\mathbf{guess}(g_2) \\
& \mathbf{in}(c,x).\mathbf{guess}(g_1).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{in}(c,x).\mathbf{guess}(g_1).\mathbf{guess}(g_2).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{in}(c,x).\mathbf{guess}(g_1).\mathbf{out}(c,\mathsf{ok}).\mathbf{guess}(g_2) \\
& \mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}).\mathbf{guess}(g_1) \\
& \mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}).\mathbf{guess}(g_1).\mathbf{guess}(g_2) \\
& \mathbf{guess}(g_2).\mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{guess}(g_2).\mathbf{guess}(g_1).\mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{guess}(g_2).\mathbf{in}(c,x).\mathbf{guess}(g_1).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{guess}(g_2).\mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}).\mathbf{guess}(g_1) \\
& \mathbf{in}(c,x).\mathbf{guess}(g_2).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{in}(c,x).\mathbf{guess}(g_2).\mathbf{guess}(g_1).\mathbf{out}(c,\mathsf{ok}) \\
& \mathbf{in}(c,x).\mathbf{guess}(g_2).\mathbf{out}(c,\mathsf{ok}).\mathbf{guess}(g_1) \\
& \mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}).\mathbf{guess}(g_2) \\
& \mathbf{in}(c,x).\mathbf{out}(c,\mathsf{ok}).\mathbf{guess}(g_2).\mathbf{guess}(g_1) \\
\} &
\end{aligned}
$$

L'explosion du nombre de traces quand on y ajoute les actions **guess** est conséquente, ainsi dans le Chapitre 5 nous mettons en exergue plusieurs méthodes d'optimisations qui réduisent drastiquement ce facteur. En effet, il n'est pas utile de considérer toutes ces positions de **guess**, certaines sont trivialement superflues, comme considérer à la fois $\mathbf{guess}(g_1).\mathbf{guess}(g_2)$ et $\mathbf{guess}(g_2).\mathbf{guess}(g_1)$. Les optimisations sur le placement des actions **guess** sont présentées dans le Chapitre 5.

Nous avons que si un processus P contient un ensemble de noms faibles $\mathcal{G}$ et est traduit vers un ensemble de traces T tel que $\mathsf{T} = \mathsf{posGuess}(\mathsf{trad}(\{P\}), \mathcal{G})$. alors toute exécution dans P menant vers une frame ϕ correspond à une exécution d'une trace de T, avec les mêmes étiquettes et menant à une même frame ϕ (Conjecture 1).

Conjecture 1 *Soit $\mathcal{P}$ un ensemble de processus, $\mathcal{G}$ l'ensemble des noms faibles utilisés dans $\mathcal{P}$. Nous avons que :*
$(\mathcal{P}, \phi) \xrightarrow{\ell_1,\ldots,\ell_n} (\mathcal{P}', \phi')$ *si et seulement si :*

$$
\begin{cases}
\exists T \in \mathsf{posGuess}(\mathsf{trad}(\mathcal{P}), \mathcal{G}) \\
\textit{tels que } (T, \phi) \xrightarrow{\ell_1,\ldots,\ell_n} (T', \phi').
\end{cases}
$$

3.4 Propriétés de sécurité sur les traces

Pour vérifier des propriétés d'authentification, nous utilisons des relations de correspondance. Dans ce chapitre nous montrons comment nous pouvons traduire un ensemble de traces avec évènements en un ensemble de traces sans évènements, comme

décrit dans la Section 3.1, depuis lequel on peut vérifier une propriété de correspondance. Pour cela nous allons nous ramener au problème d'atteignabilité.

Propriétés de correspondance : Il est intéressant, comme mentionné dans la Section 2.4, de représenter une propriété d'authentification par des propriétés de correspondance. Nous montrons ici que nous pouvons encoder une propriété de correspondance comme une propriété d'atteignabilité sur un ensemble de traces. Afin de traduire la représentation d'une propriété de correspondance dans le processus P, telle que nous voulons vérifier $P \models [\mathbf{end}(s) \Rightarrow \mathbf{begin}(s)]$ nous traduisons tout d'abord le processus vers un ensemble de traces avec évènements, cette syntaxe intermédiaire étend la syntaxe minimaliste présentée précédemment avec des évènements.

Exemple 18

$$\begin{aligned} P &= A||B \\ A &= \mathbf{begin}(\mathsf{ok}).\mathbf{out}(c, \mathsf{ok}) \\ B &= \in (c, x).\mathbf{end}(x) \end{aligned}$$

Le processus P sera traduit dans la syntaxe étendue par $Z = \mathsf{trad}(\{P\})$:

$$\begin{aligned} Z = \ \{ & \\ & \mathbf{begin}(\mathsf{ok}).\mathbf{out}(c, \mathsf{ok}).\mathbf{in}(c, x).\mathbf{end}(x) \\ & \mathbf{begin}(\mathsf{ok}).\mathbf{in}(c, x).\mathbf{out}(c, \mathsf{ok}).\mathbf{end}(x) \\ & \mathbf{begin}(\mathsf{ok}).\mathbf{in}(c, x).\mathbf{end}(x).\mathbf{out}(c, \mathsf{ok}) \\ & \mathbf{in}(c, x).\mathbf{begin}(\mathsf{ok}).\mathbf{out}(c, \mathsf{ok}).\mathbf{end}(x) \\ & \mathbf{in}(c, x).\mathbf{begin}(\mathsf{ok}).\mathbf{end}(x).\mathbf{out}(c, \mathsf{ok}) \\ & \mathbf{in}(c, x).\mathbf{end}(x).\mathbf{begin}(\mathsf{ok}).\mathbf{out}(c, \mathsf{ok}) \\ \} & \end{aligned}$$

Il n'y a pas d'évènement dans notre syntaxe minimaliste, ainsi une fois exprimé dans cette syntaxe avec évènements on traduira ceux-ci afin de se ramener à notre syntaxe sans évènements de façon à pouvoir vérifier une propriété d'atteignabilité. Ainsi, on vérifiera que ces traces traduites peuvent être exécutées jusqu'au bout, il s'agira bien sûr de considérer les traces qui permettent d'atteindre le point d'atteignabilité recherché.

Le problème d'atteignabilité pour une trace T dans notre sémantique est le suivant :

Problème d'atteignabilité d'une trace

— *Entrée :* Un trace close T

— *Sortie :* Existe-t-il $\ell_1, \ldots, \ell_n$, et une frame ϕ tels que $(T, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n} (\emptyset, \phi)$?

Si une telle exécution existe alors on dit que la trace T est *atteignable*.

Le problème d'atteignabilité, une fois exprimé sur une trace est très simplement généralisable à un ensemble de traces T, il nous suffit de vérifier si il existe une trace $T \in \mathsf{T}$ telle que T est atteignable.

Pour traduire ces évènements de correspondance nous allons ajouter des actions de test afin de vérifier si ces évènements sont effectivement en correspondance. C'est pourquoi, pour chaque **end**, nous allons générer une trace tronquée où l'on supprimera les actions après le **end** et les actions évènements. Nous ajouterons ensuite en fin de trace les tests qui permettent de vérifier la correspondance. Nous voulons vérifier que l'on ne peut atteindre l'évènement **end** que si un **begin** apparait avant dans la trace et correspond à ce **end**. Nous ajoutons donc en fin de trace, pour chaque $\mathbf{begin}(z)$ qui apparait avant un $\mathbf{end}(z')$, l'action $[z \neq z']$. Si une trace que l'on construit ainsi est atteignable alors elle viole notre propriété de correspondance.

Exemple 19 *Considérant la relation de correspondance* $[\mathbf{end}(z) \Rightarrow \mathbf{begin}(z)]$*, l'ensemble Z de l'Exemple 18 se traduira en Z' tel que :*

$$\begin{array}{rl} Z' = & \{ \\ & \quad \mathbf{out}(c, \mathsf{ok}).\mathbf{in}(c, x).[\mathsf{ok} \neq x] \\ & \quad \mathbf{in}(c, x).\mathbf{out}(c, \mathsf{ok}).[\mathsf{ok} \neq x] \\ & \quad \mathbf{in}(c, x).[\mathsf{ok} \neq x] \\ & \} \end{array}$$

On pourra résoudre le problème d'atteignabilité sur un tel ensemble de traces afin de décider de la correspondance. Si une trace de l'ensemble Z' est atteignable, alors cela signifie que la propriété de correspondance est violée.

Exemple 20 *Dans l'Exemple 11 nous exprimons une propriété d'authentification que nous voulons voir respectée dans notre protocole de la Figure 4.*

$$\begin{array}{rcl} P & = & A||B \\ A & = & \mathbf{out}(c, \langle info_A, \mathsf{hash}(\langle info_A, r_A \rangle) \rangle).\mathbf{in}(e_2, x).\mathbf{begin}(info_A). \\ & & \quad \mathbf{out}(e_1, r_A) \\ B & = & \mathbf{in}(c, y).\mathbf{out}(e_2, \mathsf{ack}).\mathbf{in}(e_1.z). \\ & & \quad \mathbf{if}\ \mathsf{hash}(\mathsf{proj}_1(y), z) = \mathsf{proj}_2(y)\ \mathbf{then}\ \mathbf{end}(\mathsf{proj}_1(y)) \end{array}$$

Nous traduisons maintenant ce processus, pour vérifier la propriété de correspondance $[\mathbf{end}(z) \Rightarrow \mathbf{begin}(z)]$*, vers un ensemble de traces, Z.*

$$\begin{array}{rl} Z = & \{ \\ & \quad T^1_{\mathsf{weak}} = \mathbf{out}(c, m_A).\mathbf{guess}(r_A).\mathbf{in}(c, x). \\ & \qquad [\mathsf{proj}_2(x) = \mathsf{hash}(\mathsf{proj}_1(x), r_A)].[info_A \neq \mathsf{proj}_1(x)], \\ & \quad T^2_{\mathsf{weak}} = \mathbf{guess}(r_A).\mathbf{out}(c, m_A).\mathbf{in}(c, x). \\ & \qquad [\mathsf{proj}_2(x) = \mathsf{hash}(\mathsf{proj}_1(x), r_A)].[info_A \neq \mathsf{proj}_1(x)], \\ & \quad T^3_{\mathsf{weak}} = \mathbf{out}(c, m_A).\mathbf{in}(c, x). \\ & \qquad [\mathsf{proj}_2(x) = \mathsf{hash}(\langle \mathsf{proj}_1(x), r_A \rangle)].[info_A \neq \mathsf{proj}_1(x)], \\ & \quad \cdots \\ & \} \end{array}$$

où $m_A = \langle info_A, \mathsf{hash}(\langle info_A, r_A \rangle) \rangle$.

Vérifier que le processus P de l'Exemple 11 satisfait cette propriété de correspondance, revient à vérifier si aucune des traces de Z n'est atteignable (Conjecture 2).

Nous notons $\mathsf{trad}_{corres}(T_{\mathsf{ev}}, [\mathsf{ev}_1(x) \Rightarrow \mathsf{ev}_2(x)])$ la traduction depuis une trace avec évènements T_{ev} vers un ensemble de traces sans évènements considérant la propriété de correspondance $[\mathsf{ev}_1(x) \Rightarrow \mathsf{ev}_2(x)]$.

$$\mathsf{trad}_{corres}(T_{\mathsf{ev}}, [\mathsf{ev}_1(x) \Rightarrow \mathsf{ev}_2(x)]) = \left\{ T \;\middle|\; \begin{array}{l} T_{\mathsf{ev}} = a_1.\cdots.a_n, a_i = \mathsf{ev}_1(u), \\ T = \mathsf{rm}\left(\begin{array}{l} a_1.\cdots.a_i \\ .[u \neq t_j]_{\{j | a_j = \mathsf{ev}_2(t_j)\}} \end{array} \right) \end{array} \right\}$$

où $[u \neq t_j]_{\{j|a_j=\mathsf{ev}_2(t_j)\}}$ représente les tests $[u \neq t_j]$ placés consécutivement en séquence et :

$$\mathsf{rm}(T_{\mathsf{ev}}) = \begin{cases} \mathbf{0} & \text{si } T_{\mathsf{ev}} = \mathbf{0} \\ \mathsf{rm}(T') & \text{si } T_{\mathsf{ev}} = a.T' \text{ et } a \in \mathcal{E}v \\ a.\mathsf{rm}(T') & \text{si } T_{\mathsf{ev}} = a.T' \text{ et } a \notin \mathcal{E}v \end{cases}$$

Pour un ensemble de traces T_{ev} nous définissons $\mathsf{trad}_{corres}(\mathsf{T}_{\mathsf{ev}})$ comme trad_{corres} appliquée à chacune des traces de T_{ev}.

Conjecture 2 *Soit une trace avec évènements T et une propriété de correspondance $[\mathsf{ev}_1(x) \Rightarrow \mathsf{ev}_2(x)]$. Nous avons que :*
T ne satisfait pas $[\mathsf{ev}_1(x) \Rightarrow \mathsf{ev}_2(x)]$ si et seulement si :
il existe une trace $T' \in \mathsf{trad}_{corres}(T, [\mathsf{ev}_1(x) \Rightarrow \mathsf{ev}_2(x)])$ tel que T' est atteignable.

Lemme 1 *Soit $\mathcal{P}$ un ensemble de processus, $\mathcal{G}$ l'ensemble des noms faibles utilisés dans $\mathcal{P}$. Nous avons que :*
$\mathcal{P} \not\models [\mathsf{ev}_1(x) \Rightarrow \mathsf{ev}_2(x)]$ si et seulement si :

$$\exists T \in \mathsf{trad}_{corres}(\mathsf{posGuess}(\mathsf{trad}(\mathcal{P}), \mathcal{G}), [\mathsf{ev}_1(x) \Rightarrow \mathsf{ev}_2(x)]) \text{ tel que } T \text{ est atteignable.}$$

Preuve. Ce résultat est une conséquence des Conjectures 1 et 2.

Ainsi nous pouvons traduire nos processus comme un ensemble de traces tout en conservant la validité de l'analyse des propriétés de correspondance. Dans le Chapitre 4 nous définissons donc une procédure de décision considérant chacune des traces de notre traduction.

Deuxième partie

Vérification

Chapitre 4

Procédure de décision

Sommaire

4.1 Propriété des variants finis 44
4.2 Modélisation des protocoles 45
4.2.1 Prédicats 45
4.2.2 Statements de la graine 48
4.2.3 Correction et complétude 50
4.2.4 Preuve de la correction et la complétude de la graine . . 51
4.3 Saturation 56
4.3.1 Procédure de saturation 56
4.3.2 Correction et complétude de la saturation 59
4.3.3 Preuve de la correction et de la complétude de la saturation 60
4.4 Procédure 65
4.4.1 Effectivité de la procédure de saturation 65
4.4.2 Description, correction et complétude de l'algorithme . . 66
4.4.3 Preuve de la correction et de la complétude de l'algorithme 67

Dans ce chapitre nous définissons formellement comment nous modélisons une trace sous forme de clauses de Horn puis comment nous appliquons notre procédure de décision afin de vérifier des propriétés d'atteignabilité. Tout d'abord nous introduisons une propriété que notre théorie équationnelle (modélisée comme un système de réécriture) doit satisfaire, la propriété des variants finis (Section 4.1). Ensuite plusieurs étapes se dégagent, nous construisons d'abord à partir de notre trace ce que nous appelons une graine, (Section 4.2), c'est-à-dire une représentation primaire de la trace sous forme de clauses de Horn. Ensuite nous appliquons une procédure de saturation (Section 4.3) afin de générer les clauses de Horn qui représentent tout ce que pourrait faire notre attaquant afin d'atteindre certains états sur cette trace, nous générons également les clauses qui nous permettrons plus tard de vérifier si les actions **guess** correspondent bien à un nom faible devinable à ce moment du protocole. Sur cette nouvelle base de connaissance saturée, nous appliquons notre algorithme (Section 4.4) afin de décider de propriétés d'atteignabilité, comme défini au Chapitre 3. Nous définissons donc ce modèle formel afin de représenter notre nouvel attaquant qui peut deviner des noms faibles. La construction de la graine est montrée correcte et complète (Sous-Section 4.2.3, Théorème 1). La procédure de saturation est elle aussi montrée correcte

et complète (Sous-Section 4.3.3,Théorème 2). Il en va de même pour notre algorithme qui va décider de l'atteignabilité d'une trace (Sous-Section 4.4.3, Théorème 3).

4.1 Propriété des variants finis

Pour représenter des primitives cryptographiques, nous utilisons des théories équationnelles. Une manière plus effective de travailler modulo une théorie équationnelle est d'utiliser un système de réécriture qui représente cette théorie (Section 1.2). Nous définissons maintenant la notion d'*ensemble complet de variants*, qui a été auparavant introduite dans [CD05].

Définition 6 *Considérant un système de réécriture convergent $\mathcal{R}$, et un ensemble de termes $\mathcal{T}$. Un ensemble de substitutions* $\text{variants}_{\mathcal{R}}(\mathcal{T})$ *est appelé un* ensemble complet de variants *pour l'ensemble de termes $\mathcal{T}$, si pour toute substitution ω il existe $\sigma \in \text{variants}_{\mathcal{R}}(\mathcal{T})$, et une substitution τ tel que : $x\omega\downarrow = x\sigma\downarrow\tau$ pour tout $x \in \text{vars}(\mathcal{T})$, et $(t\omega)\downarrow = (t\sigma)\downarrow\tau$ pour tout $t \in \mathcal{T}$.*

L'ensemble de variants d'un terme t représente un pré-calcul tel que la forme normale de toute instance de t est égale à une instance de $t\sigma\downarrow$ pour un σ dans l'ensemble de variants, sans avoir besoin d'appliquer davantage d'étapes de réécriture. Un système de réécriture a la *propriété de variant fini* si pour tout ensemble de termes, un ensemble complet de variants fini existe et est calculable. Nous écrirons le plus souvent $\text{variants}_{\mathcal{R}}(t_1, \ldots, t_n)$ au lieu de $\text{variants}_{\mathcal{R}}(\{t_1, \ldots, t_n\})$.

Exemple 21 *Appliquons ceci à $\mathcal{R}_{\mathsf{hash}}$ défini dans l'Exemple 5 :*

$$\mathcal{R}_{\mathsf{hash}} = \{\ \mathsf{proj}_1(\langle x, y\rangle) \to x \quad , \quad \mathsf{proj}_2(\langle x, y\rangle) \to y\ \}$$

Nous considérons deux ensembles de termes $\mathcal{T}_1 = \{\mathsf{proj}_1(y)\}$ et $\mathcal{T}_2 = \{\mathsf{proj}_2(y)\}$ et deux substitutions : σ_1 la substitution identité et $\sigma_2 = \{y \mapsto \langle y_1, y_2\rangle\}$. $\{\sigma_1, \sigma_2\}$ est un ensemble complet de variants pour $\mathcal{T}_1$ (resp. $\mathcal{T}_2$). En effet, pour toute substitution ω, nous avons que soit $\mathsf{proj}_i(y\omega\downarrow)$ est en forme normale, et donc σ_1 (ainsi que $\tau = \omega\downarrow$) satisfera la propriété, soit $y\omega\downarrow$ est en fait une paire $\langle m_1, m_2\rangle$, et la substitution σ_2 (ainsi que $\tau = \{y_1 \mapsto m_1, y_2 \mapsto m_2\}$) satisfera la propriété.

La propriété de variant fini est satisfaite par de nombreuses théories équationnelles intéressantes pour modéliser des protocoles cryptographiques, *e.g.* les chiffrements symétriques et asymétriques, les signatures, les signatures en aveugle ou les preuves à connaissance nulle. De plus, cette propriété implique l'existence d'un ensemble complet d'unificateurs et nous permet de le calculer efficacement [ESM10].

Définition 7 *Soit un système de réécriture convergent $\mathcal{R}$, et $\Gamma = \{u_1 = v_1, \ldots, u_k = v_k\}$ un ensemble d'équations. Un ensemble de substitutions Σ est appelé un* ensemble complet de $\mathcal{R}$-unificateurs *pour Γ si :*

1. *chaque $\sigma \in \Sigma$ est tel que $u_i\sigma\downarrow = v_i\sigma\downarrow$ pour $i \in \{1, \ldots, k\}$;*
2. *pour chaque θ tel que $u_i\theta\downarrow = v_i\theta\downarrow$ pour tout $i \in \{1, \ldots, k\}$, il existe $\sigma \in \Sigma$ et une substitution τ telle que $x\theta\downarrow = x\sigma\tau\downarrow$ pour tout $x \in \text{vars}(\Gamma)$.*

Nous notons $\mathrm{csu}_{\mathcal{R}}(\Gamma)$ *un tel ensemble.*

Quand $\mathcal{R} = \emptyset$, nous avons que pour tout $\Gamma = \{u_1 = v_1, \ldots, u_k = v_k\}$ qui admet une solution (i.e. il existe σ tel que pour tout $i \in \{1, \ldots, n\}$, $u_i\sigma = v_i\sigma$), un tel ensemble peut être choisi avec cardinalité 1. Cet élément est en fait unique à un renommage près et nous le notons $\mathrm{mgu}(\Gamma)$ ou $\mathrm{mgu}(u, v)$ quand $\Gamma = \{u = v\}$.

Dans la suite nous considérons des théories équationnelles E qui peuvent être représentées pas un système de réécriture convergent $\mathcal{R}$ qui respecte la propriété de variants finis.

4.2 Modélisation des protocoles

Comme dans la procédure originale décrite dans [CcCK12] et implémentée dans l'outil AKISS, notre procédure de décision est basée sur une modélisation pleinement abstraite d'une trace en clause de Horn. Nous donnons les détails de cette modélisation dans cette section.

4.2.1 Prédicats

Nous définissons un ensemble *d'exécutions symboliques*, notés $u, v, w, \ldots$, comme l'ensemble de séquence finies de labels symboliques.

$$u, v, w := \epsilon \mid l, w$$

où l est de la forme $\mathbf{in}(c, t)$, $\mathbf{out}(c)$, $\mathbf{test}^{=}$, $\mathbf{test}^{\neq}$, ou $\mathbf{guess}(g)$ avec $t \in \mathcal{T}(\Sigma, \mathcal{N} \uplus \mathcal{X})$, $c \in \mathcal{C}h$, et $g \in \mathcal{N}_{\mathsf{guess}}$.

La séquence vide est notée par ϵ. Intuitivement, une exécution symbolique représente un ensemble d'exécutions d'un protocole. On note $u \sqsubseteq v$ quand u est un préfixe de v.

Soit un ensemble $\mathcal{Y}$ de *variables de recettes* disjoint de $\mathcal{X}$, et nous utiliserons les lettres majuscules X, Y, Z pour représenter les variables de $\mathcal{Y}$. Nous supposons que de telles variables peuvent seulement être substituées par des termes dans $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathcal{W} \uplus \mathcal{Y})$.

Nous considérons 5 sortes de prédicats à partir desquels nous construisons les formules atomiques de notre logique. Avant de décrire ces prédicats, nous avons besoin de définir une variante de la sémantique opérationnelle.

Atteignabilité faible : Dans la Section 3.2 nous avons présenté la relation $\xrightarrow{\ell}$ pour introduire notre sémantique de traces. Pour représenter nos prédicats nous introduisons une nouvelle notation, une sémantique de traces affaiblie $\overset{\ell}{\twoheadrightarrow}$. La relation $\overset{\ell}{\twoheadrightarrow}$ est défini comme $\xrightarrow{\ell}$, excepté que les règles TEST$^{\neq}$ et GUESS sont remplacées par les deux règles suivantes :

$$(\mathbf{guess}(g).T, \phi) \xrightarrow{\mathbf{guess}(g)}\!\!\!\!\!\rightarrow\; (T, \phi \cup \{\mathsf{w}_{|\phi|+1} \mapsto g\})$$
$$([s \neq t].T, \phi) \xrightarrow{\mathbf{test}^{\neq}}\!\!\!\!\!\rightarrow\; (T, \phi)$$

Cette sémantique alternative ne vérifie pas si un nom est effectivement devinable ou si deux termes sont effectivement différents. Quand on parlera d'atteignabilité considérant cette sémantique, nous parlerons d'*atteignabilité faible*.

Nous définissons également une déduction plus faible que celle introduite dans la définition 3. Elle permet de construire des termes avec des noms faibles. Cela nous permet d'exprimer le fait que l'on peut construire des termes comme définis dans la définition des données devinables (Définition 4).

Définition 8 (Déduction faible) *Soit ϕ une frame et $t \in T(\Sigma, \mathcal{N})$.*

On dit que t est faiblement déductible *de ϕ (par R), noté $\phi \vdash^{+}_{R} t$ quand $R\phi =_{\mathsf{E}} t$ pour $R \in \mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathrm{dom}(\phi) \uplus \mathcal{N}_{\mathsf{weak}})$.*

Ci-dessous w représente une exécution symbolique, que nous appellerons un *monde*. R, R' sont des termes de $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathcal{W} \uplus \mathcal{Y})$ et t est un terme de $\mathcal{T}(\Sigma, \mathcal{N} \uplus \mathcal{X})$. La signification de ces prédicats est présentée ci-dessous. La sémantique formelle est détaillée dans la Figure 4.1.

Sémantique des prédicats :

- *Prédicat d'atteignabilité faible* : r_w est satisfait quand l'exécution w est exécutable dans la sémantique alternative $\overset{l}{\twoheadrightarrow}$, i.e., ignorant les contraintes sur les **guess** et les tests négatifs.
- *Prédicat de connaissance de l'attaquant* : $\mathsf{k}^{-}_{w}(R, t)$ est satisfait si quand l'exécution w est exécutable (toujours considérant $\twoheadrightarrow$), le message t peut être construit par l'attaquant utilisant la recette R qui ne contient pas de noms dans $\mathcal{N}_{\mathsf{guess}}$;
- *Prédicat de connaissance étendue* : $\mathsf{k}^{+}_{w}(R, t)$ est satisfait si quand l'exécution w est exécutable ($\twoheadrightarrow$), le message t peut être construit en utilisant la recette R qui peut contenir des noms dans $\mathcal{N}_{\mathsf{guess}}$.
- *Prédicat identité* : $\mathsf{i}_w(R, R')$ est satisfait si quand l'exécution w est exécutable ($\twoheadrightarrow$), R et R' (qui peuvent contenir des noms dans $\mathcal{N}_{\mathsf{guess}}$) sont des recettes pour le même terme ; et
- *Prédicat d'identité faiblement atteignable* : $\mathsf{ri}_w(\overrightarrow{R}, \overrightarrow{R'})$ est la forme courte de la conjonction $\mathsf{i}_w(R_1, R'_1), \ldots, \mathsf{i}_w(R_n, R'_n)$, et r_w, où $\overrightarrow{R} = (R_1, \ldots, R_n)$, et $\overrightarrow{R'} = (R'_1, \ldots, R'_n)$.

Une formule atomique (close) est interprétée considérant une paire contenant une trace T et une frame ϕ, et nous écrivons $(T, \phi) \models f$ quand la formule atomique f est satisfaite pour (T, ϕ) ou plus simplement $T \models f$ quand ϕ est la frame vide. Nous considérons les formules de premier ordre construites depuis les formules atomiques ci-dessus et les opérateurs usuels (conjonction, disjonction, négation, implication, et les quantificateurs universel et existentiel). La sémantique est définie comme on peut s'y attendre, mais le domaine des variables quantifiées dépend de leur type : les variables de $\mathcal{X}$ peuvent être remplacées par des termes de $\mathcal{T}(\Sigma, \mathcal{N})$, tandis que les variables de recettes de $\mathcal{Y}$ peuvent être remplacées par des recettes.

$$
\begin{array}{ll}
(T_0,\phi_0) \models \mathsf{r}_{l_1,\ldots,l_n} & \text{si } (T_0,\phi_0) \overset{l_1}{\twoheadrightarrow} (T_1,\phi_1) \overset{l_2}{\twoheadrightarrow} \ldots \overset{l_n}{\twoheadrightarrow} (T_n,\phi_n) \\
(T_0,\phi_0) \models \mathsf{k}^-_{l_1,\ldots,l_n}(R,t) & \text{si} \left| \begin{array}{l} \text{quand } (T_0,\phi_0) \overset{l_1}{\twoheadrightarrow} (T_1,\phi_1) \overset{l_2}{\twoheadrightarrow} \ldots \overset{l_n}{\twoheadrightarrow} (T_n,\phi_n) \\ \quad \text{alors } \phi_n \vdash^-_R t \end{array} \right. \\
(T_0,\phi_0) \models \mathsf{k}^+_{l_1,\ldots,l_n}(R,t) & \text{si} \left| \begin{array}{l} \text{quand } (T_0,\phi_0) \overset{l_1}{\twoheadrightarrow} (T_1,\phi_1) \overset{l_2}{\twoheadrightarrow} \ldots \overset{l_n}{\twoheadrightarrow} (T_n,\phi_n) \\ \quad \text{alors } \phi_n \vdash^+_R t \end{array} \right. \\
(T_0,\phi_0) \models \mathsf{i}_{l_1,\ldots,l_n}(R,R') & \text{s'il existe } t \text{ tel que } (T_0,\phi_0) \models \mathsf{k}^+_{l_1,\ldots,l_n}(R,t) \\
 & \quad \text{et } (T_0,\phi_0) \models \mathsf{k}^+_{l_1,\ldots,l_n}(R',t) \\
(T_0,\phi_0) \models \mathsf{ri}_{l_1,\ldots,l_n}(\overrightarrow{R},\overrightarrow{R'}) & \text{si } (T_0,\phi_0) \models \mathsf{r}_{l_1,\ldots,l_n} \text{ et pour tout } 1 \leq i \leq k, \\
 & \quad (T_0,\phi_0) \models \mathsf{i}_{l_1,\ldots,l_n}(R_i,R'_i) \\
 & \quad \text{où } \overrightarrow{R},\overrightarrow{R'} = (R_1,\ldots,R_k),(R'_1,\ldots,R'_k)
\end{array}
$$

FIGURE 4.1 – Sémantique des formules atomiques

Exemple 22 *Soit T^1_{weak}, une trace de Z tel que défini dans l'Exemple 20 :*

$T^1_{\mathsf{weak}} = \mathbf{out}(c,m_A).\mathbf{guess}(r_A).\mathbf{in}(c,x).[\mathsf{proj}_2(x) = \mathsf{hash}(\mathsf{proj}_1(x),r_A)].[info_A \neq \mathsf{proj}_1(x)]$

où $m_A = \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle$. Nous nous intéressons à une exécution possible de cette trace :

$$w_0 = \mathbf{out}(c),\mathbf{guess}(r_A),\mathbf{in}(c,m_I),\mathbf{out}(c),\mathbf{test}^=,\mathbf{test}^{\neq}$$

avec $m_I = \langle info_I, \mathsf{hash}(\langle info_I, r_A\rangle)\rangle$. Nous avons $T^1_{\mathsf{weak}} \models \mathsf{r}_{w_0}$, et c'est une conséquence directe de l'exécution suivante :

$$
\begin{array}{rl}
(T^1_{\mathsf{weak}},\emptyset) \xrightarrow{\mathbf{out}(c)} \xrightarrow{\mathbf{guess}(r_A)} & (\mathbf{in}(c,y).T'_{\mathsf{weak}},\{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A\}) \\
\xrightarrow{\mathbf{in}(c,m_I))} & (T'_{\mathsf{weak}},\{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A\}) \\
\xrightarrow{\mathbf{test}^=} \xrightarrow{\mathbf{test}^{\neq}} & (\mathbf{0},\{\mathsf{w}_1 \mapsto m_A, \mathsf{w}_2 \mapsto r_A, \mathsf{w}_3 \mapsto r_A\})
\end{array}
$$

Nous avons que $T^1_{\mathsf{weak}} \models \mathsf{i}_{\mathbf{out}(c)}(R_1,R_2)$ où les recettes R_1 et R_2 sont : $R_1 = \mathsf{proj}_2(\mathsf{w}_1)$ et $R_2 = \mathsf{hash}(\langle \mathsf{proj}_1(\mathsf{w}_1), r_A\rangle))$. En effet, la seule frame atteignable depuis T_{weak} depuis l'exécution $\mathbf{out}(c)$ est ϕ comme défini dans l'Exemple 7 :

$$\phi = \{\mathsf{w}_1 \mapsto \langle info_A, \mathsf{hash}(\langle info_A, r_A\rangle)\rangle\}$$

Soit $t = \mathsf{hash}(\langle info_A, r_A\rangle)$, nous avons que $\phi \vdash^+_{R_1} t$ (en fait nous avons $\phi \vdash^-_{R_1} t$) et $\phi \vdash^+_{R_2} t$.

Intéressons nous maintenant à la trace T^3_{weak} du même Exemple 20 :

$$T^3_{\mathsf{weak}} = \mathbf{out}(c,m_A).\mathbf{in}(c,x).[\mathsf{proj}_2(x) = \mathsf{hash}(\langle \mathsf{proj}_1(x), r_A\rangle)].[info_A \neq \mathsf{proj}_1(x)]$$

L'exécution suivante fait partie des exécutions possibles :

$$w'_0 = \mathbf{guess}(r_A),\mathbf{out}(c),\mathbf{in}(c,m'_I),\mathbf{out}(c),\mathbf{test}^=,\mathbf{test}^{\neq}$$

avec $m'_I = \langle info_I, \mathsf{hash}(\langle info_I, \langle n_I, r_A\rangle\rangle)\rangle$. Nous avons aussi que $T^2_{\mathsf{weak}} \models \mathsf{r}_{w'_0}$. Une telle trace est donc exécutable considérant la sémantique alternative $\twoheadrightarrow$, ceci bien que le nom faible r_A ne soit pas devinable à cet endroit de l'exécution.

4.2.2 Statements de la graine

Nous identifions maintenant un sous ensemble de formules, que nous appelons *statements*. Les statements prendrons la forme de clauses de Horn.

Définition 9 *Un* statement *est une clause de Horn de la forme :*

$$H \Leftarrow \mathsf{k}^{\star_1}_{u_1}(X_1, t_1), \ldots, \mathsf{k}^{\star_n}_{u_n}(X_n, t_n)$$

où :

- $H \in \{\mathsf{r}_{u_0},\ \mathsf{k}^{-}_{u_0}(R,t),\ \mathsf{k}^{+}_{u_0}(R,t),\ \mathsf{i}_{u_0}(R,R'),\ \mathsf{ri}_{u_0}(\overrightarrow{R},\overrightarrow{R'})\}$;
- $u_0, u_1, \ldots, u_n$ *sont des exécutions symboliques tels que* $u_i \sqsubseteq u_0$ *pour tout* $i \in \{1, \ldots, n\}$;
- $*_i \in \{-, +\}$
- $t, t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{N} \uplus \mathcal{X})$;
- R, R' *tout comme les éléments de* $\overrightarrow{R}$ *(resp.* $\overrightarrow{R'}$*) sont dans* $\mathcal{T}(\Sigma, \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}} \uplus \mathcal{W} \uplus \mathcal{Y})$;
- *et* $X_1, \ldots, X_n$ *sont des variables distinctes de* $\mathcal{Y}$.

Enfin, nous avons que $\mathrm{vars}(t) \subseteq \mathrm{vars}(t_1, \ldots, t_n)$ *quand* $H = \mathsf{k}^{\star}_{u_0}(R,t)$ *avec* $\star \in \{-,+\}$.

Dans la définition ci-dessus, nous supposons implicitement que toutes les variables sont universellement quantifiées, i.e. tous les statements sont clos. Par abus de langage nous appelons quelque fois σ une substitution close pour un statement $H \Leftarrow B_1, \ldots, B_n$ quand σ est close pour chacune des formules atomiques $H, B_1, \ldots, B_n$.

Comme mentionné ci-dessus, notre procédure de décision est basée sur une modélisation abstraite d'une trace vers des clauses de Horn. Dans cette section, étant donnée une trace close T, nous donnerons un ensemble de statements $\mathsf{seed}(T)$ qui servira de point de départ pour la modélisation. Nous devons aussi établir que l'ensemble de statements $\mathsf{seed}(T)$ est une abstraction correcte et (partiellement) complète de la trace close T. Afin de définir formellement $\mathsf{seed}(T)$, on commence par fixer quelques conventions.

Soit $T = \mathsf{a}_1.\mathsf{a}_2.\ldots.\mathsf{a}_n$ une trace close. On suppose sans perte de généralité la convention de nommage suivante :

1. si a_i est une action de réception alors $\mathsf{a}_i = \mathbf{in}(c_i, x_i)$.
2. si a_i est une action d'émission alors $\mathsf{a}_i = \mathbf{out}(c_i, t_i)$.
3. si a_i représente l'action de deviner alors $\mathsf{a}_i = \mathbf{guess}(g_i)$.
4. si a_i représente un test positif alors $\mathsf{a}_i = [s_i = t_i]$.
5. si a_i représente un test négatif alors $\mathsf{a}_i = [s_i \neq t_i]$.

De plus, on suppose que $x_i \neq x_j$ pour chaque $i \neq j$. Pour chaque $1 \leq i \leq n$, nous définissons le label symbolique ℓ_i comme suit :

$$\ell_i = \begin{cases} \mathbf{in}(c_i, x_i) & \text{si } \mathsf{a}_i = \mathbf{in}(c_i, x_i) \\ \mathbf{out}(c_i) & \text{si } \mathsf{a}_i = \mathbf{out}(c_i, t_i) \\ \mathbf{guess}(g_i) & \text{si } \mathsf{a}_i = \mathbf{guess}(g_i) \\ \mathbf{test}^{=} & \text{si } \mathsf{a}_i = [s_i = t_i] \\ \mathbf{test}^{\neq} & \text{si } \mathsf{a}_i = [s_i \neq t_i] \end{cases}$$

$$\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow} \Leftarrow \{\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j\in R(m)}$$
pour tout $0 \leq m \leq n$
pour tout $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^=(m)})$
pour tout $\tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_m\sigma)$

$$\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, t_m\sigma\tau\downarrow) \Leftarrow \{\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j\in R(m)}$$
pour tout $m \in S(n)$
pour tout $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^=(m)})$
pour tout $\tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_m\sigma, t_m\sigma)$

$$\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, g_m) \Leftarrow \{\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j\in R(m)}$$
pour tout $m \in G(n)$
pour tout $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^=(m)})$
pour tout $\tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_m\sigma)$

$$\mathsf{k}^-_{\ell_1,\ldots,\ell_m}(\mathsf{f}(Y_1, \ldots, Y_k), \mathsf{f}(y_1, \ldots, y_k)\tau\downarrow) \Leftarrow \{\mathsf{k}^-_{\ell_1,\ldots,\ell_m}(Y_j, y_j\tau\downarrow)\}_{j\in\{1,\ldots,k\}}$$
pour tout $0 \leq m \leq n$
pour tous les symboles de fonctions f d'arité k
pour tout $\tau \in \mathrm{variants}_{\mathcal{R}}(\mathsf{f}(y_1, \ldots, y_k))$.

$$\mathsf{k}^+_{\ell_1,\ldots,\ell_m}(\mathsf{f}(Y_1, \ldots, Y_k), \mathsf{f}(y_1, \ldots, y_k)\tau\downarrow) \Leftarrow \{\mathsf{k}^+_{\ell_1,\ldots,\ell_m}(Y_j, y_j\tau\downarrow)\}_{j\in\{1,\ldots,k\}}$$
pour tout $0 \leq m \leq n$
pour tous les symboles de fonctions f d'arité k
pour tout $\tau \in \mathrm{variants}_{\mathcal{R}}(\mathsf{f}(y_1, \ldots, y_k))$.

$$\mathsf{k}^-_{\epsilon}(c, c) \Leftarrow \quad \text{pour tout noms } c \in \mathcal{N}^0_{\mathsf{pub}}$$
$$\mathsf{k}^+_{\epsilon}(g, g) \Leftarrow \quad \text{pour tout noms } g \in \mathcal{N}^0_{\mathsf{guess}}$$

FIGURE 4.2 – l'ensemble $\mathsf{seed}(T, \mathcal{N}^0_{\mathsf{pub}}, \mathcal{N}^0_{\mathsf{guess}})$ des statements de la graine

Pour chaque $0 \leq m \leq n$, soit $R(m)$ (resp. $S(m)$, $G(m)$, $T^=(m)$, et $T^{\neq}(m)$) l'ensemble des indices correspondant à des actions de réception (resp. émission, **guess**, test positif et négatif) parmi $\mathsf{a}_1, \ldots, \mathsf{a}_m$. Aussi, nous notons $|S|$ la taille d'un tel ensemble S. Plus formellement nous avons que :

— $R(m) = \{\ i \ \mid\ 1 \leq i \leq m \text{ et } \mathsf{a}_i = \mathbf{in}(c_i, x_i)\ \}$;
— $S(m) = \{\ i \ \mid\ 1 \leq i \leq m \text{ et } \mathsf{a}_i = \mathbf{out}(c_i, t_i)\ \}$;
— $G(m) = \{\ i \ \mid\ 1 \leq i \leq m \text{ et } \mathsf{a}_i = \mathbf{guess}(g_i)\ \}$;
— $T^=(m) = \{\ i \ \mid\ 1 \leq i \leq m \text{ et } \mathsf{a}_i = [s_i = t_i]\ \}$;
— $T^{\neq}(m) = \{\ i \ \mid\ 1 \leq i \leq m \text{ et } \mathsf{a}_i = [s_i \neq t_i]\ \}$.

Soit un ensemble de noms publics $\mathcal{N}^0_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$, et un ensemble de noms devinables $\mathcal{N}^0_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$, *l'ensemble des statements de la graine* associés à T, $\mathcal{N}^0_{\mathsf{pub}}$, et $\mathcal{N}^0_{\mathsf{guess}}$ noté $\mathsf{seed}(T, \mathcal{N}^0_{\mathsf{pub}}, \mathcal{N}^0_{\mathsf{guess}})$, est défini comme l'ensemble des statements donné en Figure 4.2. Intuitivement, les 3 premiers types de statements de la graine nous permettent d'ex-

primer que dès que l'attaquant peut déduire des messages nécessaires pour passer des actions de réception qui ont lieu avant l'étape m, la trace est en effet exécutable (avec la sémantique faible) jusqu'à l'étape m. De plus, dans le cas où la $m^{\text{ème}}$ action est une émission ou une action **guess**, le terme correspondant deviendra déductible (fortement) par l'attaquant.

Si $(\mathcal{N}^0_{\mathsf{pub}}, \mathcal{N}^0_{\mathsf{guess}}) = (\mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{guess}})$, alors nous écrivons $\mathsf{seed}(T)$ comme une forme courte de $\mathsf{seed}(T, \mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{guess}})$. Cet ensemble est appelé *la graine associée à* T.

Exemple 23 *Intéressons nous toujours à la trace* T^1_{weak} *de l'Exemple 20. La graine construite à partir de cette trace,* $\mathsf{seed}(T^1_{\mathsf{weak}})$*, contiendra entre autres les statements suivants :*

$$
\begin{array}{ll}
f_1: & \mathsf{k}^-_{\mathbf{out}(c)}(\mathsf{w}_1, m_A) \Leftarrow \\
f_2: & \mathsf{k}^-_{\mathbf{out}(c).\mathbf{guess}(r_A)}(\mathsf{w}_2, r_A) \Leftarrow \\
f_3: & \mathsf{r}_w \Leftarrow \mathsf{k}^-_{\mathbf{out}(c).\mathbf{guess}(r_A)}(Y, \langle y_1, \mathsf{hash}(\langle y_1, r_A\rangle)\rangle) \\
f^+_4: & \mathsf{k}^+_{\mathbf{out}(c)}(\mathsf{hash}(X), \mathsf{hash}(x)) \Leftarrow \mathsf{k}^+_{\mathbf{out}(c)}(X, x) \\
f^-_5: & \mathsf{k}^-_{\mathbf{out}(c)}(\mathsf{proj}_2(Y), \mathsf{proj}_2(y)) \Leftarrow \mathsf{k}^-_{\mathbf{out}(c)}(Y, y) \\
f^-_6: & \mathsf{k}^-_{\mathbf{out}(c)}(\mathsf{proj}_2(Y), y_2) \Leftarrow \mathsf{k}^-_{\mathbf{out}(c)}(Y, \langle y_1, y_2\rangle)
\end{array}
$$

où $w = \mathbf{out}(c), \mathbf{guess}(r_A), \mathbf{in}(c, \langle y_1, \mathsf{hash}(\langle y_1, r_A\rangle)\rangle), \mathbf{out}(c), \mathbf{test}^=, \mathbf{test}^{\neq}$.

Le statement f_1 *représente le fait que* m_A *est fortement déductible depuis la recette* w_1 *une fois que le premier envoi a été effectué, en effet, l'attaquant écoute les messages qui transitent sur des canaux publics. La statement* f_2 *représente que* r_A *est fortement déductible une fois que les deux premières actions* $\mathbf{out}(c)$ *et* $\mathbf{guess}(r_A)$ *ont été effectuées, rappelons que la sémantique du prédicat de connaissance de l'attaquant est définie à l'aide de la relation* $\overset{\ell}{\twoheadrightarrow}$ *(Figure 4.1), les actions* **guess** *sont donc ici considérées au même titre qu'une action d'émission classique. Le statement* f_3 *signifie que la trace* T^1_{weak} *est (faiblement) exécutable, pour toute instance d'exécution de* w*, jusqu'à la dernière action de la trace, si un message de la forme* $\langle m, \mathsf{hash}(\langle m, r_A\rangle)\rangle$ *(pour un* m *quelconque) est déductible depuis l'information disponible à l'attaquant après l'exécution des deux premières actions, i.e. connaissant le premier envoi* m_A *et la valeur* r_A*. Des statements, e.g.,* f^+_4*,* f^-_5 *et* f^-_6*, représentent les possibilités de l'attaquant pour construire de nouveaux termes. Les deux derniers (*f^-_5 *et* f^-_6*) sont calculés en se basant sur* $\text{variants}_{\mathcal{R}}(\mathsf{proj}_2(y)) = \{\sigma_1, \sigma_2\}$ *(voir Exemple 21).*

4.2.3 Correction et complétude

Nous montrons brièvement que la graine est une modélisation correcte et (partiellement) complète d'une trace (Théorème 1). Cependant, nous avons besoin d'une définition supplémentaire pour étayer ce fait.

Définition 10 *Soit un ensemble K de statements, $\mathcal{H}(K)$ est le plus petit ensemble de faits clos tel que :*

$$\textsc{Conseq}\ \frac{\sigma\ \textit{clos},\ f = \Big(H \Leftarrow B_1, \ldots, B_n\Big) \in K \qquad B_1\sigma \in \mathcal{H}(K),\ \ldots,\ B_n\sigma \in \mathcal{H}(K)}{H\sigma \in \mathcal{H}(K)}$$

$$\textsc{Extend}\ \frac{\mathsf{k}^{\star}_{u}(R,t) \in \mathcal{H}(K)\ \textit{avec} \star \in \{-,+\}}{\mathsf{k}^{\star}_{uv}(R,t) \in \mathcal{H}(K)}$$

$$\textsc{Weakening}\ \frac{\mathsf{k}^{-}_{u}(R,t) \in \mathcal{H}(K)}{\mathsf{k}^{+}_{u}(R,t) \in \mathcal{H}(K)}$$

Nous montrons qu'en considérant uniquement ce que représentent les prédicats d'atteignabilité faible et de connaissance de l'attaquant, l'ensemble $\mathsf{seed}(T)$ est une abstraction complète d'une trace.

Théorème 1 *Soit T une trace close.*

- *Correction : $T \models f$ pour chaque $f \in \mathsf{seed}(T) \cup \mathcal{H}(\mathsf{seed}(T))$;*
- *Complétude : Si $(T, \emptyset) \xrightarrow{l_1,\ldots,l_m}\!\!\!\!\!\rightarrow\ (Q, \phi)$ alors*
  1. *$\mathsf{r}_{l_1\downarrow,\ldots,l_m\downarrow} \in \mathcal{H}(\mathsf{seed}(T))$;*
  2. *si $\phi \vdash^{\star}_{R} t$ avec $\star \in \{-,+\}$ alors $\mathsf{k}^{\star}_{l_1\downarrow,\ldots,l_m\downarrow}(R, t\downarrow) \in \mathcal{H}(\mathsf{seed}(T))$.*

Nous montrerons comment la complétude de $\mathsf{seed}(T)$ peut être construite pour atteindre une abstraction complète, i.e., incluant aussi les identités d'une trace T et une procédure pour vérifier l'atteignabilité.

4.2.4 Preuve de la correction et la complétude de la graine

Dans cette section nous donnons la preuve du Théorème 1. La partie de correction de la graine est une conséquence du Lemme 2 et de la Proposition 1. Ainsi, le Lemme 2 montre que considérant une trace T, tout statement de $\mathsf{seed}(T)$ est correct. La Proposition 1 nous dit que, si un ensemble de statement K est correct, alors les statements de $\mathcal{H}(K)$ sont corrects. La partie complétude est quant à elle démontrée par la preuve du Lemme 3.

Le Lemme 2 nous démontre que tous les statements de la graine sont corrects.

Lemme 2 *Soit T une trace close.*
Nous avons que $T \models f$ pour tout statement $f \in \mathsf{seed}(T)$.

Preuve. Supposons la convention de nommage pour T comme dans la définition de la graine. Nous prouvons que $T \models f$ pour tout statement $f \in \mathsf{seed}(T)$ en procédant par étude de cas sur les 7 types de statements dans la graine.

<u>*Cas 1.*</u> Soit m tel que $0 \leq m \leq n$, $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k \in T^{=}(m)})$,
et $\tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_m\sigma)$. Soit f le statement suivant :

$$\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow} \Leftarrow \{\mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j \in R(m)}$$

Soit ω une substitution arbitraire qui clôt f. De plus, supposons que pour tout $j \in R(m)$, $T \models (\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow))\omega$. Il faut montrer que $T \models (\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow})\omega$. Pour cela nous allons montrer une propriété plus forte. Nous montrons que :

$$T \models (\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_p\sigma\tau\downarrow})\omega \text{ pour tout } 0 \leq p \leq m.$$

Nous procédons par induction sur p. Dans le cas de base, quand $p = 0$ nous avons que $(\mathsf{r}_{\ell_1\sigma\tau\downarrow,\cdots,\ell_p\sigma\tau\downarrow})\omega = \mathsf{r}$ et c'est pourquoi nous avons trivialement que $T \models (\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_p\sigma\tau\downarrow})\omega$. Quand $p > 0$, nous supposons que $T \models (\mathsf{r}_{\ell_1\sigma\tau\downarrow,\cdots,\ell_{p-1}\sigma\tau\downarrow})\omega$, et nous montrons que $T \models (\mathsf{r}_{\ell_1\sigma\tau\downarrow,\cdots,\ell_p\sigma\tau\downarrow})\omega$ par étude de cas sur a_p. Avant de faire cette étude de cas nous fixons quelques notations. Soit $T_1 = T$ et $\varphi_1 = \emptyset$. Comme $T \models (\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{p-1}\sigma\tau\downarrow})\omega$, nous avons que $(T_i, \varphi_i) \xrightarrow{\ell_i\sigma\tau\downarrow\omega}\!\!\!\!\rightarrow (T_{i+1}, \varphi_{i+1})$ et pour tout $1 \leq i < p$, où $T_i = (\mathsf{a}_i, \ldots, \mathsf{a}_n)\{x_j \mapsto x_j\sigma\tau\downarrow\omega\downarrow\}_{j\in R(i-1)}$ et où φ_{i+1} étend φ_i (pour tout $1 \leq i < p$). Nous pouvons maintenant procéder à l'étude de cas.

1. Si $\mathsf{a}_p = \mathbf{out}(c_p, t_p)$, alors $\ell_p = \mathbf{out}(c_p)$.
   Soit $T_{p+1} = (\mathsf{a}_{p+1}, \ldots, \mathsf{a}_n)\{x_j \mapsto x_j\sigma\tau\downarrow\omega\downarrow\}_{j\in R(p)}$ et $\varphi_{p+1} = \varphi_p \cup \{\mathsf{w}_{|\varphi_p|+1} \mapsto t_p\sigma\tau\downarrow\omega\downarrow\}$. Nous avons que $(T_p, \varphi_p) \xrightarrow{\ell_p}\!\!\!\!\rightarrow (T_{p+1}, \varphi_{p+1})$, ce qu'il nous fallait démontrer.
2. Si $\mathsf{a}_p = \mathbf{guess}(g_p)$ alors $\ell_p = \mathbf{guess}(g_p)$.
   Soit $T_{p+1} = (\mathsf{a}_{p+1} \ldots \mathsf{a}_n)\{x_j \mapsto x_j\sigma\tau\downarrow\omega\downarrow\}_{j\in R(p)}$ et $\varphi_{p+1} = \varphi_p \cup \{\mathsf{w}_{|\varphi_p|+1} \mapsto g_p\}$. Nous avons, comme dans le cas précédent, que $(T_p, \varphi_p) \xrightarrow{\ell_p}\!\!\!\!\rightarrow (T_{p+1}, \varphi_{p+1})$, ce que nous voulions démontrer.
3. Si $\mathsf{a}_p = \mathbf{in}(c_p, x_p)$ alors $\ell_p = \mathbf{in}(c_p, x_p\sigma\tau\downarrow\omega)$ et $p \in R(p)$.
   Soit $T_{p+1} = (\mathsf{a}_{p+1} \ldots \mathsf{a}_n)\{x_j \mapsto x_j\sigma\tau\downarrow\omega\downarrow\}_{j\in R(p)}$ et $\varphi_{p+1} = \varphi_p$. Comme $p \in R(p)$, nous avons que $T \models \mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{p-1}\sigma\tau\downarrow}(X_p, x_p\sigma\tau\downarrow)\omega$ (c'est un antécédent de f). C'est pourquoi, nous avons que $\varphi_p \vdash^-_{X_p\omega} x_p\sigma\tau\downarrow\omega$. Nous obtenons par la définition de $\twoheadrightarrow$ que $(T_p, \varphi_p) \xrightarrow{\ell_p}\!\!\!\!\rightarrow (T_{p+1}, \varphi_{p+1})$, qui est ce que nous voulions montrer.
4. Si $\mathsf{a}_p = [s_p = t_p]$, alors $\ell_p = \mathbf{test}^=$.
   Soit $T_{p+1} = (\mathsf{a}_{p+1} \ldots, \mathsf{a}_n)\{x_j \mapsto x_j\sigma\tau\downarrow\omega\downarrow\}_{j\in R(p)}$, et $\varphi_{p+1} = \varphi_p$. Comme $\sigma \in \mathrm{csu}_{\mathcal{R}}\{s_p = t_p\}_{k\in T^=(m)}$, nous avons que $s_p\sigma\downarrow = t_p\sigma\downarrow$, et c'est pourquoi $s_p\sigma\tau\downarrow\omega = t_p\sigma\tau\downarrow\omega$. Nous avons que $(T_p, \varphi_p) \xrightarrow{\ell_p}\!\!\!\!\rightarrow (T_{p+1}, \varphi_{p+1})$, ce que nous voulions prouver.
5. Si $\mathsf{a}_p = [s_p \neq t_p]$, alors $\ell_p = \mathbf{test}^{\neq}$. Soit $T_{p+1} = (\mathsf{a}_{p+1} \ldots \mathsf{a}_n)\{x_j \mapsto x_j\sigma\tau\downarrow\omega\downarrow\}_{j\in R(p)}$, et $\varphi_{p+1} = \varphi_p$. Nous avons que $(T_p, \varphi_p) \xrightarrow{\ell_p}\!\!\!\!\rightarrow (T_{p+1}, \varphi_{p+1})$, ce que nous voulions montrer.

Nous avons montré que $T \models (\mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_p\sigma\tau\downarrow})\omega$.

<u>*Cas 2.*</u> Soit $m \in S(n)$, une substitution $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^=(m)})$, $\tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_m\sigma, t_m\sigma)$, et f un statement comme suit :

$$\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, t_m\sigma\tau\downarrow) \Leftarrow \{\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j\in R(m)}.$$

Nous montrons que $T \models f$. Soit ω une substitution qui clôt f. Nous supposons que :

$$T \models \mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\omega \text{ pour tout } j \in R(m).$$

Nous montrons que $T \models (\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, t_m\sigma\tau\downarrow))\omega$. Soit $o(y)$ le terme envoyé où deviné à la $y^{\text{ème}}$ action d'émission ou action **guess**, i.e. $o(y) = t_j$ quand $j = min\{x \mid |S(x)| + |G(x)| = y\}$ appartient à $S(n)$, ou $o(y) = g_j$ sinon (i.e. quand $j \in G(n)$). Soit $T_i = (\mathsf{a}_i \ldots \mathsf{a}_n)\{x_j \mapsto x_j\sigma\tau\downarrow\omega\downarrow\}_{j\in R(i-1)}$ et $\varphi_i = \bigcup_{1\leq j<|S(i)|+|G(i)|}\{\mathsf{w}_j \mapsto o(j)\sigma\tau\downarrow\omega\downarrow\}$ pour $1 \leq i \leq m+1$.
Si

$$(T_1, \varphi_1) \xrightarrow{\ell_1\sigma\tau\downarrow\omega} (T_2, \varphi_2) \cdots \xrightarrow{\ell_m\sigma\tau\downarrow\omega} (T_{m+1}, \varphi_{m+1})$$

alors

$$\varphi_{m+1}(\mathsf{w}_{|S(m)|+|G(m)|}) = o(m)\sigma\tau\downarrow\omega\downarrow = t_m\sigma\tau\downarrow\omega\downarrow$$

et d'où $\varphi \vdash^-_{\mathsf{w}_{|S(m)|+|G(m)|}} t_m\sigma\tau\omega\downarrow$. C'est pourquoi, nous avons que $\varphi \vdash^-_{\mathsf{w}_{|S(m)|+|G(m)|}} (t_m\sigma\tau)\downarrow\omega$ ce qui implique que $T \models (\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\cdots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, t_m\sigma\tau\downarrow))\omega$.

Cas 3. Soit $m \in G(n)$, $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^=(m)})$.
Soit f un statement de la forme :

$$\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\cdots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, g_m) \Leftarrow \{\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_j\sigma\tau\downarrow}(X_j, x_j\sigma\downarrow)\}_{j\in R(m)}.$$

Nous montrons que $T \models f$. Soit ω une substitution qui clôt f. Nous supposons que :

$$T \models \mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\downarrow)\omega \text{ pour tout } j \in R(m)$$

Nous montrons que $T \models (\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, g_m)\omega$. Nous considérons $o(y)$ tel que défini dans le cas précédent. Soit $T_i = (\mathsf{a}_i \ldots \mathsf{a}_n)\{x_j \mapsto x_j\sigma\downarrow\omega\downarrow\}_{j\in R(i-1)}$ et $\varphi_i = \bigcup_{1\leq j<|S(i)|+|G(i)|}\{\mathsf{w}_j \mapsto o(j)\sigma\tau\downarrow\omega\downarrow\}$ pour $1 \leq i \leq m+1$.
Si

$$(T_1, \varphi_1) \xrightarrow{\ell_1\sigma\tau\downarrow\omega} (T_2, \varphi_2) \xrightarrow{\ell_2\sigma\tau\downarrow\omega} \cdots \xrightarrow{\ell_m\sigma\tau\downarrow\omega} (T_{m+1}, \varphi_{m+1})$$

alors

$$\varphi_{m+1}(\mathsf{w}_{|S(m)|+|G(m)|}) = o(m) = g_m$$

et donc $T \models (\mathsf{k}^-_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_{|S(m)|+|G(m)|}, g_m))\omega$.

Cas 4. $T \models (\mathsf{k}^-_\epsilon(c, c) \Leftarrow)$ est satisfait trivialement puisque $\emptyset \vdash^-_c c$;

Cas 5. $T \models (\mathsf{k}^+_\epsilon(g, g) \Leftarrow)$ est satisfait trivialement puisque $\emptyset \vdash^+_g g$;

Cas 6 et 7. Soit $0 \leq m \leq n$, f une fonction d'arité k, $\tau \in \mathrm{variants}_{\mathcal{R}}(\mathsf{f}(y_1, \cdots, y_k))$, et $\star \in \{-, +\}$. Nous devons montrer que :

$$T \models \mathsf{k}^\star_{\ell_1,\ldots,\ell_m}(\mathsf{f}(Y_1, \ldots, Y_k), \mathsf{f}(y_1, \ldots, y_k)\tau\downarrow) \Leftarrow \{\mathsf{k}^\star_{\ell_1,\ldots,\ell_m}(Y_j, y_j\tau\downarrow)\}_{j\in\{1,..,k\}}.$$

Soit ω une substitution qui clôt f tel que $T \models (\mathsf{k}^\star_{\ell_1,\ldots,\ell_m}(Y_j, y_j\tau\downarrow))\omega$ pour tout $1 \leq j \leq k$. Nous devons montrer que :

$$T \models \mathsf{k}^\star_{\ell_1,\ldots,\ell_m}(\mathsf{f}(Y_1, \ldots, Y_k), \mathsf{f}(y_1, \ldots, y_k)\tau\downarrow)\omega.$$

Supposons que

$$(T_1, \varphi_1) \xrightarrow{\ell_1\omega} (T_2, \varphi_2) \xrightarrow{\ell_2\omega} \cdots \xrightarrow{\ell_m\omega} (T_{m+1}, \varphi_{m+1})$$

Si cette dérivation n'est pas possible nous pouvons conclure. Nous avons que $\varphi_{m+1} \vdash^{\star}_{Y_j\omega} y_j\tau{\downarrow}\omega$ pour tout $1 \leq j \leq k$. Ceci implique :

$$\begin{array}{rl} \varphi_{m+1} \vdash^{\star}_{\mathsf{f}(Y_1\omega,\cdots,Y_k\omega)} & \mathsf{f}(y_1\tau{\downarrow}\omega,\ldots,y_k\tau{\downarrow}\omega) \\ =_{\mathcal{R}} & \mathsf{f}(y_1,\cdots,y_k)\tau{\downarrow}\omega \end{array}$$

et d'où $T \models \mathsf{k}^{\star}_{\ell_1,\ldots,\ell_m}(\mathsf{f}(Y_1,\ldots,Y_k),\mathsf{f}(y_1,\ldots,y_k)\tau{\downarrow})\omega$.

□

Proposition 1 *Soit T une trace close, et K un ensemble de statements tel que pour tout $f \in K$ nous avons que $T \models f$. Nous avons que $T \models f$ pour tout $f \in \mathcal{H}(K)$.*

Preuve. Soit $f \in \mathcal{H}(K)$, et π un arbre de preuve témoignant de ce fait. Nous procédons par induction structurelle sur π.

Cas de base. Dans un tel cas, l'arbre de preuve π est obtenu en appliquant la règle Conseq avec $n = 0$. Nous avons que $f' = (H \Leftarrow) \in K$ et $f = f'\sigma$ où σ est une substitution qui clôt f'. Comme $f' \in K$, par hypothèse, $T \models f'$. D'où, comme toutes les variables dans f' sont universellement quantifiées, $T \models f'\sigma$.

Cas inductif. Nous procédons par distinction de cas sur la dernière règle de π.

- Conseq : Nous avons que $f' = (H \Leftarrow B_1,\cdots,B_n) \in K$, σ est une substitution qui clôt f' tel que $f = H\sigma$ et $B_i\sigma \in \mathcal{H}(K)$ pour $1 \leq i \leq n$. Comme $H \Leftarrow B_1,\cdots,B_n \in K$ nous avons par hypothèse que $T \models (H \Leftarrow B_1,\cdots,B_n)$ et d'où $T \models (H \Leftarrow B_1,\cdots,B_n)\sigma$. Par hypothèse d'induction nous avons aussi que $T \models B_i\sigma$. D'où, nous concluons que $T \models H\sigma$.
- Extend : Dans un tel cas, nous avons que $f = \mathsf{k}^{\star}_{uv}(R,t)$, et nous savons que $\mathsf{k}^{\star}_{u}(R,t) \in \mathcal{H}(K)$. Par hypothèse d'induction $T \models \mathsf{k}^{\star}_{u}(R,t)$. La sémantique de $\mathsf{k}^{\star}$ nous donne $T \models \mathsf{k}^{\star}_{uv}(R,t)$.
- Weakening : Dans un tel cas, nous avons que $f = \mathsf{k}^{+}_{w}(R,t)$, et nous savons que $\mathsf{k}^{-}_{w}(R,t) \in \mathcal{H}(K)$. Par hypothèse d'induction $T \models \mathsf{k}^{-}_{w}(R,t)$. De par la sémantique de k^{+} et k^{-} nous obtenons $T \models \mathsf{k}^{+}_{w}(R,t)$. □

La correction du Théorème 1 est une conséquence du Lemme 2 et de la Proposition 1.

Nous montrons maintenant la complétude du Théorème 1.

Lemme 3 *Soit T une trace close. Si $(T,\emptyset) \xrightarrow{l_1,\ldots,l_m}\!\!\!\!\!\rightarrow (Q,\varphi)$ alors*

1. $\mathsf{r}_{l_1{\downarrow},\ldots,l_m{\downarrow}} \in \mathcal{H}(\mathsf{seed}(T))$;
2. *si $\varphi \vdash^{\star}_{R} t$ avec $\star \in \{-,+\}$ alors $\mathsf{k}^{\star}_{l_1{\downarrow},\ldots,l_m{\downarrow}}(R,t{\downarrow}) \in \mathcal{H}(\mathsf{seed}(T))$.*

Preuve. Supposons la même convention de nommage pour T et les étiquettes ℓ_i que dans la Section 4.2.2. Comme $(T,\emptyset) \xrightarrow{l_1,\ldots,l_m}\!\!\!\!\!\rightarrow (Q,\varphi)$, il existe ω tel que $(l_1,\ldots,l_m) = (\ell_1,\ldots,\ell_m)\omega$ et $s_k\omega =_{\mathcal{R}} t_k\omega$ pour tout $k \in T^{=}(n)$. Nous procédons par induction sur m.

Pour montrer le premier point, nous observons qu'il existe $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^{=}(n)})$ tel que :

1. $\mathrm{dom}(\sigma) \subseteq X$;
2. $s_k\sigma =_{\mathcal{R}} t_k\sigma$ pour tout $k \in T^{=}(n)$;
3. $\omega[X] =_{\mathcal{R}} (\sigma\pi)[X]$ pour une substitution π

où $X = \mathrm{vars}(\{s_k, t_k\}_{k\in T^{=}(n)})$. Il suit que $(\ell_1, \ldots, \ell_m)\omega\downarrow = (\ell_1, \ldots, \ell_m)\sigma\pi\downarrow$. Par définition de $\mathrm{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_m\sigma)$, il existe $\tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1\sigma, \ldots, \ell_m\sigma)$ tel que $(\ell_1, \ldots, \ell_m)\sigma\pi\downarrow = (\ell_1, \ldots, \ell_m)\sigma\tau\downarrow\tau'$ pour une substitution τ'. Par définition de $\mathsf{seed}(T)$, nous avons que $f \in \mathsf{seed}(T)$ où

$$f = \mathsf{r}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow} \Leftarrow \{\mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow,\cdots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)_{j\in R(m)}\}$$

Soit τ'' la substitution qui étend τ' avec $\{X_j \mapsto R_j\}_{j\in R(m)}$ où R_j sont les recettes pour $x_j\omega$. Nous avons par hypothèse d'induction que chaque antécédent de $f\tau''$ est dans $\mathcal{H}(\mathsf{seed}(T))$. Donc

$$\mathsf{r}_{\ell_1\sigma\tau\downarrow\tau'',\ldots,\ell_m\sigma\tau\downarrow\tau''} = \mathsf{r}_{\ell_1\sigma\tau\downarrow\tau',\ldots,\ell_m\sigma\tau\downarrow\tau'} \in \mathcal{H}(\mathsf{seed}(T))$$

Nous prouvons maintenant le second point. Supposons que le statement est satisfait pour tout indice plus petit que m et prouvons le pour m.

Par induction sur R nous montrons que : si $\varphi \vdash^{\star}_{R} R\varphi\downarrow$ alors $\mathsf{k}^{\star}_{l_1\downarrow,\cdots,l_m\downarrow}(R, R\varphi\downarrow) \in \mathcal{H}(\mathsf{seed}(T))$ avec $\star \in \{-, +\}$.

1. Si $R = c$ est un nom public, comme $f = (\mathsf{k}^{-}_{\epsilon}(c, c) \Leftarrow) \in \mathsf{seed}(T)$ par définition, nous avons que $\mathsf{k}^{-}_{\epsilon}(c, c) \in \mathcal{H}(\mathsf{seed}(T))$ et $\mathsf{k}^{-}_{l_1\downarrow,\cdots,l_m\downarrow}(c, c) \in \mathcal{H}(\mathsf{seed}(T))$ par la règle Extend. Par application de la règle Weakening $\mathsf{k}^{+}_{l_1\downarrow,\cdots,l_m\varphi\downarrow}(c, c) \in \mathsf{seed}(T)$.
2. Si $R = \mathsf{w}_k$, soit n le plus petit indice tel que $|S(n)|+|G(n)| = k$, nous distinguons deux cas :

   (a) $\mathsf{a}_n = \mathbf{out}(c, t_n)$ pour un canal c quelconque. Nous choisissons $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k\in T^{=}(n)})$ tel que $(\ell_1, \cdots, \ell_n)\omega\downarrow = (\ell_1, \cdots, \ell_n)\sigma\pi\downarrow$ pour une substitution π. Soit $\tau \in \mathrm{variants}\,(\ell_1, \cdots, \ell_n, t_n)\sigma$ et τ' tels que $(\ell_1, \cdots, \ell_n, t_n)\omega = (\ell_1, \cdots, \ell_n, t_n)\sigma\tau\downarrow\tau'$. Par définition de $\mathsf{seed}(T)$, nous avons que $h \in \mathsf{seed}(T)$ où

   $$h = \mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_n\sigma\tau\downarrow}(\mathsf{w}_k, t_n\sigma\tau\downarrow) \Leftarrow \{\mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j\in R(n)}$$

   Pour $j \in R(n)$, soit R_j une recette de $x_j\sigma\tau\downarrow\tau' =_{\mathcal{R}} x_j\omega$ dans le plus petit préfixe possible de φ. Soit $\tau'' = \tau' \cup \{X_j \mapsto R_j\}_{j\in R(n)}$. Les antécédents de $h\tau''$ sont dans $\mathcal{H}(\mathsf{seed}(T))$ par hypothèse d'induction. C'est pourquoi

   $$\begin{array}{rl} & \mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow\tau'',\ldots,\ell_n\sigma\tau\downarrow\tau''}(\mathsf{w}_k, t_n\sigma\tau\downarrow\tau'') \\ = & \mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow\tau',\cdots,\ell_n\sigma\tau\downarrow\tau'}(\mathsf{w}_k, t_n\sigma\tau\downarrow\tau') \\ = & \mathsf{k}^{-}_{\ell_1\omega\downarrow,\cdots,\ell_n\omega\downarrow}(\mathsf{w}_k, t_n\omega\downarrow) \in \mathcal{H}(\mathsf{seed}(T)) \end{array}$$

   Puisque $(\ell_1, \cdots, \ell_n)\omega\downarrow$ est un préfixe (pas forcément strict) de $(\ell_1, \cdots, \ell_m)\omega\downarrow$, en appliquant éventuellement la règle Extend, nous avons que $\mathsf{k}^{-}_{\ell_1\omega\downarrow,\ldots,\ell_m\omega\downarrow}(\mathsf{w}_k, t_n\omega\downarrow) = \mathsf{k}^{-}_{\ell_1\omega\downarrow,\ldots,\ell_m\omega\downarrow}(R_j, R_j\varphi\downarrow) \in \mathcal{H}(\mathsf{seed}(T))$. Utilisant la règle Weakening nous avons aussi que $\mathsf{k}^{+}_{\ell_1\omega\downarrow,\ldots,\ell_m\omega\downarrow}(R_j, R_j\varphi\downarrow) \in \mathcal{H}(\mathsf{seed}(T))$.

(b) $\mathsf{a}_n = \mathsf{guess}(g_n)$. Soit $\sigma \in \mathrm{csu}_{\mathcal{R}}(\{s_k = t_k\}_{k \in T^{=}(n)})$ tel que $(\ell_1, \ldots, \ell_k)\omega\downarrow = (\ell_1, \ldots, \ell_n)\sigma\pi\downarrow$ pour une substitution π. Soit $\tau \in \mathrm{variants}_{\mathcal{R}}(\ell_1, \cdots, \ell_n)\sigma$ et τ' tel que $(\ell_1, \ldots, \ell_n)\omega = (\ell_1, \ldots, \ell_n)\sigma\tau\downarrow\tau'$. Par définition de $\mathsf{seed}(T)$, nous avons que $h \in \mathsf{seed}(T)$ où

$$h = \mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow,\ldots,\ell_m\sigma\tau\downarrow}(\mathsf{w}_k, g_n) \Leftarrow \{\mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow,\cdots,\ell_{j-1}\sigma\tau\downarrow}(X_j, x_j\sigma\tau\downarrow)\}_{j \in R(n)}$$

Pour $j \in R(n)$, soit R_j une recette de $x_j\sigma\tau\downarrow\tau' =_{\mathcal{R}} x_j\omega$ dans le plus petit préfixe possible de φ. Soit $\tau'' = \tau' \cup \{X_j \mapsto R_j\}_{j \in R(n)}$. Nous avons que les antécédents de $h\tau''$ sont dans $\mathcal{H}(\mathsf{seed}(T))$ par hypothèse d'induction. Donc

$$\begin{array}{rl} & \mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow\tau'',\ldots,\ell_n\sigma\tau\downarrow\tau''}(\mathsf{w}_k, g_n) \\ = & \mathsf{k}^{-}_{\ell_1\sigma\tau\downarrow\tau',\ldots,\ell_n\sigma\tau\downarrow\tau'}(\mathsf{w}_k, g_n) \\ = & \mathsf{k}^{-}_{\ell_1\omega\downarrow,\ldots,\ell_n\omega\downarrow}(\mathsf{w}_k, g_n) \in \mathcal{H}(\mathsf{seed}(T)) \end{array}$$

Puisque $(\ell_1, \ldots, \ell_n)\omega\downarrow$ est un préfixe de $(\ell_1, \ldots, \ell_m)\omega\downarrow$, en appliquant éventuellement la règle Extend, nous avons que $\mathsf{k}^{-}_{\ell_1\omega\downarrow,\ldots,\ell_m\omega\downarrow}(\mathsf{w}_k, g_m) = \mathsf{k}^{-}_{\ell_1\omega\downarrow,\ldots,\ell_m\omega\downarrow}(R_j, R_j\varphi\downarrow) \in \mathcal{H}(\mathsf{seed}(T))$. Utilisant la règle Weakening, nous avons aussi que $\mathsf{k}^{+}_{\ell_1\omega\downarrow,\ldots,\ell_m\omega\downarrow}(R_j, R_j\varphi\downarrow) \in \mathcal{H}(\mathsf{seed}(T))$

3. Si $R = g$ est un nom devinable, alors nous avons que $\star = +$, et comme $(f = \mathsf{k}^{+}_{\epsilon}(g, g) \Leftarrow) \in \mathsf{seed}(T)$ par définition, nous avons que $\mathsf{k}^{+}_{\epsilon}(g, g) \in \mathcal{H}(\mathsf{seed}(T))$ et $\mathsf{k}^{+}_{l_1\downarrow,\ldots,l_m\varphi\downarrow}(g, g) \in \mathcal{H}(\mathsf{seed}(T))$ par la règle Extend.
4. Si $R = \mathsf{f}(R_1, \ldots, R_k)$, soit $\tau \in \mathrm{variants}_{\mathcal{R}}(\mathsf{f}(y_1, \ldots, y_k))$ et τ' tel que $R\varphi\downarrow = (\mathsf{f}(y_1, \ldots, y_k)\tau)\downarrow\tau'$. Par la définition de $\mathsf{seed}(T)$, nous avons le statement $h \in \mathsf{seed}(T)$ avec $\star \in \{-, +\}$:

$$\mathsf{k}^{\star}_{\ell_1,\cdots,\ell_n}(\mathsf{f}(Y_1, \cdots, Y_k), \mathsf{f}(y_1, \cdots, y_k)\tau\downarrow) \Leftarrow \{\mathsf{k}^{\star}_{\ell_1,\cdots,\ell_n}(Y_j, y_j\tau\downarrow)\}_{j \in \{1,\cdots,k\}}$$

Soit $\tau'' = \omega \cup \tau' \cup \{Y_j \mapsto R_j\}_{j \in \{1,\ldots,k\}}$. Nous avons que tous les antécédents de $g\tau''$ sont dans $\mathcal{H}(\mathsf{seed}(T))$ par hypothèse d'induction. C'est pourquoi, la tête de $g\tau''$ est aussi dans $\mathcal{H}(\mathsf{seed}(T))$, et nous concluons. □

4.3 Saturation

La phase de saturation va appliquer une procédure de résolution sur un ensemble de statements et va générer des tests existant dans des frames atteignables.

4.3.1 Procédure de saturation

Dans cette section, nous décrivons la procédure de saturation. Elle modifie l'ensemble de statements provenant de la graine, nous appelons cet ensemble modifié une *base de connaissance.*

Définition 11 *Soit un statement $f = (H \Leftarrow B_1, \ldots, B_n)$,*

— *f est dit résolu si pour tout $1 \leq i \leq n$, nous avons que $B_i = \mathsf{k}^{\star_i}_{w_i}(X_i, x_i)$ pour $\star_i \in \{-, +\}$, $x_i \in \mathcal{X}$, et $X_i \in \mathcal{Y}$.*

— *f est dit bien formé si quand f est résolu et que $H = \mathsf{k}^{\star}_{w}(R,t)$ pour $\star \in \{-,+\}$, nous avons que $t \notin \mathcal{X}$.*

Un ensemble de statements bien formés est appelé une base de connaissance. Si K est une base de connaissance, $K_{\mathsf{solved}} = \{f \in K \mid f \text{ est résolu }\}$.

La procédure de saturation (notée sat) est un processus non-déterministe dont le but est de produire une base de connaissance depuis une base initiale. La procédure de saturation agit de la façon suivante : chaque fois qu'un nouveau statement est *généré*, la base de connaissance est *mise à jour* avec celui ci ; ce processus continue jusqu'à l'obtention d'un point fixe ; à la fin nous obtenons une base de connaissance $K_{\mathsf{sat}} \in \mathsf{sat}(K_{\mathsf{init}})$. L'ensemble des points fixes atteignables depuis une base initiale K_{init} est notée $\mathsf{sat}(K_{\mathsf{init}})$.

Avant d'introduire les règles de saturation nous définissons $\mathcal{G}^{\ell}(w)$ comme l'ensemble des indices correspondant aux labels guess dans le monde w, i.e., $G^{\ell}(w) = \{i \mid w = \ell_1, \ldots, \ell_n, 1 \leq i \leq n \text{ et } \ell_i = \mathsf{guess}(g_i)\}$.

$$\textsc{Resolution}\ \frac{\begin{array}{c} f = \Big(H \Leftarrow \mathsf{k}^{\star}_{uv}(X,t), B_1, \ldots, B_n\Big) \\ g = \Big(\mathsf{k}^{\star'}_{w}(R,t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \qquad f \in K, g \in K_{\mathsf{solved}} \\ \sigma = \mathrm{mgu}(\mathsf{k}_u(X,t), \mathsf{k}_w(R,t')) \qquad t \notin \mathcal{X} \qquad \star, \star' \in \{-,+\} \qquad \star' \leq \star \end{array}}{K = K \Cup h \text{ où } h = \Big(H \Leftarrow B_1, \cdots, B_m\Big)\sigma}$$

$$\textsc{Equation}\ \frac{\begin{array}{c} f = \Big(\mathsf{k}^{\star}_{u}(R,t) \Leftarrow B_1, \ldots, B_n\Big) \quad g = \Big(\mathsf{k}^{\star'}_{u'v'}(R',t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \\ f, g \in K_{\mathsf{solved}} \qquad \sigma = \mathrm{mgu}(\langle u,t\rangle, \langle u',t'\rangle) \qquad \star, \star' \in \{-,+\} \end{array}}{K = K \Cup h \text{ où } h = \Big(\mathsf{i}_{u'v'}(R,R') \Leftarrow B_1, \ldots, B_m\Big)\sigma}$$

$$\textsc{Test}\ \frac{\begin{array}{c} g = \mathsf{r}_w \Leftarrow \mathcal{B} \quad f_1 = \Big(\mathsf{i}_{w_1}(R_1, R'_1) \Leftarrow \mathcal{B}_1\Big) \quad \cdots \quad f_n = \Big(\mathsf{i}_{w_n}(R_n, R'_n) \Leftarrow \mathcal{B}_n\Big) \\ g, f_1, \ldots, f_n \in K_{\mathsf{solved}} \\ \sigma = \mathrm{mgu}(\{w_i x_i, = w\}_{1 \leq i \leq n}) \quad n = |\mathcal{G}^{\ell}(w)|, \text{ et } |\mathcal{G}^{\ell}(w_i)| < i \text{ pour } 1 \leq i \leq n \end{array}}{K = K \Cup h \text{ où } h = \Big(\mathsf{ri}_w((R_1, \ldots, R_n), (R'_1, \ldots, R'_n)) \Leftarrow \mathcal{B}, \mathcal{B}_1, \ldots, \mathcal{B}_n\Big)\sigma}$$

FIGURE 4.3 – Règles de saturation

Génération de nouveaux statements : Étant donnée une base de connaissance K, des nouveaux statements sont générés en appliquant les règles de la Figure 4.3. Chaque règle génère un nouveau statement h. La règle RÉSOLUTION applique la règle de résolution similaire à la logique de premier-ordre entre un statement non résolu f et un statement résolu g. Nous n'avons pas besoin que $\star' = \star$ mais seulement que $\star' \leq \star$, ce qui est formellement défini comme suit :

$$\star' \leq \star \text{ si, et seulement si, } \begin{cases} \text{soit } \star' = \star \ ; \\ \text{soit } (\star', \star) = (-,+). \end{cases}$$

Cette condition reflète l'intuition que tout terme fortement déductible, est aussi faiblement déductible. La règle ÉQUATION nous permet de dériver de nouvelles identités sur les recettes. La règle TEST nous permet de dériver une séquence d'identités (une par action **guess**) qui sont nécessairement satisfaites au cours de l'exécution du protocole. Une fois que le statement h est généré, nous mettons à jour la base de connaissnace K avec h comme expliqué ci-dessous.

Mise à jour : Nous définissons maintenant l'opérateur de mise à jour $\Cup\!\!\!\!\cdot$ qui ajoute les statements générés par les règles de la Figure 4.3 à la base de connaissance.

Définition 12 *Soit K une base de connaissance, et $f = H \Leftarrow \mathcal{B}$ un statement. La* mise à jour de K par f*, noté $K \Cup\!\!\!\!\cdot\, f$, est $K \cup \{f\}$ quand f soit n'est pas un statement de déduction résolu, ou est un statement de déduction bien formé et résolu. Autrement, nous avons que $f = \mathsf{k}^{\star}_{u}(R, x) \Leftarrow \mathcal{B}$ avec des statements de la forme $\mathsf{k}^{\star'}_{u'}(X, x) \in \mathcal{B}$ (par définition d'un statement). Dans un tel cas, nous avons que :*

$$K \Cup\!\!\!\!\cdot\, f = K \cup (\mathsf{i}_u(R, X) \Leftarrow \mathcal{B})$$

Base de connaissance initiale : Nous définissons finalement sur quelle base de connaissance nous initions la procédure de saturation.

Définition 13 *Soit un ensemble de statements S, la* base de connaissance initiale *associée à S, notée $K_{\mathsf{init}}(S)$, est définie pour être la base de connaissance vide mise à jour par l'ensemble S, i.e.,*

$$K_{\mathsf{init}}(S) = (((\emptyset \Cup\!\!\!\!\cdot\, f_1) \Cup\!\!\!\!\cdot \ldots) \Cup\!\!\!\!\cdot\, f_n)$$

où $f_1, \ldots, f_n$ est une énumération des statements dans S.

Soit une trace close T, $\mathcal{N}^0_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$, et $\mathcal{N}^0_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$. Nous avons que l'ensemble $K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}^0_{\mathsf{pub}}, \mathcal{N}^0_{\mathsf{guess}}))$ est une base de connaissance. De plus, étant donnée une base de connaissance K, tout ensemble de statements obtenu par application de la procédure de saturation à cette base de connaissance est aussi une base de connaissance.

Exemple 24 *Nous considérons la graine introduite dans l'Exemple 23 :*

$$\begin{array}{ll}
f_1: & \mathsf{k}^{-}_{\mathbf{out}(c)}(\mathsf{w}_1, m_A) \Leftarrow \\
f_2: & \mathsf{k}^{-}_{\mathbf{out}(c).\mathbf{guess}(r_A)}(\mathsf{w}_2, r_A) \Leftarrow \\
f_3: & \mathsf{r}_w \Leftarrow \mathsf{k}^{-}_{\mathbf{out}(c).\mathbf{guess}(r_A)}(Y, \langle y_1, \mathsf{hash}(\langle y_1, r_A\rangle)\rangle) \\
f^{+}_4: & \mathsf{k}^{+}_{\mathbf{out}(c)}(\mathsf{hash}(X), \mathsf{hash}(x)) \Leftarrow \mathsf{k}^{+}_{\mathbf{out}(c)}(X, x) \\
f^{-}_5: & \mathsf{k}^{-}_{\mathbf{out}(c)}(\mathsf{proj}_2(Y), \mathsf{proj}_2(y)) \Leftarrow \mathsf{k}^{-}_{\mathbf{out}(c)}(Y, y) \\
f^{-}_6: & \mathsf{k}^{-}_{\mathbf{out}(c)}(\mathsf{proj}_2(Y), y_2) \Leftarrow \mathsf{k}^{-}_{\mathbf{out}(c)}(Y, \langle y_1, y_2\rangle)
\end{array}$$

Une application de la règle de résolution entre f^{-}_6 et le statement résolu f_1 nous amène au statement résolu suivant :

$$h = \mathsf{k}^{-}_{\mathbf{out}(c)}(\mathsf{proj}_2(\mathsf{w}_1), \mathsf{hash}(\langle info_A, r_A\rangle)) \Leftarrow$$

qui est simplement ajouté à la base de connaissance par l'opérateur de mise à jour. Nous pouvons ensuite décider d'appliquer la règle équation entre h et f_4^+ ce qui nous amène au statement :

$$\mathsf{i}_{\mathbf{out}(c)}(\mathsf{proj}_2(\mathsf{w}_1), \mathsf{hash}(X)) \Leftarrow \mathsf{k}^+_{\mathbf{out}(c)}(X, \langle info_A, r_A \rangle)$$

et ensuite, après plusieurs étapes de résolution, nous pouvons dériver le statement résolu

$$\mathsf{i}_{\mathbf{out}(c)}(\mathsf{proj}_2(\mathsf{w}_1), \mathsf{hash}(\langle info_A, r_A \rangle) \Leftarrow$$

De la même manière, nous sommes capables de dériver $\mathsf{r}_w \Leftarrow \mathsf{k}^-_{\mathbf{out}(c),\mathbf{guess}(r_A)}(Y_1, y_1)$ *en réalisant plusieurs étapes de résolution depuis* f_3*. C'est pourquoi, nous pouvons appliquer la règle* Test *pour dériver* f_{final} *:*

$$\mathsf{ri}_w(\mathsf{proj}_2(\mathsf{w}_1), \mathsf{hash}(\langle info_A, r_A \rangle) \Leftarrow \mathsf{k}^-_{\mathbf{out}(c),\mathbf{guess}(r_A)}(Y_1, y_1)$$

Intuitivement, ce statement signifie que l'exécution symbolique w est en effet exécutable et nous amène au test $\mathsf{proj}_2(\mathsf{w}_1) = \mathsf{hash}(\langle info_A, r_A \rangle)$ *qui tient dans la frame résultante. Si l'attaquant connait une valeur publique qu'il utilisera pour instancier* y_1*.*

4.3.2 Correction et complétude de la saturation

Nous étendons $\mathcal{H}(K)$ pour établir que tout $K_{\mathsf{sat}} \in \mathsf{sat}(K_{\mathsf{init}}(T))$ est une abstraction complète de T. Ainsi, $\mathcal{H}_{\mathsf{e}}(K)$ va considérer les propriétés relatives aux identités, réflexivité, transitivité, symétrie, congruence, le fait qu'une identité vraie dans un petit monde est également vraie dans son extension, et l'impact que ces identités ont sur la connaissance de l'attaquant.

Définition 14 *Soit K un ensemble de statements. Nous définissons* $\mathcal{H}_{\mathsf{e}}(K)$ *comme le plus petit ensemble de faits contenant* $\mathcal{H}(K)$ *et qui est clos sous les règles de la Figure 4.4.*

$$\textsc{Refl}\ \frac{}{\mathsf{i}_w(R, R) \in \mathcal{H}_{\mathsf{e}}(K)} \qquad \textsc{Tran}\ \frac{\mathsf{i}_w(R_1, R_2) \in \mathcal{H}_{\mathsf{e}}(K) \quad \mathsf{i}_w(R_2, R_3) \in \mathcal{H}_{\mathsf{e}}(K)}{\mathsf{i}_w(R_1, R_3) \in \mathcal{H}_{\mathsf{e}}(K)}$$

$$\textsc{Sym}\ \frac{\mathsf{i}_w(R_1, R_2) \in \mathcal{H}_{\mathsf{e}}(K)}{\mathsf{i}_w(R_2, R_1) \in \mathcal{H}_{\mathsf{e}}(K)} \qquad \textsc{Cong}\ \frac{\mathsf{i}_w(R_1, R'_1), \ldots, \mathsf{i}_w(R_n, R'_n) \in \mathcal{H}_{\mathsf{e}}(K) \ \ \mathsf{f} \in \Sigma}{\mathsf{i}_w(\mathsf{f}(R_1, \ldots R_n), \mathsf{f}(R'_1, \ldots R'_n)) \in \mathcal{H}_{\mathsf{e}}(K)}$$

$$\textsc{Ext}\ \frac{\mathsf{i}_u(R, R') \in \mathcal{H}_{\mathsf{e}}(K)}{\mathsf{i}_{uv}(R, R') \in \mathcal{H}_{\mathsf{e}}(K)} \qquad \textsc{Eq. Conseq.}\ \frac{\mathsf{k}^+_w(R, t) \in \mathcal{H}(K) \qquad \mathsf{i}_w(R, R') \in \mathcal{H}_{\mathsf{e}}(K)}{\mathsf{k}^+_w(R', t) \in \mathcal{H}_{\mathsf{e}}(K)}$$

Figure 4.4 – Règles de $\mathcal{H}_{\mathsf{e}}(K)$

Nous avons que l'ensemble de statements résolus produit par la procédure de saturation est une abstraction correcte et complète de la trace close T (considérant $\twoheadrightarrow$). Plus formellement nous avons le résultat suivant.

Théorème 2 *Soit $K \in \mathsf{sat}(K_{\mathsf{init}}(T))$ pour une trace close T. Nous avons que :*

— *Correction : $T \models f$ pour tout $f \in K \cup \mathcal{H}_{\mathsf{e}}(K)$;*

— *Complétude : Si $(T, \emptyset) \xrightarrow{l_1,\ldots,l_n}\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow (Q, \phi)$ alors*

1. *$\mathsf{r}_{l_1\downarrow,\ldots,l_n\downarrow} \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$;*
2. *si $\phi \vdash^+_R t$ alors $\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(R, t\downarrow) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$;*
3. *si $\phi \vdash^+_R t$ et $\phi \vdash^+_{R'} t$, alors $\mathsf{i}_{l_1\downarrow,\ldots,l_n\downarrow}(R, R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$.*

4.3.3 Preuve de la correction et de la complétude de la saturation

La correction du Théorème 2 est une conséquence des lemmes énoncés et prouvés dans cette section. Il y a 3 lemmes représentant les 3 règles de la saturation : le Lemme 4 concernant la règle de RESOLUTION, le Lemme 5 pour la règle EQUATION, et le Lemme 6 pour la règle TEST. Ensuite, nous établissons la correction de l'opération de mise à jour (voir Lemme 7), et de $\mathcal{H}_{\mathsf{e}}$ (voir Lemme 8).

La complétude de ce même théorème est exhibé par la démonstration de la Proposition 3. Cette Proposition est conséquence des Lemmes 9, 10 et de la Proposition 2.

Lemme 4 *Soit T une trace close. Nous considérons les statements f, g et h comme définis dans la règle de* RESOLUTION. *Si $T \models f$ et $T \models g$ alors nous avons que $T \models h$.*

Preuve. Nous avons que :

— $f = \Big(H \Leftarrow \mathsf{k}^\star_{uv}(X, t), B_1, \ldots, B_n\Big) \in K$,

— $g = \Big(\mathsf{k}^{\star'}_w(R, t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \in K_{\mathsf{solved}}$, et

— $h = \Big(H \Leftarrow B_1, \cdots, B_m\Big)\sigma$

où $\sigma = \mathrm{mgu}(\mathsf{k}_u(X, t), \mathsf{k}_w(R, t'))$, $t \notin \mathcal{X}$, $\star, \star' \in \{-, +\}$, et $\star' \leq \star$.

Soit τ une substitution arbitraire qui clôt h tel que $T \models B_i\sigma\tau$ pour $1 \leq i \leq m$. Nous devons montrer que $T \models H\sigma\tau$.

Nous avons que $T \models g$, et $T \models B_j\sigma\tau$ pour $n + 1 \leq j \leq m$. Ainsi, nous déduisons que $T \models \mathsf{k}^{\star'}_w(R, t')\sigma\tau$. Par définition de σ, nous avons que $T \models \mathsf{k}^{\star'}_u(X, t)\sigma\tau$, et nous déduisons que $T \models \mathsf{k}^\star_{uv}(X, t)\sigma\tau$ depuis la sémantique des prédicats k^- et k^+. Donc, nous avons que $T \models \mathsf{k}^\star_{uv}(X, t)\sigma\tau$, $T \models B_i\sigma\tau$ pour $i \in \{1, \ldots n\}$, et $T \models f$. Cela nous permet de conclure que $T \models H\sigma\tau$. □

Lemme 5 *Soit T une trace close. Nous considérons les statements f, g et h comme défini dans la règle* EQUATION. *Si $T \models f$ et $T \models g$ alors nous avons que $T \models h$.*

Preuve. Nous avons que :

— $f = \Big(\mathsf{k}^\star_u(R, t) \Leftarrow B_1, \ldots, B_n\Big) \in K_{\mathsf{solved}}$,

— $g = \Big(\mathsf{k}^{\star'}_{u'v'}(R', t') \Leftarrow B_{n+1}, \ldots, B_m\Big) \in K_{\mathsf{solved}}$, et

— $h = \Big(\mathsf{i}_{u'v'}(R, R') \Leftarrow B_1, \ldots, B_m\Big)\sigma$

où $\sigma = \text{mgu}(\langle u, t\rangle, \langle u', t'\rangle)$, et $\star, \star' \in \{-, +\}$.

Soit τ une substitution arbitraire qui clôt h tel que $T \models B_i\sigma\tau$ pour $1 \leq i \leq m$. Nous devons montrer que $T \models \mathsf{i}_{u'v'}(R, R')\sigma\tau$.

Nous avons que $T \models f$, et $T \models B_j\sigma\tau$ pour $1 \leq j \leq n$. Donc, nous déduisons que $T \models \mathsf{k}^{\star}_{u}(R, t)\sigma\tau$. Nous avons aussi que $T \models g$, et $T \models B_j\sigma\tau$ pour $n+1 \leq j \leq m$. C'est pourquoi, nous déduisons que $T \models \mathsf{k}^{\star'}_{u'v'}(R', t')\sigma\tau$. Par définition de σ, et, de par la sémantique de k^{-} et k^{+}, nous avons que $T \models \mathsf{k}^{+}_{u'v'}(R, t')\sigma\tau$, et $T \models \mathsf{k}^{+}_{u'v'}(R', t')\sigma\tau$. Cela nous permet de conclure que $T \models \mathsf{i}_{u'v'}(R, R')\sigma\tau$. □

Lemme 6 *Soit T une trace close. Nous considérons les statements g, $f_0, \ldots, f_n$, et h comme définis dans la règle* TEST. *Si $T \models g$, et $T \models f_i$ pour $i \in \{0, \ldots, n\}$ alors nous avons que $T \models h$.*

Preuve. Nous avons que :

— $g = \Big(\mathsf{r}_w \Leftarrow \mathcal{B}\Big) \in K_{\mathsf{solved}}$,

— $f_i = \Big(\mathsf{i}_{w_i}(R_i, R'_i) \Leftarrow \mathcal{B}_i\Big) \in K_{\mathsf{solved}}$ pour $i \in \{1, \ldots, n\}$, et

— $h = \Big(\mathsf{ri}_w(\overrightarrow{R}, \overrightarrow{R'}) \Leftarrow \mathcal{B}, \mathcal{B}_0, \ldots, \mathcal{B}_n\Big)\sigma$

où $n = |\mathcal{G}^{\ell}(w)|$, $|\mathcal{G}^{\ell}(w_i)| = i$ pour $0 \leq i \leq n$, et $\sigma = \text{mgu}(\{w_0x_0 = w, \ldots, w_nx_n = w\})$.

Soit τ une substitution quelconque qui clôt h tel que $T \models \mathcal{B}\sigma\tau$, et $T \models \mathcal{B}_i\sigma\tau$ pour $0 \leq i \leq n$. Nous devons montrer que $T \models \mathsf{ri}_w(\overrightarrow{R}, \overrightarrow{R'})\sigma\tau$.

Nous avons que $T \models g$ et $T \models \mathcal{B}\sigma\tau$. Donc, nous déduisons que $T \models \mathsf{r}_w\sigma\tau$. Soit $i \in \{1, \ldots, n\}$. Nous avons aussi que $T \models f_i$, et $T \models \mathcal{B}_i\sigma\tau$. C'est pourquoi, nous déduisons que $T \models \mathsf{i}_{w_i}(R_i, R'_i)\sigma\tau$ pour $i \in \{1, \ldots, n\}$. Par définition de σ, et par la sémantique de i, nous avons que $T \models \mathsf{i}_w(R_i, R'_i)\sigma\tau$ pour $i \in \{1, \ldots, n\}$. Cela nous permet de conclure que $T \models \mathsf{ri}_w(\overrightarrow{R}, \overrightarrow{R'})\sigma\tau$. □

Lemme 7 *Soit T une trace close, et K une base de connaissance telle que $T \models f$ pour tout $f \in K$. Soit h un statement tel que $T \models h$. Nous avons que $T \models g$ pour tout $g \in K \Cup h$.*

Preuve. Dans le cas où $K \Cup h$ est $K \cup \{h\}$ le résultat est trivialement satisfait. Sinon, nous avons que

$$h = \mathsf{k}^{\star}_{u}(R, x) \Leftarrow \mathcal{B} \text{ avec } \mathsf{k}^{\star'}_{u'}(X, x) \in \mathcal{B} \text{ pour } \star', u', \text{ et } X \text{ quelconques.}$$

Dans un tel cas, nous avons que $K \Cup h = K \cup \{h'\}$ où $h' = (\mathsf{i}_u(R, X) \Leftarrow \mathcal{B})$ Donc, nous avons simplement à montrer que $T \models h'$. Soit τ une substitution arbitraire qui clôt h' et tel que $T \models \mathcal{B}\tau$. En particulier, nous avons que $T \models \mathsf{k}^{\star'}_{u'}(X, x)\tau$, et par définition d'un statement, nous avons que $u'\tau$ est un préfixe de $u\tau$. Donc, nous avons aussi que $T \models \mathsf{k}^{\star'}_{u}(X, x)\tau$. Notons que τ clôt h, et puisque $T \models h$, nous avons que $T \models \mathsf{k}^{\star}_{u}(R, x)\tau$. De par la sémantique k^{-} et k^{+}, nous avons que $T \models \mathsf{k}^{+}_{u}(X, x)\tau$ et $T \models \mathsf{k}^{+}_{u}(R, x)\tau$. Ceci nous permet de conclure que $T \models \mathsf{i}_u(R, X)\tau$. □

Lemme 8 *Soit T une trace close, et K une base de connaissance telle que $T \models f$ pour tout $f \in K$. Nous avons que $T \models g$ pour tout $g \in \mathcal{H}_{\mathsf{e}}(K)$.*

Preuve. Nous montrons ce résultat par induction structurelle sur l'arbre de preuve témoignant du fait que $H \in \mathcal{H}_{\mathsf{e}}(K)$ basé sur la Proposition 1 pour conclure quand l'arbre de preuve est réduit à une feuille. □

Nous avons obtenu la correction du Théorème 1 comme conséquence des lemmes précédents. Nous nous intéressons maintenant à la partie complétude.

Nous introduisons tout d'abord une notation. Étant donné un arbre de preuve Π, nous notons $|\Pi|$ sa taille, i.e. le nombre de ses nœuds.

Le Lemme 9 nous montre que l'on peut passer d'un statement dans K_{solved} à sa tête instanciée dans $\mathcal{H}(K)$ en ne regardant que les statements résolus dans une base saturée.

Lemme 9 *Soit K une base de connaissance saturée et $f \in K$ un statement :*

$$f = \big(H \Leftarrow B_1, \ldots, B_n\big).$$

Si σ est une substitution qui clôt f telle que $B_i\sigma \in \mathcal{H}(K_{\mathsf{solved}})$ pour tout $1 \leq i \leq n$ alors nous avons que $H\sigma \in \mathcal{H}(K_{\mathsf{solved}})$.

Preuve. Par hypothèse nous avons une $B_i\sigma \in \mathcal{H}(K_{\mathsf{solved}})$ pour tout $1 \leq i \leq n$. Donc il existe les arbres de preuve $\Pi_1, \ldots, \Pi_n$ témoignant de ces faits.

Nous montrons le résultat pas induction sur $\sum_{i=1}^{n} |\Pi_i|$.

Si f est un statement résolu, nous concluons immédiatement par la définition de $\mathcal{H}(K_{\mathsf{solved}})$. Sinon, si f n'est pas un statement résolu, il existe un B_j $(1 \leq j \leq n)$ tel que $B_j = \mathsf{k}^{\star_j}_{w_j}(X_j, t_j)$ et $t_j \notin \mathcal{X}$. Comme $B_j\sigma \in \mathcal{H}(K_{\mathsf{solved}})$, il existe

$$g = \big(\mathsf{k}^{\star'_j}_{u'_j}(R'_j, t'_j) \Leftarrow B_{n+1}, \ldots, B_m\big) \in K_{\mathsf{solved}}$$

et une a substitution σ' qui clôt g tels que $B_{n+1}\sigma', \ldots, B_m\sigma' \in \mathcal{H}(K_{\mathsf{solved}})$, $\star'_j \leq \star_j$, $R'_j\sigma' = X_j\sigma$, $t'_j\sigma' = t_j\sigma$, $u'_j\sigma' = u_j\sigma$ avec u_j préfixe de w_j, et les arbres de preuves $\Pi_{n+1}, \ldots, \Pi_m$ témoignant $B_{n+1}\sigma', \ldots, B_m\sigma' \in \mathcal{H}(K_{\mathsf{solved}})$ tels que $\sum_{i=n+1}^{m} |\Pi_i| < |\Pi_j|$.

Comme $\omega = \sigma \cup \sigma'$ est un unificateur de $H' = \mathsf{k}_{u'_j}(R'_j, t'_j)$ et $\mathsf{k}_{u_j}(X_j, t_j)$, les deux termes sont unifiables. Nous notons $\tau = \mathrm{mgu}(\mathsf{k}_{u'_j}(R'_j, t'_j), \mathsf{k}_{u_j}(X_j, t_j))$ leur unificateur le plus général. Comme K est saturée, la règle de Resolution a été appliquée à f et g résultant le statement :

$$h = \big(H \Leftarrow B_1, \ldots, B_{j-1}, B_{j+1}, \ldots, B_m\big)\tau$$

qui se doit d'être dans K. Notons que dans ce cas H est de la forme $\mathsf{k}^{\star}_{w}(R, t)$ Nous savons que $t \notin \mathcal{X}$ puisque $f \in K$ et K est une base de connaissance. C'est pourquoi, par la fonction de mise à jour, le statement h est ajouté à la base de connaissance.

Comme ω est un unificateur de $H' = \mathsf{k}_{u'_j}(R'_j, t'_j)$ et $\mathsf{k}_{u_j}(X_j, t_j)$, alors que τ est l'unificateur le plus général, donc il existe un ω' tel que $\omega = \tau\omega'$. Nous avons que ω' est une substitution qui clôt h, et nous avons que

— $B_i\tau\omega \in \mathcal{H}(K_{\mathsf{solved}})$ for $i \in \{1, \ldots, j-1, j+1, \ldots, m\}$, et

— $\sum_{i=1}^{n} |\Pi_i| = \sum_{i=1}^{j-1} |\Pi_i| + |\Pi_j| + \sum_{i=j+1}^{n} |\Pi_i| > \sum_{i=1}^{j-1} |\Pi_i| + \sum_{i=n+1}^{m} |\Pi_i| + \sum_{i=j+1}^{n} |\Pi_i$
$= \sum_{i=1}^{j-1} |\Pi_i| + \sum_{i=j+1}^{m} |\Pi_i|$.

C'est pourquoi nous pouvons appliquer l'hypothèse d'induction à h et ω' pour conclure. □

Le Lemme 10 montre que l'on a une identité dans $\mathcal{H}(K)$ quand les deux k qui témoignent de cette identité sont dans $\mathcal{H}(K)$. Pour cela il va falloir nous replacer dans la base saturée qui a été à même de générer des identités, et vérifier qu'une telle identité est bien conséquence de cette génération.

Lemme 10 *Soit K une base de connaissance saturée. Si $\mathsf{k}^{\star}_{u}(R,t) \in \mathcal{H}(K_{\mathsf{solved}})$ et $\mathsf{k}^{\star'}_{uv}(R',t) \in \mathcal{H}(K_{\mathsf{solved}})$ alors $\mathsf{i}_w(R,R') \in \mathcal{H}(K_{\mathsf{solved}})$ pour un w préfixe de uv.*

Preuve. Comme $\mathsf{k}^{\star}_{u}(R,t) \in \mathcal{H}(K_{\mathsf{solved}})$, Nous savons qu'il existe :

$$f = \left(\mathsf{k}^{*}_{w}(S,s) \Leftarrow B_1, \ldots, B_n\right) \in K_{\mathsf{solved}}$$

et une substitution σ qui clôt f tels que $B_i\sigma \in \mathcal{H}(K_{\mathsf{solved}})$ $(1 \leq i \leq n)$, $* \leq \star$, et $\mathsf{k}_w(S,s)\sigma = \mathsf{k}_{u'}(R,t)$ pour un u' préfixe de u.

De la même manière, comme $\mathsf{k}^{\star'}_{uv}(R',t) \in \mathcal{H}(K_{\mathsf{solved}})$, nous savons qu'il existe :

$$g = \left(\mathsf{k}^{*'}_{w'}(S',s') \Leftarrow B'_1, \ldots, B'_m\right) \in K_{\mathsf{solved}}$$

et une substitution σ' qui clôt g tels que $B'_i\sigma \in \mathcal{H}(K_{\mathsf{solved}})$ $(1 \leq i \leq m)$, $*' \leq \star'$, et $\mathsf{k}_{w'}(S',s')\sigma' = \mathsf{k}_{u''}(R',t)$ pour un u'' préfixe uv.

Nous avons que $w\sigma$ est un préfixe de u ce qui implique que $w\sigma$ est un préfixe de uv, et nous savons que $w'\sigma'$ est un préfixe de uv. Soit $w = l_1, \ldots, l_p$ et $w' = l'_1, \ldots, l'_q$. Supposons $q \leq p$, l'autre cas étant le symétrique de celui-ci, nous avons que $(l_1, \ldots, l_q)\sigma = (l'_1, \ldots, l'_q)\sigma'$.

Nous avons que $\sigma \cup \sigma'$ est un unificateur de $\langle (l_1, \ldots, l_q), s\rangle$ et $\langle (l'_1, \ldots, l'_q), s'\rangle$. Suit qu'il existe $\tau = \mathrm{mgu}(\langle (l_1, \ldots, l_q), s\rangle, \langle (l'_1, \ldots, l'_q), s'\rangle)$. Comme K est saturée, le statement :

$$h = \left(\mathsf{i}_{l_1,\ldots,l_p}(S,S') \Leftarrow B_1, \ldots, B_n, B'_1, \ldots, B'_m\right)\tau \in K$$

résultant de l'application de la règle Equation à f et g est dans K. Comme $\sigma \cup \sigma'$ est un unificateur de $\langle (l_1, \ldots, l_q), s\rangle$ et $\langle (l'_1, \ldots, l'_q), s'\rangle$, et τ est l'unificateur le plus général de ces deux termes, il existe ω tel que $\sigma \cup \sigma' = \tau\omega$. Nous avons que ω clôt h et que :

$$B_1\tau\omega, \ldots, B_n\tau\omega, B'_1\tau\omega, \ldots, B'_m\tau\omega \in \mathcal{H}(K_{\mathsf{solved}})$$

Donc, par le Lemme 9 nous avons que :

$$\mathsf{i}_{l_1,\ldots,l_p}(S,S')\tau\omega = \mathsf{i}_{l_1\sigma,\ldots,l_p\sigma}(R,R') \in \mathcal{H}(K_{\mathsf{solved}}).$$

Comme $(l_1, \ldots, l_p)\sigma$ est un préfixe de uv nous pouvons conclure. □

Dans la Proposition 2 nous montrons quelles propriétés nous préservons du passage de la graine à une base saturée.

Proposition 2 *Soit S une graine et $K \in \mathsf{sat}(K_{\mathsf{init}}(S))$. Nous avons que :*
- *Si $\mathsf{r}_w \in \mathcal{H}(S)$ alors $\mathsf{r}_w \in \mathcal{H}(K_{\mathsf{solved}})$; et*
- *Si $\mathsf{k}^{\star}_w(R,t) \in \mathcal{H}(S)$ alors il existe R' tel que $\mathsf{k}^{\star}_w(R',t) \in \mathcal{H}(K_{\mathsf{solved}})$ et $\mathsf{i}_w(R,R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$.*

Preuve. Soit $H_0 = \mathsf{r}_w$ ou $H_0 = \mathsf{k}^{\star}_w(R,t)$ tels que $H_0 \in \mathcal{H}(S)$. Nous prouvons ce résultat par induction sur l'arbre de preuve témoignant du fait que $H_0 \in \mathcal{H}(S)$. Nous procédons par distinction de cas sur la dernière règle appliquée pour dériver H_0.
Case CONSEQ. Il y a un statement :

$$f = \Big(H \Leftarrow B_1, \ldots, B_n\Big) \in S$$

et une substitution σ qui clôt f tels que $H_0 = H\sigma$ et $B_i\sigma \in \mathcal{H}(S)$. Sans perte de généralité nous supposons que $B_i = \mathsf{k}^{\star_i}_{w_i}(X_i, s_i)$. Par hypothèse d'induction, nous avons qu'il existe $R'_1, \ldots, R'_n$ tels que :
- $\mathsf{k}^{\star_i}_{w_i\sigma}(R'_i, s_i\sigma) \in \mathcal{H}(K_{\mathsf{solved}})$ pour $1 \leq i \leq n$; et
- $\mathsf{i}_{w_i\sigma}(X_i\sigma, R'_i) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$ pour $1 \leq i \leq n$.

Soit σ' la substitution définie comme σ sauf qu'elle instancie X_i en R'_i pour tout $1 \leq i \leq n$. Nous montrerons que $H\sigma' \in \mathcal{H}(K_{\mathsf{solved}})$. Comme K a été mis à jour par f, il y a deux cas différents :

1. Si $f \in K$, Nous concluons par le Lemme 9.
2. Sinon, nous avons que $f = \mathsf{k}^{\star}_u(S,x) \Leftarrow \mathcal{B}$ avec $\mathsf{k}^{\star'}_{u'}(X,x) \in \mathcal{B}$, et dans un tel cas nous avons que $h = \big(\mathsf{i}_u(S,X) \Leftarrow \mathcal{B}\big)$ in K_{solved}. De plus, par définition d'un statement (de la graine), nous avons que $\star \geq \star'$. Soit $R' = X\sigma'$. Puisque $\mathcal{B} = B_1, \ldots, B_n$, et $B_i\sigma' \in \mathcal{H}(K_{\mathsf{solved}})$ pour $1 \leq i \leq n$, nous avons que $\mathsf{i}_{u\sigma'}(S\sigma', X\sigma') \in \mathcal{H}(K_{\mathsf{solved}})$. Nous avons aussi que $\mathsf{k}^{\star'}_{u'\sigma'}(X\sigma', x\sigma') \in \mathcal{H}(K_{\mathsf{solved}})$, et de par la règle EXTEND, nous déduisons que :

   $$\mathsf{k}^{\star'}_{u\sigma'}(X\sigma', x\sigma') = \mathsf{k}^{\star'}_{u\sigma}(X\sigma', x\sigma) = \mathsf{k}^{\star'}_w(R', t) \in \mathcal{H}(K_{\mathsf{solved}}).$$

   Notons que, Dans le cas $\star \neq \star'$, puisque nous savons que $\star \geq \star'$, nous déduisons que $\star = +$ et $\star' = -$, et c'est pourquoi nous concluons que $\mathsf{k}^{\star}_w(R',t) \in \mathcal{H}(K_{\mathsf{solved}})$ en se basant sur la règle WEAKENING. Puisque $\mathsf{i}_{w_i\sigma}(X_i\sigma, R'_i) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$ pour $1 \leq i \leq n$, nous avons aussi de par la règle EXT que $\mathsf{i}_{u\sigma}(X_i\sigma, R'_i) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$ pour $1 \leq i \leq n$, et donc $\mathsf{i}_{u\sigma}(S\sigma, S\sigma') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$. Notons que $u\sigma' = u\sigma$, et c'est pourquoi de par la règle TRANS, nous avons que :

   $$\mathsf{i}_{u\sigma}(S\sigma, X\sigma') = \mathsf{i}_w(R, R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}}).$$

Cas EXTEND. Dans un tel cas, nous avons que $H_0 = \mathsf{k}^{\star}_w(R,t)$ et $\mathsf{k}^{\star}_u(R,t) \in \mathcal{H}(S)$ pour un préfixe u de w, dans ce cas par hypothèse d'induction nous savons qu'il existe R' tel que $\mathsf{k}^{\star}_u(R',t) \in \mathcal{H}(K_{\mathsf{solved}})$ et $\mathsf{i}_u(R,R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$. Nous concluons que $\mathsf{k}^{\star}_w(R',t) \in \mathcal{H}(K_{\mathsf{solved}})$ et $\mathsf{i}_w(R,R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$ en appliquant les règles EXTEND et EXT.

Cas WEAKENING. Dans un tel cas, nous avons que $H_0 = \mathsf{k}^{+}_w(R,t)$ et $\mathsf{k}^{-}_w(R,t) \in \mathcal{H}(S)$. Par hypothèse d'induction, nous avons qu'il existe R' tel que $\mathsf{k}^{-}_w(R',t) \in \mathcal{H}(K_{\mathsf{solved}})$ et $\mathsf{i}_w(R,R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$. Nous concluons que $\mathsf{k}^{+}_w(R',t) \in \mathcal{H}(K_{\mathsf{solved}})$ en appliquant la règle WEAKENING. □

Nous sommes maintenant capable d'établir la complétude énoncée dans le Théorème 2.

La Proposition 3 articule les lemmes et la proposition précédents afin de prouver la partie complétude du théorème 2.

Proposition 3 *Soit T une trace close et K une base de connaissance telle que $K \in \mathsf{sat}(K_{\mathsf{init}}(T))$. Si $(T, \emptyset) \xrightarrow{l_1,\ldots,l_n}\!\!\!\!\!\rightarrow (Q, \varphi)$ alors nous avons que :*

1. $\mathsf{r}_{l_1\downarrow,\ldots,l_n\downarrow} \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$;
2. *si* $\varphi \vdash_R^+ t$ *alors* $\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(R, t\downarrow) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$;
3. *si* $\varphi \vdash_R^+ t$ *et* $\varphi \vdash_{R'}^+ t$, *alors* $\mathsf{i}_{l_1\downarrow,\ldots,l_n\downarrow}(R, R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$.

Preuve. Nous prouvons chaque point séparément.

1. De par le Théorème 1, nous avons que $\mathsf{r}_{l_1\downarrow,\ldots,l_n\downarrow} \in \mathcal{H}(\mathsf{seed}(T))$. Donc, de par la Proposition 2, nous savons que $\mathsf{r}_{l_1\downarrow,\ldots,l_n\downarrow} \in \mathcal{H}(K_{\mathsf{solved}})) \subseteq \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$. Ceci nous permet de conclure.

2. De par le Théorème 1, nous avons que $\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(R, t\downarrow) \in \mathcal{H}(S)$. C'est pourquoi, de par la Proposition 2, nous savons qu'il existe R' tel que :

$$\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(R', t\downarrow) \in \mathcal{H}(K_{\mathsf{solved}}) \text{ et } \mathsf{i}_{l_1\downarrow,\ldots,l_n\downarrow}(R, R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}}).$$

Nous permettant de conclure que $\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(R, t\downarrow) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$.

3. Du point précédant, nous savons que $\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(R, t\downarrow) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$ et $\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(R', t\downarrow) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$. Cela signifie qu'il existe S et S' tels que :

— $\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(S, t\downarrow) \in \mathcal{H}(K_{\mathsf{solved}})$ et $\mathsf{i}_{l_1\downarrow,\ldots,l_n\downarrow}(R, S) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$;

— $\mathsf{k}^+_{l_1\downarrow,\ldots,l_n\downarrow}(S', t\downarrow) \in \mathcal{H}(K_{\mathsf{solved}})$ et $\mathsf{i}_{l_1\downarrow,\ldots,l_n\downarrow}(R', S') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$.

Appliquant le Lemme 10, nous déduisons que $\mathsf{i}_w(S, S') \in \mathcal{H}(K_{\mathsf{solved}})$ pour un w préfixe de $l_1\downarrow, \ldots, l_n\downarrow$, et nous concluons que $\mathsf{i}_{l_1\downarrow,\ldots,l_n\downarrow}(S, S') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$ de par la règle Ext de la définition de $\mathcal{H}_{\mathsf{e}}$. Maintenant, se basant sur la règle Trans de la définition de $\mathcal{H}_{\mathsf{e}}$, nous concluons que $\mathsf{i}_{l_1\downarrow,\ldots,l_n\downarrow}(R, R') \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$. □

4.4 Procédure

Dans cette section, nous discutons de l'effectivité et de la terminaison de la procédure de saturation. Ensuite, nous décrivons notre algorithme pour résoudre le problème d'atteignabilité avec des actions **guess** comme décrit à la fin de la Section 3.4. Nous discutons de l'intégration de notre algorithme dans l'outil de vérification AKISS dans le Chapitre 5.

4.4.1 Effectivité de la procédure de saturation

Nous discutons ici des problèmes pour calculer une base de connaissance saturée effective et nous discutons aussi de la terminaison de la procédure de saturation.

Un premier problème vient du fait que $K_{\mathsf{init}}(T)$ est infini. En effet, l'ensemble $\mathsf{seed}(T)$ pour une trace close T est infini parce que $\mathcal{N}_{\mathsf{pub}}$ et $\mathcal{N}_{\mathsf{guess}}$ contiennent un ensemble infini de noms. Nous suivons [CcCK12] pour surmonter cette difficulté. Ainsi, nous considérons seulement les noms apparaissant dans T pour sa saturation ; les clauses représentant les noms n'apparaissant pas dans T n'influencent pas la saturation des autres clauses.

Lemme 11 *Soit T une trace close, $\mathcal{N}^T_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$ (resp. $\mathcal{N}^T_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$) l'ensemble fini de noms publics (resp. noms faibles) apparaissant dans T, et la base de connaissance saturée $K^0 \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}^T_{\mathsf{pub}}, \mathcal{N}^T_{\mathsf{guess}})))$. Nous avons que $K \subseteq K^0 \cup \mathcal{S}$ pour $K \in \mathsf{sat}(K_{\mathsf{init}}(T))$ et un ensemble $\mathcal{S}$ contenant des statements de la forme :*

- $\mathsf{k}^-_\epsilon(n, n) \Leftarrow$ *pour tout* $n \in \mathcal{N}_{\mathsf{pub}}$ *;*
- $\mathsf{k}^+_\epsilon(g, g) \Leftarrow$ *pour tout* $g \in \mathcal{N}_{\mathsf{guess}}$ *;*
- $\mathsf{i}_\epsilon(n, n) \Leftarrow$ *pour tout* $n \in \mathcal{N}_{\mathsf{pub}} \cup \mathcal{N}_{\mathsf{guess}}$ *;*
- $\mathsf{ri}_u((R_1, \ldots, R_k), (R'_1, \ldots, R'_k)) \Leftarrow \mathcal{B}$ *où* $(R_{i_0}, R'_{i_0}) = (n, n)$ *pour* $1 \leq i_0 \leq k$, *et* $n \in \mathcal{N}_{\mathsf{pub}} \uplus \mathcal{N}_{\mathsf{guess}}$.

Preuve. Observons d'abord que tout nom $n \in \mathcal{N}_{\mathsf{pub}} \setminus \mathcal{N}^T_{\mathsf{pub}}$ (respectivement $g \in \mathcal{N}_{\mathsf{guess}} \setminus \mathcal{N}^T_{\mathsf{guess}}$) n'apparait pas dans la graine, sauf dans le statement $\mathsf{k}^-_\epsilon(n, n) \Leftarrow$ (respectivement $\mathsf{k}^+_\epsilon(g, g) \Leftarrow$). C'est pourquoi, le statement $\mathsf{k}^-_\epsilon(n, n) \Leftarrow$ (respectivement $\mathsf{k}^+_\epsilon(g, g) \Leftarrow$) n'est jamais utilisé dans la règle Resolution de la saturation, et tout statement de connaissance dans $\mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}_{\mathsf{pub}}, \mathcal{N}_{\mathsf{guess}})))$ est aussi dans $\mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}^T_{\mathsf{pub}}, \mathcal{N}^T_{\mathsf{guess}}))) \cup \mathcal{S}$. De la même manière, comme n et g s'unifient uniquement entre eux (ou avec une variable, mais ce cas est interdit par la bonne forme des statements) ils déclenchent uniquement la règle Equation pour produire des statements $\mathsf{i}(n, n) \Leftarrow$ et $\mathsf{i}(g, g) \Leftarrow$. Cependant, un statement de ce type est aussi inclut dans $\mathcal{S}$. Finalement, Si des statements $\mathsf{i}(n, n) \Leftarrow$ ou $\mathsf{i}(g, g) \Leftarrow$ sont utilisés pour déclencher la règle Test, nous obtenons un statement résultat déjà inclus dans $\mathcal{S}$. □

La procédure de saturation peut elle même ne pas terminer même si la base de connaissance initiale est finie. Comme montré dans [CcCK12] nous conjecturons que notre procédure de saturation termine pour la classe de théories équationnelles sous-termes convergentes. Plus important, nous avons implémenté notre procédure et l'avons testée sur de nombreux exemples considérant de nombreuses théories équationnelles qui ne sont pas sous-termes convergents et notre outil a terminé sur tous ces exemples pratiques que nous avons testés.

4.4.2 Description, correction et complétude de l'algorithme

Notre procédure est décrite dans la Figure 1. Soit T une trace close et $K^0 \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}^T_{\mathsf{pub}}, \mathcal{N}^T_{\mathsf{guess}})))$ où $\mathcal{N}^T_{\mathsf{pub}}$ (resp. $\mathcal{N}^T_{\mathsf{guess}}$) est l'ensemble contenant tous les noms publics (resp. noms faibles) apparaissant dans T. Dans notre procédure, T est représenté par l'ensemble K^0_{solved} de statements résolus de K^0.

Le test $\mathsf{Reach}(T, K^0_{\mathsf{solved}})$ vérifie s'il existe une identité faiblement atteignable de la forme : $\mathsf{ri}_{\ell_1,\ldots,\ell_n}(\overrightarrow{R}, \overrightarrow{R'}) \Leftarrow \mathcal{B}$ dans K^0_{solved} qui correspondrait à une exécution réelle si l'on remplace les variables restantes par des constantes fraiches. Pour cela nous allons vérifier que les actions **guess** sont exécutables (dans la sémantique forte $\xrightarrow{\ell}$), nous vérifions si toutes les recettes $R_j\omega$ et $R'_j\omega$ témoignent du fait que la $j^{\text{ème}}$ action **guess**, **guess**(g) est en effet réalisable, c'est à dire que le nom faible g en paramètre peut être deviné avec la connaissance de l'attaquant à ce moment précis de l'exécution. Nous vérifions également si les tests négatifs sont satisfaits. Si tous ces tests réussissent alors T est atteignable.

Algorithme 1 Procédure pour vérifier l'atteignabilité

Procédure $\mathsf{Reach}(T,\, K^0_{\mathsf{solved}})$
$n \leftarrow$ le nombre d'actions de T.
Pour tout $\mathsf{ri}_{\ell_1,\ldots,\ell_n}(\overrightarrow{R}, \overrightarrow{R'}) \Leftarrow \mathsf{k}^{\star_1}_{w_1}(X_1, x_1), \ldots, \mathsf{k}^{\star_m}_{w_m}(X_m, x_m) \in K^0_{\mathsf{solved}}$ **faire :**
$c_1, \ldots, c_k$ des noms frais de $\mathcal{N}_{\mathsf{pub}}$ tels que :
$\sigma : \mathrm{vars}(\ell_1, \ldots, \ell_n) \cup \{x_1, \ldots, x_m\} \to \{c_1, \ldots, c_k\}$ est une bijection.
$\omega = \{X_i \mapsto x_i\sigma \mid 1 \leq i \leq m\}$.
Vérifie si $(T, \emptyset) = (T_0, \phi_0) \xrightarrow{\ell_1\sigma} \ldots (T_{n-1}, \phi_{n-1}) \xrightarrow{\ell_n\sigma} (T_n, \phi_n) = (\emptyset, \phi)$
On utilise les recettes $R_j\omega$ et $R'_j\omega$ comme un témoin pour deviner le nom faible de la $j^{\text{ème}}$ action **guess**.
Si cette exécution est valide, retourner **Vrai**.
fin Pour
Retourner **Faux**.
fin Procédure

Théorème 3 *Soit T une trace close, $\mathcal{N}^T_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$ (resp. $\mathcal{N}^T_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$) l'ensemble fini de noms publics (resp. noms faibles) qui apparaissent dans T. Soit la base de connaissance saturée $K^0 \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}^T_{\mathsf{pub}}, \mathcal{N}^T_{\mathsf{guess}})))$.*

Nous avons que T est atteignable si et seulement si $\mathsf{Reach}(T,\, K^0_{\mathsf{solved}})$ est vrai.

La partie de ce théorème sur la correction est une conséquence évidente de notre algorithme. La complétude demande plus de précisions. Elle est basée sur le Théorème 2 mais du travail reste à faire afin d'assurer que les tests négatifs seront satisfait. Plus important, nous devons vérifier que suffisamment de prédicats ri seront générés et testés. Ceci est nécessaire afin de nous assurer que quand un nom faible peut être deviné, alors notre algorithme trouvera un test témoignant de ce fait. Intuitivement notre procédure générera seulement des petits tests et nos résultats de complétude montrent qu'ils sont en effet suffisants pour trouver une attaque. Notons que nous devons considérer tous les tests (un par action **guess**) ensemble, et cela peut demander de réaliser des instanciations qui ne seront pas nécessaires quand nous ne considérons qu'une seule action **guess**. Cela vient du fait que l'attaquant peut avoir besoin d'émettre des messages qui peuvent être complexes pour s'assurer que toutes les données faibles peuvent être devinées quand nécessaire (sur la même trace).

4.4.3 Preuve de la correction et de la complétude de l'algorithme

La correction de l'algorithme 1 est triviale car vérifiée pendant la procédure. Les lemmes et propositions suivants s'attachent donc à montrer la complétude de l'algorithme.

Étant donné une frame ϕ et $g \in \mathcal{N}_{\mathsf{guess}}$ tels que g est devinable de ϕ. Nous écrivons $\phi \Vdash_{M,N} g$ pour exhiber les recettes M et N qui témoignent de ce fait.

Le Lemme 12 va s'attacher à montrer que si un nom faible est devinable, alors il existe alors il existe une identité exhibant ce fait dans $\mathcal{H}(K_{\mathsf{solved}})$. La difficulté ici est que ce nom peut être devinable avec les recettes M' et N' en témoin, et nous retrouverons l'identité témoignant de ce fait dans $\mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$ qui ne sera pas telle

quelle dans $\mathcal{H}(K_{\mathsf{solved}})$. Nous devons donc montrer qu'il existe toujours une autre paire de recette témoignant de ce fait dans $\mathcal{H}(K_{\mathsf{solved}})$.

Lemme 12 *Soit T une trace close et (T', φ) tel que $(T, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_n}\!\!\!\!\!\rightarrow (T', \varphi)$, et $g \in \mathcal{N}_{\mathsf{guess}}$ tel que g est devinable de φ. Soit $K = \mathsf{sat}(K_{\mathsf{init}}(T))$. Nous avons que $\mathsf{i}_w(M, N) \in \mathcal{H}(K_{\mathsf{solved}})$ pour des recettes M, N, et un $w \sqsubseteq \ell_1\downarrow, \ldots, \ell_n\downarrow$, et $\varphi \Vdash_{M,N} g$.*

Preuve. Puisque g est devinable de ϕ, nous savons qu'il existe M et N tels que $\phi \Vdash_{M,N} g$. Donc, nous avons que :

1. $(M = N)\phi$; et
2. $(M\{g \mapsto g'\} \neq N\{g \mapsto g'\})\phi$ pour un nom frais g'.

Donc $\phi \vdash^+_M t$ et $\phi \vdash^+_N t$ pour un terme t, et de par la complétude exprimée dans le Théorème 2, nous avons que $\mathsf{i}_{\mathsf{w}}(M, N) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$ où $\mathsf{w} = \ell_1\downarrow, \ldots, \ell_n\downarrow$. Nous montrerons que cela implique que $\mathsf{i}_{\mathsf{w}'}(M', N') \in \mathcal{H}(K_{\mathsf{solved}})$, pour w', M', et N' tels que $\mathsf{w}' \sqsubseteq \mathsf{w}$ et $\phi \Vdash_{M',N'} g$.

Ainsi, nous procédons par induction sur l'arbre de preuve témoignant du fait que $\mathsf{i}_{\mathsf{w}}(M, N) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$.

Cas de base : Notons que le cas de la réflexivité est exclus puisque nous savons que $\phi \Vdash_{M,M} g$. Donc il nous reste seulement le cas trivial où $\mathsf{i}_{\mathsf{w}}(M, N) \in \mathcal{H}(K_{\mathsf{solved}})$.

Cas inductifs : Nous procédons par analyse de cas sur la dernière règle appliquée pour dériver l'arbre de preuve.

SYM
$$\frac{\mathsf{i}_{\mathsf{w}}(N, M) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})}{\mathsf{i}_{\mathsf{w}}(M, N) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})}$$

Puisque $\phi \Vdash_{M,N} g$ nous avons que $\phi \Vdash_{N,M} g$ et, appliquant notre hypothèse d'induction sur l'arbre de preuve pour $\mathsf{i}_{\mathsf{w}}(N, M)$, nous pouvons conclure.

TRAN
$$\frac{\mathsf{i}_{\mathsf{w}}(M, M_1),\ \mathsf{i}_{\mathsf{w}}(M_1, N) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})}{\mathsf{i}_{\mathsf{w}}(M, N) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})}$$

De par la correction exprimée dans le Théorème 2, nous avons que $T \models \mathsf{i}_{\mathsf{w}}(M, M_1)$ et $T \models \mathsf{i}_{\mathsf{w}}(M_1, N)$. D'où, si $(T, \emptyset) \xrightarrow{\ell_1,\ldots,\ell_{|\mathsf{w}|}}\!\!\!\!\!\rightarrow (T', \phi)$, nous avons que $(M = M_1)\phi$ et $(M_1 = N)\phi$. Maintenant supposons que $(M\{g \mapsto g'\} = M_1\{g \mapsto g'\})\phi$ et $(M_1\{g \mapsto g'\} = N\{g \mapsto g'\})\phi$. Cela impliquerait que $(M\{g \mapsto g'\} = N\{g \mapsto g'\})\phi$, et amènerait à une contradiction. D'où, nous avons que soit $(M\{g \mapsto g'\} \neq M_1\{g \mapsto g'\})\phi$ ou $(M_1\{g \mapsto g'\} \neq N\{g \mapsto g'\})\phi$, ce qui implique que soit $\phi \Vdash_{M_1,N} g$ or $\phi \Vdash_{M_1,N} g$. Donc, nous pouvons appliquer notre hypothèse d'induction sur l'arbre de preuve nous permettant de dériver $\mathsf{i}_{\mathsf{w}}(M, M_1)$ ou $\mathsf{i}_{\mathsf{w}}(M_1, N)$, et conclure.

EXT
$$\frac{\mathsf{i}_{\mathsf{w}'}(M, N) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}}),\ \mathsf{w}' \sqsubseteq \mathsf{w}}{\mathsf{i}_{\mathsf{w}}(M, N) \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})}$$

Nous concluons par application de notre hypothèse d'induction sur l'arbre de preuve nous permettant de dériver $\mathsf{i}_{\mathsf{w}'}(M, N)$, et la relation de transitivité de la relation $\sqsubseteq$.

CONG

$$\frac{\mathsf{i_w}(M_1, N_1) \dots \mathsf{i_w}(M_k, N_k) \in \mathcal{H}_\mathsf{e}(K_\mathsf{solved})}{\mathsf{i_w}(\mathsf{f}(M_1, \dots, M_k), \mathsf{f}(N_1, \dots, N_k)) \in \mathcal{H}_\mathsf{e}(K_\mathsf{solved})}$$

Soit $M = \mathsf{f}(M_1, \dots, M_k)$ et $N = \mathsf{f}(N_1, \dots, N_k)$. Nous avons, de la correction exprimée dans le Théorème 2, que $(M_1 = N_1)\phi, \dots, (M_k = N_k)\phi$. Pour conclure en sa basant sur notre hypothèse d'induction, nous devons montrer qu'il existe j tel que $(M_j\{g \mapsto g'\} \neq N_j\{g \mapsto g'\})\phi$. Par contradiction, nous supposons que pour tout j tel que $1 \leq j \leq k$, nous avons que $(M_j\{g \mapsto g'\} = N_j\{g \mapsto g'\})\phi$. Donc, nous avons que $(M\{g \mapsto g'\} = N\{g \mapsto g'\})\phi$, ce qui nous amène à une contradiction.

D'où, nous avons qu'il existe M', N', et $\mathsf{w}' \sqsubseteq \mathsf{w}$ tels que $\mathsf{i_{w'}}(M', N') \in \mathcal{H}(K_\mathsf{solved})$ et $\varphi \Vdash_{M',N'} g$. □

Dans le Lemme 13 nous montrons sous quelles conditions nous pouvons avoir un statements de type ri dans $\mathcal{H}(K_\mathsf{solved})$.

Lemme 13 *Soit K une base de connaissance saturée telle que :*
— $\mathsf{r}_u \in \mathcal{H}(K_\mathsf{solved})$ *avec* $|\mathcal{G}^\ell(u)| = n$ *; et*
— $\mathsf{i}_{u_i}(R_i, R'_i) \in \mathcal{H}(K_\mathsf{solved})$ *avec* $u_i \sqsubseteq u$ *et* $|\mathcal{G}^\ell(u_i)| < i$ *pour* $1 \leq i \leq n$.
Nous avons $\mathsf{ri}_u(\overrightarrow{R}, \overrightarrow{R'}) \in \mathcal{H}(K_\mathsf{solved})$ *où* $(\overrightarrow{R}, \overrightarrow{R'}) = ((R_1, \dots, R_n), (R'_1, \dots, R'_n))$.

Preuve. Comme $\mathsf{r}_u \in \mathcal{H}(K_\mathsf{solved})$, par la définition de $\mathcal{H}$, il existe un statement résolu $g = (\mathsf{r}_v \Leftarrow \mathcal{B}) \in K_\mathsf{solved}$ et une substitution σ qui clôt g tel que $\forall b \in \mathcal{B}, b\sigma \in \mathcal{H}(K_\mathsf{solved})$ et tel que $u = v\sigma$. Comme $\mathsf{i}_{u_1}(R_1, R'_1), \dots, \mathsf{i}_{u_k}(R_k, R'_k) \in \mathcal{H}(K_\mathsf{solved})$ il existe, par la définition de $\mathcal{H}$, statements résolus $f_1 = (\mathsf{i_{w_1}}(T_1, T'_1) \Leftarrow \mathcal{B}_1), \dots, f_k = (\mathsf{i_{w_k}}(T_k, T'_k) \Leftarrow \mathcal{B}_k)$ et les substitutions $\tau_1, \dots, \tau_k$ qui closent respectivement $f_1, \dots, f_k$ tels que $\forall b \in \mathcal{B}_i\tau_i, b \in \mathcal{H}(K_\mathsf{solved})$ et $u_i = \mathsf{w}_i\tau_i$, $R_i = T_i\tau_i$ et $R'_i = T'_i\tau_i$, pour tout $i, 1 \leq i \leq k$.

Puisque $\mathsf{w}_i\tau_i = u_i$ et $u_i \sqsubseteq u$, $1 \leq i \leq k$, il existe un unificateur le plus général ω tel que $\omega = \mathrm{mgu}(v, \mathsf{w}_1.x_1, \dots, \mathsf{w}_k.x_k)$.

Soit π défini tel que $(\sigma \cup \tau_1 \cup \dots \cup \tau_k) = \omega\pi$.

Comme la base de connaissance est saturée la règle de saturation TEST doit s'être déclenchée pour $f_1, \dots, f_k$ et g. C'est pourquoi K doit avoir été mis à jour avec h où

$$h = (\mathsf{ri}_v(\overrightarrow{T}, \overrightarrow{T'}) \Leftarrow \mathcal{B}, \mathcal{B}_1, \dots, \mathcal{B}_k)\omega$$

Mais comme h n'est pas un fait de déduction, la mise à jour doit avoir ajouté simplement h à K et donc $h \in K$.

Nous avons que $\forall b \in \mathcal{B}, b\omega\pi = b\sigma \in \mathcal{H}(K_\mathsf{solved})$ et que $\forall b \in \mathcal{B}_i, 1 \leq i \leq k, b\omega\pi = b\tau_i \in \mathcal{H}(K_\mathsf{solved})$ pour tout $1 \leq i \leq k$. En appliquant le Lemme 9 au statement h et la substitution π, nous obtenons que $\mathsf{ri}_{v\omega\pi}(\overrightarrow{T}\omega\pi, \overrightarrow{T'}\omega\pi) = \mathsf{ri}_u(\overrightarrow{R}, \overrightarrow{R'}) \in \mathcal{H}(K_\mathrm{solved})$. □

Dans le Lemme 14 nous relevons l'exécution abstraite où nous avons introduit des constantes à la place des variables, en l'exécution concrète que nous avions à l'origine.

Lemme 14 *Soit T une trace close, et $\{c_1, \dots, c_r\}$ les noms qui n'apparaissent pas dans T. Soit φ_0 la frame vide. Si*

$$(T_0, \varphi_0) \xrightarrow{\ell_1}\!\!\!\!\rightarrow (T_1, \varphi_1) \xrightarrow{\ell_2}\!\!\!\!\rightarrow \dots \xrightarrow{\ell_n}\!\!\!\!\rightarrow (T_n, \varphi_n)$$

et

$$(T_0, \varphi_0) \xrightarrow{\ell'_1}\!\!\!\!\rightarrow (T'_1, \varphi'_1) \xrightarrow{\ell'_2}\!\!\!\!\rightarrow \ldots \xrightarrow{\ell'_n}\!\!\!\!\rightarrow (T'_n, \varphi'_n)$$

tel que $\ell'_1\!\downarrow, \ldots, \ell'_n\!\downarrow = (\ell_1\!\downarrow, \ldots, \ell_n\!\downarrow)\pi\!\downarrow$ *pour une substitution* π *qui remplace les constantes* $\{c_1, \ldots, c_r\}$ *par des messages, alors nous avons que* $T'_n\!\downarrow = T_n\pi\!\downarrow$ *et* $\varphi'_n\!\downarrow = \varphi_n\pi\!\downarrow$.

Preuve. Nous montrons ce résultat par induction sur la taille de la dérivation Notons que, quand $n = 0$, le résultat est satisfait trivialement C'est pourquoi, nous supposons que $T'_{n-1}\!\downarrow = T_{n-1}\pi\!\downarrow$ et $\varphi'_{n-1}\!\downarrow = \varphi_{n-1}\pi\!\downarrow$, et nous montrons que $T'_n\!\downarrow = T_n\pi\!\downarrow$ et $\varphi'_n\!\downarrow = \varphi_n\pi\!\downarrow$. Par hypothèse, nous savons que $(T_{n-1}, \varphi_{n-1}) \xrightarrow{\ell_n}\!\!\!\!\rightarrow (T_n, \varphi_n)$, $(T'_{n-1}, \varphi'_{n-1}) \xrightarrow{\ell'_n}\!\!\!\!\rightarrow (T'_n, \varphi'_n)$, et $\ell'_n\!\downarrow = (\ell_n\!\downarrow)\pi\!\downarrow$. Nous procédons par distinction de cas sur l'action ℓ_n.
Cas $\ell_n = \mathbf{out}(c)$ *pour un* c *quelconque.* Dans un tel cas, nous avons que $T_{n-1} = \mathbf{out}(c, t).U$ et $T'_{n-1} = \mathbf{out}(c, t').U'$ pour t, t', U, et U' tels que $t'\!\downarrow = t\pi\!\downarrow$ et $U'\!\downarrow = U\pi\!\downarrow$. Nous avons que :

— $(T_n, \varphi_n) = (U, \varphi_{n-1} \cup \{\{w_{|\operatorname{dom}(\phi_{n-1})|+1} \mapsto t\}\})$,
— $(T'_n, \varphi'_n) = (U', \varphi'_{n-1} \cup \{\{w_{|\operatorname{dom}(\phi'_{n-1})|+1} \mapsto t'\}\})$.

Il est facile de voir que $T'_n\!\downarrow = T_n\pi\!\downarrow$ et $\varphi'_n\!\downarrow = \varphi_n\pi\!\downarrow$.
Cas $\ell_n = \mathbf{in}(c, t)$ *pour* c *et* t *quelconques.* Dans un tel cas, nous avons que $T_{n-1} = \mathbf{in}(c, x).U$ et $T'_{n-1} = \mathbf{in}(c, x).U'$ pour x, U, et U' tels que $U'\!\downarrow = U\pi\!\downarrow$. Soit $\ell'_n = \mathbf{in}(c, t')$. Par hypothèse, nous savons que $t'\!\downarrow = (t\!\downarrow)\pi\!\downarrow$. Nous avons que :

— $(T_n, \varphi_n) = (U\{x \mapsto t\!\downarrow\}, \varphi_{n-1})$,
— $(T'_n, \varphi'_n) = (U'\{x \mapsto t'\!\downarrow\}, \varphi'_{n-1})$.

C'est pourquoi, nous avons que :

— $\varphi_n\pi\!\downarrow = \varphi_{n-1}\pi\!\downarrow = \varphi'_{n-1}\!\downarrow = \varphi'_n\!\downarrow$;
— $T_n\pi\!\downarrow = ((U\pi)\!\downarrow)\{x \mapsto (t\!\downarrow)\pi\!\downarrow\}\!\downarrow = T'_n\!\downarrow$.

Cas $\ell_n = \mathbf{test}^{=}$ *ou* $\mathbf{test}^{\neq}$. Dans un tel cas, nous concluons facilement en se basant sur hypothèse d'induction.
Cas $\ell_n = \mathsf{guess}(g)$ *pour un* g *quelconque.* De la même manière, nous concluons facilement en se basant sur notre hypothèse d'induction. □

Dans le Lemme 15 nous montrons un invariant sur les variables de premier ordre dans les statements.

Lemme 15 *Soit* T *une trace close et* K *une base de connaissance saturée associée à* T. *Pour tout statement* $f \in K$, *nous avons que :*

1. *Si* $f = \left(\mathsf{k}^{\star}_{\ell_1,\ldots,\ell_n}(R, t) \Leftarrow \{\mathsf{k}^{\star_i}_{w_i}(X_i, t_i)\}_{1\leq i\leq m}\right)$ *et* $x \in \operatorname{vars}(t)$
   alors $x \in \operatorname{vars}(t_1, \ldots, t_m)$.
2. *Si* $f = \left(\mathsf{r}_{\ell_1,\ldots,\ell_n} \Leftarrow \{\mathsf{k}^{\star_i}_{w_i}(X_i, t_i)\}_{1\leq i\leq m}\right)$ *et* $x \in \operatorname{vars}(\ell_k)$
   alors il existe $w_j = \ell_1, \ldots, \ell_{k'}$ *avec* $k' < k$ *tel que* $x \in \operatorname{vars}(t_j)$.
3. *Si* $f = \left(\mathsf{ri}_{\ell_1,\ldots,\ell_n}(\overrightarrow{S}, \overrightarrow{S'}) \Leftarrow \{\mathsf{k}^{\star_i}_{w_i}(X_i, t_i)\}_{1\leq i\leq m}\right)$ *et* $x \in \operatorname{vars}(\ell_k)$
   alors il existe $w_j = \ell_1, \ldots, \ell_{k'}$ *avec* $k' < k$ *tel que* $x \in \operatorname{vars}(t_j)$.

Preuve. Les statements de la graine satisfont ces propriétés, qui sont préservées par la mise à jour et la saturation. □

Ce Lemme 16 montre un invariant sur les variables des recettes.

Lemme 16 *Soit T une trace close et K une base de connaissance saturée associée à T. Pour tout statement $f \in K$, nous avons que :*

- *Si $f = \big(\mathsf{k}^{\star}_{w}(R,t) \Leftarrow \mathsf{k}^{\star_1}_{w_1}(X_1,t_1),\ldots,\mathsf{k}^{\star_m}_{w_m}(X_m,t_m)\big)$ alors $\mathrm{vars}(R)\cap\mathcal{Y} \subseteq \{X_1,\ldots,X_m\}$.*
- *Si $f = \big(\mathsf{i}_{w}(R,R') \Leftarrow \mathsf{k}^{\star_1}_{w_1}(X_1,t_1),\ldots,\mathsf{k}^{\star_m}_{w_m}(X_m,t_m)\big)$ alors $\mathrm{vars}(\{R,R'\}) \cap \mathcal{Y} \subseteq \{X_1,\ldots,X_m\}$.*
- *Si $f = \big(\mathsf{ri}_{\ell_1,\ldots,\ell_n}(\overrightarrow{S},\overrightarrow{S'}) \Leftarrow \{\mathsf{k}^{\star_i}_{w_i}(X_i,t_i)\}_{1\leq i\leq m}\big)$ où $\overrightarrow{S} = (S_1,\ldots,S_k)$, $\overrightarrow{S'} = (S'_1,\ldots,S'_k)$, $p_1,\ldots,p_k$ sont les indices où les actions* **guess** *ont lieu dans $\ell_1,\ldots,\ell_n$, $i_0 \in \{1,\ldots,k\}$, et $X \in \mathrm{vars}(\{S_{i_0},S'_{i_0}\})$, alors il existe $j \in \{1,\ldots,m\}$ tel que $X = X_j$ et $|w_j| \leq p_{i_0}$.*

Preuve. Les statements de la graine satisfont cette propriété qui est préservée par la mise à jour et la saturation. □

Lemme 17 *Soit T une trace close et K une base de connaissance saturée associée à T. Pour tout statement $f \in K$, nous avons que :*

1. *Si $f = \big(\mathsf{k}^{\star}_{\ell_1,\ldots,\ell_n}(R,t) \Leftarrow \{\mathsf{k}^{\star_i}_{w_i}(X_i,t_i)\}_{1\leq i\leq m}\big)$ alors $R \notin \mathcal{Y}$. De plus, dans le cas où $\star = +$, nous avons que $\{X_i \mid \star_i = + \text{ avec } 1 \leq i \leq m\} \subseteq \mathrm{vars}(R)$.*
2. *Si $f = \big(\mathsf{i}_{\ell_1,\ldots,\ell_n}(R,R') \Leftarrow \{\mathsf{k}^{\star_i}_{w_i}(X_i,t_i)\}_{1\leq i\leq m}\big)$ avec $i \in \{1,\ldots,m\}$ tel que $\star_i = +$ et $X_i \in \mathrm{vars}(R,R')$, alors X_i est soit un sous-terme strict de R ou un sous-terme strict de R'.*
3. *Si $f = \big(\mathsf{ri}_{\ell_1,\ldots,\ell_n}(\overrightarrow{S},\overrightarrow{S'}) \Leftarrow \{\mathsf{k}^{\star_i}_{w_i}(X_i,t_i)\}_{1\leq i\leq m}\big)$ avec $\overrightarrow{S} = (S_1,\ldots,S_k)$ et $\overrightarrow{S'} = (S'_1,\ldots,S'_k)$, et $i \in \{1,\ldots,m\}$ tel que $\star_i = +$ et $X_i \in \mathrm{vars}(S_{i_0},S'_{i_0})$ pour $i_0 \in \{1,\ldots,k\}$, alors X_i est un sous-terme strict de S_{i_0} ou S'_{i_0}.*

Preuve. Les statements de la graine satisfont cette propriété. Ces propriétés sont satisfaites par la saturation.

Notons que l'opérateur de mise à jour peut introduire une variable de recette dans la tête du statement de l'identité résultante. Cependant, cela arrive seulement quand un statement de la forme $f = \mathsf{k}^{\star}_{u}(R,x) \Leftarrow \mathcal{B}$ avec $\mathsf{k}^{\star'}_{u'}(X,x) \in \mathcal{B}$ a été généré soit par la procédure de saturation ou est initialement présent dans la graine. Dans les deux cas, nous avons que $R \notin \mathcal{Y}$. Donc, considérant le statement résultant qui est de la forme

$$\mathsf{i}_{u}(R,X) \Leftarrow \mathcal{B}$$

il reste à montrer que la propriété est satisfaite pour X. La propriété est trivialement satisfaite quand $\star' = -$. Donc, nous considérons le cas où $\star' = +$. dans un tel cas, nous avons que $\star = +$, et de par la propriété additionnelle définie dans le point 1, nous savons que X est un sous-terme strict de R. Cela nous permet de conclure. □

Nous allons maintenant prouver la complétude de l'algorithme, exprimé dans le théorème 4 ci-dessous.

Théorème 4 *Soit T une trace close, et $K \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T)))$. Si T est atteignable alors* Reach*(T, K_{solved}) est satisfait.*

Preuve. Si T est atteignable alors il existe une exécution témoignant du fait :

$$(T,\emptyset) \xrightarrow{l_1} (T_1,\varphi_1) \ldots \xrightarrow{l_n} (T_n,\varphi_n) = (\emptyset,\varphi).$$

Soit k le nombre de **guess** dans cette trace d'exécution, et $p_1, \ldots, p_k$ les indices où ces actions **guess** apparaissent dans cette trace. En se basant sur la complétude du Théorème 2 nous avons que :

$$\mathsf{r}_{l_1\downarrow,\ldots,l_n\downarrow} \in \mathcal{H}_{\mathsf{e}}(K_{\mathsf{solved}})$$

Par la définition de $\mathcal{H}_{\mathsf{e}}$, nous avons qu'elle ne contient pas de statements d'atteignabilité de plus que ceux dans $\mathcal{H}$. Donc $\mathsf{r}_{l_1\downarrow,\ldots,l_n\downarrow} \in \mathcal{H}(K_{\mathsf{solved}})$. En se basant sur le Lemme 12, nous avons que :

$$\mathsf{i}_{l_1\downarrow,\ldots,l_{p'_i}\downarrow}(R_{p_i}, R'_{p_i}) \in \mathcal{H}(K_{\mathsf{solved}})$$

pour R_{p_i}, R'_{p_i} tels que $\varphi_{p_i-1} \Vdash_{R_{p_i},R'_{p_i}} g_i$ avec $p'_i < p_i$, et où $g_i \in \mathcal{N}_{\mathsf{guess}}$ est l'action **guess** apparaissant à la $p_i{}^{\text{ème}}$ action de la trace d'exécution. Donc, nous pouvons appliquer le Lemme 13, pour déduire que :

$$\mathsf{ri}_{l_1\downarrow,\cdots,l_n\downarrow}(\overrightarrow{R}, \overrightarrow{R'}) \in \mathcal{H}(K_{\mathsf{solved}})$$

où $\overrightarrow{R} = (R_{p_1}, \ldots, R_{p_k})$, et $\overrightarrow{R'} = (R'_{p_1}, \ldots, R'_{p_k})$. Par la définition de $\mathcal{H}$, nous savons qu'il existe un statement f dans K_{solved} :

$$f = \mathsf{ri}_{\ell_1,\ldots,\ell_n}(\overrightarrow{S}, \overrightarrow{S'}) \Leftarrow \mathsf{k}^{\star_1}_{w_1}(X_1, x_1), \ldots, \mathsf{k}^{\star_m}_{w_m}(X_m, x_m)$$

et une substitution τ telle que $\ell_i\tau = l_i\downarrow$ avec $1 \leq i \leq n$, $\overrightarrow{S}\tau = \overrightarrow{R}$, et $\overrightarrow{S'}\tau = \overrightarrow{R'}$; et $\mathsf{k}^{\star_j}_{w_j}(X_j, x_j)\tau \in \mathcal{H}(K_{\mathsf{solved}})$ avec $1 \leq j \leq m$.

Nous appelons $\alpha = \left(\overrightarrow{R}, \overrightarrow{R'}, f, \tau\right)$ un témoin et nous définissons sa taille $\mathsf{sz}(\alpha) = (s, s')$ où s, s' sont des entiers tels que :

— $s = \sum_{i=1}^{k}\{|R_i| \mid g_i \text{ apparait dans } R_i\} + \sum_{i=1}^{k}\{|R'_i| \mid g_i \text{ apparait dans } R'_i\}$; et
— $s' = \sum_{i=1}^{k}\{|R_i| \mid g_i \text{ n'apparait } \textit{pas} \text{ dans } R_i\} + \sum_{i=1}^{k}\{|R'_i| \mid g_i \text{ n'apparait } \textit{pas} \text{ dans } R'_i\}$.

Observons que $s + s' = \sum_{i=1}^{k}(|R_i| + |R'_i|)$. De plus, pour chaque test R_i, R'_i, notons que soit R_i ou R'_i contribuent à la mesure s. Supposons sans perte de généralité que nous choisissons le témoin $\alpha = \left(\overrightarrow{R}, \overrightarrow{R'}, f, \tau\right)$ dont sa taille est minimale dans l'ordre lexicographique.

Soit $\sigma_1 : \{x_1, \ldots, x_m\} \to \{c_1, \ldots, c_r\}$ une bijection. Soit σ_2 une substitution telle que $\mathrm{dom}(\sigma_2) = \{X_1, \ldots, X_m\}$, et $\sigma_2(X_j) = x_j\sigma_1$. D'après le Lemme 15 et le Lemme 16, $\sigma_1 \uplus \sigma_2$ substitution qui clôt f. Nous obtenons que :

$$\mathsf{ri}_{\ell_1\sigma_1,\ldots,\ell_n\sigma_1}(\overrightarrow{S}\sigma_2, \overrightarrow{S'}\sigma_2) \in \mathcal{H}(K_{\mathsf{solved}}).$$

Par le Théorème 2 (correction), il suit que :

$$(T, \emptyset) \xrightarrow{\ell_1\sigma_1} (T'_1, \varphi'_1) \xrightarrow{\ell_2\sigma_1} \ldots \xrightarrow{\ell_n\sigma_1} (T'_n, \varphi'_n) = (0, \varphi')$$

Soit $\tau = \tau_1 \uplus \tau_2$ où τ_1 (resp. τ_2) a le même domaine que σ_1 (resp. σ_2). Soit π_1 une substitution telle que $\mathrm{dom}(\pi_1) = \{c_1, \ldots, c_r\}$, et $\pi_1(c_j) = \tau_1(\sigma_1^{-1}(c_j))$. C'est pourquoi, nous avons les deux dérivations suivantes :

1. $(T, \emptyset) \xrightarrow{\ell_1\sigma_1}\!\!\!\!\rightarrow (T'_1, \varphi'_1) \xrightarrow{\ell_2\sigma_1}\!\!\!\!\rightarrow (T'_2, \varphi'_2) \ldots \xrightarrow{\ell_n\sigma_n}\!\!\!\!\rightarrow (T'_n, \varphi'_n) = (0, \varphi')$
2. $(T, \emptyset) \xrightarrow{l_1}\!\!\!\!\rightarrow (T_1, \varphi_1) \xrightarrow{l_2}\!\!\!\!\rightarrow (T_2, \varphi_2) \ldots \xrightarrow{l_n}\!\!\!\!\rightarrow (T_n, \varphi_n) = (0, \varphi)$.

Comme $l_i{\downarrow} = \ell_i\tau_1 = \ell_i\sigma_1\pi_1$ nous pouvons appliquer le Lemme 14, et nous obtenons que $T_i = T'_i\pi_1$ pour $1 \leq i \leq n$, et $\varphi = \varphi'\pi_1$.

Il reste à établir la validité de l'exécution 1 par rapport à $\rightarrow$, i.e.

$$(T, \emptyset) \xrightarrow{\ell_1\sigma_1} (T'_1, \varphi'_1) \ldots \xrightarrow{\ell_n\sigma_1} (T'_n, \varphi'_n) = (\emptyset, \varphi').$$

Pour cela nous avons besoin de montrer que les conditions de bord des actions de tests négatifs et actions **guess** sont satisfaites.

À propos des actions de tests négatifs, supposons que la $i$ème action dans la trace close T est un test négatif de la forme $\mathsf{a}_i = [s_i \neq t_i]$ qui est instanciée dans $[s_i\theta \neq t_i\theta]$ dans la dérivation 2 et dans $[s_i\theta' \neq t_i\theta']$ dans la dérivation 1. De plus, nous avons vu que $T_i = T'_i\pi_1$, donc nous avons que $s_i\theta = s_i\theta'\pi_1$ et $t_i\theta = t_i\theta'\pi_1$. Supposons par contradiction que $s_i\theta' =_{\mathsf{E}} t_i\theta'$ pour un i quelconque. Donc, nous avons que $s_i\theta'\pi_1 =_{\mathsf{E}} t_i\theta'\pi_1$, et donc $s_i\theta =_{\mathsf{E}} t_i\theta$. Puisque la dérivation 2 est une exécution valide par rapport à $\rightarrow$, nous savons que $s_i\theta{\downarrow} \neq t_i\theta{\downarrow}$. Nous obtenons une contradiction.

À propos des actions **guess**, nous avons besoin de montrer que :

$$(S_i\sigma_2\{g_i \mapsto g'_i\})\varphi'{\downarrow} \neq (S'_i\sigma_2\{g_i \mapsto g'_i\})\varphi'{\downarrow} \quad \text{pour } 1 \leq i \leq k.$$

Soit $i_0 \in \{1, \ldots, k\}$. Nous établissons tout d'abord la proposition suivante :
Proposition. Soit $\mathsf{k}^{\star_{j_1}}_{w_{j_1}}(X_{j_1}, x_{j_1})$ et $\mathsf{k}^{\star_{j_2}}_{w_{j_2}}(X_{j_2}, x_{j_2})$ deux hypothèses de f telles que $x_{j_1} = x_{j_2}$, et $|w_{j_1}|, |w_{j_2}| < p_{i_0}$. De plus, dans le cas où $\star_{j_1} = +$ (resp. $\star_{j_2} = +$), nous supposons que X_{j_1} (resp. X_{j_2}) apparait dans S_{i_0}, ou S'_{i_0}. Dans une tel cas nous avons que :

$$(X_{j_1}\tau\{g_{i_0} \mapsto g'_{i_0}\} = X_{j_2}\tau\{g_{i_0} \mapsto g'_{i_0}\})\varphi$$

Preuve de la proposition. Dans le cas où $\star_{j_1} = \star_{j_2} = -$, nous savons que g_{i_0} n'apparait ni dans $X_{j_1}\tau$ ni dans $X_{j_2}\tau$, et c'est pourquoi nous concluons facilement que

$$(X_{j_1}\tau\{g_{i_0} \mapsto g'_{i_0}\} = X_{j_2}\tau\{g_{i_0} \mapsto g'_{i_0}\})\varphi$$

Dans le cas où $+ \in \{\star_{j_1}, \star_{j_2}\}$, nous savons aussi que X_{j_1} ou X_{j_2} apparaissent dans S_{i_0} ou dans S'_{i_0}. Supposons par contradiction que :

$$(X_{j_1}\tau\{g_{i_0} \mapsto g'_{i_0}\} \neq X_{j_2}\tau\{g_{i_0} \mapsto g'_{i_0}\})\varphi$$

En se basant sur le Lemme 10, puisque $\mathsf{k}^{\star_{j_1}}_{w_{j_1}\tau}(X_{j_1}\tau, x_{j_1}\tau)$ et $\mathsf{k}^{\star_{j_2}}_{w_{j_2}\tau}(X_{j_2}\tau, x_{j_2}\tau)$ sont tous deux dans $\mathcal{H}(K_{\mathsf{solved}})$, nous avons que $\mathsf{i}_{w\tau}(X_{j_1}\tau, X_{j_2}\tau) \in \mathcal{H}(K_{\mathsf{solved}})$ pour $w \sqsubseteq w_{j_i}\tau$ avec $i = 1$ ou $i = 2$. Donc, nous pouvons appliquer le Lemme 13 considérant les mêmes éléments que ceux introduits au début de la preuve de notre théorème mais remplaçant $\mathsf{i}_{l_1{\downarrow},\ldots,l_{p'_{i_0}}{\downarrow}}(R_{p_{i_0}}, R'_{p_{i_0}}) \in \mathcal{H}(K_{\mathsf{solved}})$ par $\mathsf{i}_{w\tau}(X_{j_1}\tau, X_{j_2}\tau) \in \mathcal{H}(K_{\mathsf{solved}})$. Alors, par la définition de $\mathcal{H}$, nous obtiendrons qu'il existe un autre statement f' qui témoigne de l'atteignabilité. Comme $+ \in \{\star_{j_1}, \star_{j_2}\}$, par le Lemme 17 la taille de notre nouveau témoin est plus petite que la taille de notre témoin précédent. D'où, nous contredisons la minimalité du témoin que nous avons choisi précédemment ce qui conclut la preuve de cette proposition.
À partir de maintenant, supposons, sans perte de généralité, que pour f, nous avons que :

— si $|w_i| < p_{i_0} \leq |w_j|$ alors $i < j$;
— si $|w_i|, |w_j| < p_{i_0}$, $\star_i = -$, et $\star_j = +$ alors $i < j$;
— si $|w_i| < |w_j| < p_{i_0}$, $\star_i = -$, et $\star_j = -$ alors $i < j$;
— si $|w_i|, |w_j| < p_{i_0}$, $\star_i = +$, $\star_j = +$, $X_i \in \mathrm{vars}(S_{i_0}, S'_{i_0})$, et $X_j \notin \mathrm{vars}(S_{i_0}, S'_{i_0})$ alors $i < j$.

En d'autres mots, supposons que les hypothèses de f sont ordonnées en plaçant d'abord les hypothèse k^- en augmentant le monde (jusqu'à ce que l'on atteigne la taille p_{i_0}), et alors les hypothèses k^+ contribuent aux tests S_{i_0}, S'_{i_0}.

Soit τ_{2,i_0} la substitution telle que $\tau_{2,i_0}(X_i) = \tau_2(X_{least(i)})$ pour $1 \leq i \leq m$ où $least(j) = \min\{i \mid x_i = x_j\}$. En d'autres mots, nous changeons des recettes. Cependant, en se basant sur notre précédente proposition, nous préservons les égalités entre messages. C'est pourquoi, du Lemme 16, nous avons que :

— $(S_{i_0}\tau_2\{g_{i_0} \mapsto g'_{i_0}\} = S_{i_0}\tau_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi$, et d'où $(R_{i_0}\{g_{i_0} \mapsto g'_{i_0}\} = S_{i_0}\tau_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi$; et
— $(S'_{i_0}\tau_2\{g_{i_0} \mapsto g'_{i_0}\} = S'_{i_0}\tau_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi$, et d'où $(R_{i_0}\{g_{i_0} \mapsto g'_{i_0}\} = S'_{i_0}\tau_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi$.

Maintenant, nous supposons, par contradiction, que :

$$(S_{i_0}\sigma_2\{g_{i_0} \mapsto g'_{i_0}\})\varphi' =_{\mathsf{E}} (S'_{i_0}\sigma_2\{g_{i_0} \mapsto g'_{i_0}\})\varphi'.$$

Soit π^-_{1,i_0} qui est π_1 restreint aux constantes qui apparaissent dans $\ell_1\sigma_1, \ldots, \ell_{p_{i_0}}\sigma_1$ et ce sont celles qui peuvent apparaitre dans $\varphi'|_{\{\mathsf{w}_1,\ldots,\mathsf{w}_{o(i_0)}\}}$ où $o(i_0)$ est le nombre d'émissions apparaissant avant l'action $\mathbf{guess}(g_{i_0})$ dans la trace étudiée. Appliquant π^-_{1,i_0} des deux cotés de l'égalité, nous déduisons que

$$\big((S_{i_0}\sigma_2\{g_{i_0} \mapsto g'_{i_0}\})\varphi'\big)\pi^-_{1,i_0} = \big((S'_{i_0}\sigma_2\{g_{i_0} \mapsto g'_{i_0}\})\varphi'\big)\pi^-_{1,i_0}.$$

Nous avons que :

$$\begin{array}{rl} & ((S_{i_0}\sigma_2\{g_{i_0} \mapsto g'_{i_0}\})\varphi')\pi^-_{1,i_0} \\ = & ((S_{i_0}(\sigma^-_{2,i_0} \uplus \sigma^+_{2,i_0})\{g_{i_0} \mapsto g'_{i_0}\})\pi^-_{1,i_0}(\varphi'\pi^-_{1,i_0}) \\ = & [(S_{i_0}\tau^-_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi]\sigma^+_{2,i_0} \end{array}$$

où $\mathrm{dom}(\tau^-_{2,i_0}) = \mathrm{dom}(\sigma^-_{2,i_0}) = \{X \mid \mathsf{k}^-_w(X, x)$ est une hypothèse de f avec $|w| < p_{i_0}\}$, $\sigma_2 = \sigma^-_{2,i_0} \uplus \sigma^+_{2,i_0}$, et $\tau_{2,i_0} = \tau^-_{2,i_0} \uplus \tau^+_{2,i_0}$, Nous avons un résultat similaire pour S'_{i_0}. C'est pourquoi nous avons que :

$$[(S_{i_0}\tau^-_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi]\sigma^+_{2,i_0} = [(S'_{i_0}\tau^-_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi]\sigma^+_{2,i_0}$$

Nous considérons maintenant une substitution π qui remplace les constantes qui n'apparaissent pas dans $\ell_1\sigma_1, \ldots, \ell_{p_{i_0}}\sigma_1$ mais qui apparaissent dans $S_{i_0}\sigma_2$ ou alors dans $S'_{i_0}\sigma_2$ par des messages tel que $\pi(\sigma^+_{2,i_0}(X)) = (X\tau^+_{2,i_0})\{g_{i_0} \mapsto g'_{i_0}\}\varphi$. Notons que π est bien-défini puisque dans le cas où $\sigma^+_{2,i_0}(X) = \sigma^+_{2,i_0}(Y)$ et les variables X et Y apparaissent dans S_{i_0} ou alors dans S'_{i_0}, nous avons que $X\tau_{2,i_0} = Y\tau_{2,i_0}$.

Appliquant π aux deux cotés de l'égalité, nous déduisons que :

$$\begin{array}{c} \big((S_{i_0}\tau^-_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi\big)\sigma^+_{2,i_0}\pi \\ = \\ \big((S'_{i_0}\tau^-_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi\big)\sigma^+_{2,i_0}\pi \end{array}$$

C'est pourquoi, nous avons que :

$$\begin{gathered}(S_{i_0}(\tau^-_{2,i_0} \uplus \tau^+_{2,i_0})\{g_{i_0} \mapsto g'_{i_0}\})\varphi \\ = \\ (S'_{i_0}(\tau^-_{2,i_0} \uplus \tau^+_{2,i_0})\{g_{i_0} \mapsto g'_{i_0}\})\varphi\end{gathered}$$

Cela nous permet de conclure que :

$$(S_{i_0}\tau_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi = (S'_{i_0}\tau_{2,i_0}\{g_{i_0} \mapsto g'_{i_0}\})\varphi$$

D'où, nous avons que $R_{i_0}\{g_{i_0} \mapsto g'_{i_0}\}\varphi = R'_{i_0}\{g_{i_0} \mapsto g'_{i_0}\}\varphi$ menant à une contradiction. □

Théorème 3 *Soit T une trace close, $\mathcal{N}^T_{\mathsf{pub}} \subseteq \mathcal{N}_{\mathsf{pub}}$ (resp. $\mathcal{N}^T_{\mathsf{guess}} \subseteq \mathcal{N}_{\mathsf{guess}}$) l'ensemble fini de noms publics (resp. noms faibles) qui apparaissent dans T. Soit la base de connaissance saturée $K^0 \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T, \mathcal{N}^T_{\mathsf{pub}}, \mathcal{N}^T_{\mathsf{guess}})))$.*

Nous avons que T est atteignable si et seulement si $\mathsf{Reach}(T,\ K^0_{\mathsf{solved}})$ est vrai.

Preuve. Nous prouvons les deux directions séparément.

$\Leftarrow$ Si $\mathsf{Reach}(T,\ K^0_{\mathsf{solved}})$ est satisfait, alors nous avons en effet une exécution depuis $(T, \emptyset)$ vers $(\emptyset, \varphi)$, et donc, T est atteignable.

$\Rightarrow$ Par le Théorème 4, si T est atteignable alors $\mathsf{Reach}(T,\ K_{\mathsf{solved}})$ est satisfait pour tout $K \in \mathsf{sat}(K_{\mathsf{init}}(\mathsf{seed}(T)))$. Donc, il y a un statement identité atteignable f témoignant que T est atteignable. Par le Lemme 11 $K \subseteq K^0 \cup \mathcal{S}$ pour $K \in \mathsf{sat}(K_{\mathsf{init}}(T))$ et un ensemble $\mathcal{S}$ contenant des statements d'une forme spécifique. Comme $\mathcal{S}$ ne contient que des statements identités atteignables avec au moins un test trivial nous avons que $f \in K^0$ et d'où $\mathsf{Reach}(T,\ K^0_{\mathsf{solved}})$ est satisfait. □

Nous avons dans ce chapitre montré qu'une base de connaissance saturée construite à partir de la graine d'une trace T représente de manière correcte et complète les comportements des exécutions possibles de cette trace (Section 4.3). Nous avons également décrit une procédure (Algorithme 1) dont l'intégration dans l'outil AKISS sera détaillée dans le Chapitre 5.

Chapitre 5

Implémentation

Sommaire

5.1 Fonctionnement d'AKISS original 78
5.1.1 Modélisation 79
5.1.2 Vérification 81
5.2 Adaptation de l'outil AKISS à un attaquant qui devine les noms faibles 83
5.2.1 Modélisation avec des noms faibles 83
5.2.2 Vérification en présence de noms faibles 84

Nous avons implémenté la théorie décrite dans les chapitres précédents dans l'outil AKISS [AkG]. AKISS [CcCK12] est un outil écrit en OCaml qui permet de faire de la vérification formelle et automatique de propriétés de sécurité sur des protocoles cryptographiques, il est disponible en *open-source* à l'adresse [Aki]. Les protocoles sont représentés comme des processus avec une syntaxe proche de celle présentée dans le Chapitre 2 et l'outil vérifie entre deux processus des propriétés d'équivalence de traces. Pour vérifier une telle équivalence (comme défini dans [CcCK12]), il va générer les tests représentatifs atteignables depuis une trace du premier processus puis vérifier leur exécutabilité dans le deuxième processus et réciproquement. Nous utilisons la structure de cet outil afin de vérifier des propriétés d'atteignabilité, modifiant les modules qui génèrent les tests pour générer des égalités entre recettes comme décrites dans la Définition 4 du Chapitre 1, de ces tests nous ne gardons qu'un sous-ensemble représentatif. Ce sont ces tests qui nous permettent de vérifier si l'attaquant peut deviner les noms faibles nécessaires à une éventuelle attaque.

La Figure 5.1 représente les différents modules de l'outil AKISS. La première phase, la modélisation, est réalisée par l'utilisateur. Il faut fournir à AKISS :

— les détails des primitives cryptographiques utilisées dans le protocole sous forme d'un système de réécriture (Section 4.1) ;
— le protocole sous la forme d'un processus (Section 2.1) ; et
— la propriété de sécurité à la fois encodée dans le protocole et comme une question à l'outil (Section 2.4).

Ici nous nous intéressons à des propriétés d'authentification encodées comme des propriétés de correspondance (Section 2.4). On demandera donc à AKISS si un protocole

satisfait une certaine propriété de correspondance. Puis vient la phase de vérification où AKISS va successivement :

1. transformer la modélisation (déclarations, protocole, propriété de sécurité) en traces, cette transformation est décrite dans le Chapitre 3 ;
2. générer pour chaque trace une graine, voir Section 4.2 ;
3. saturer les clauses de Horn résultantes, la procédure de saturation est détaillée dans la Section 4.3 ; puis
4. appliquer la procédure de décision (Section 4.4) associée au résultat à vérifier ;
5. AKISS retourne soit une réponse positive, c'est-à-dire une preuve de sécurité, soit une trace d'attaque.

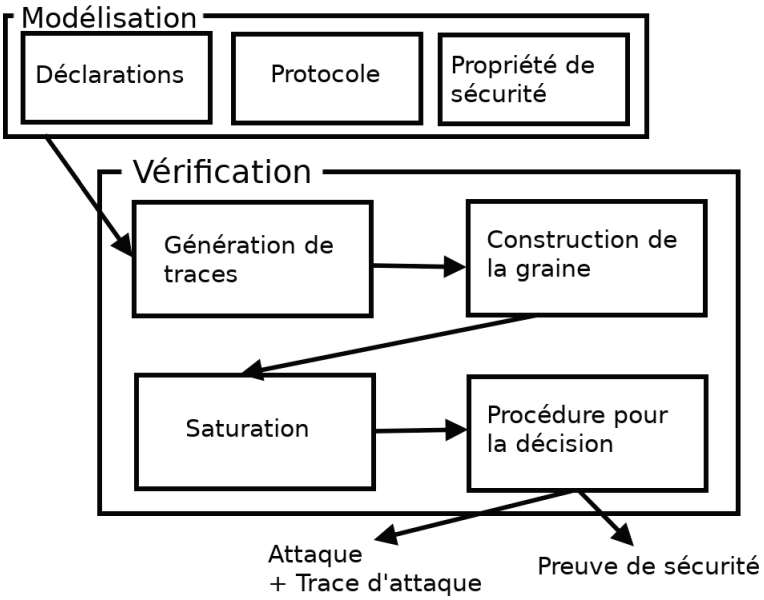


FIGURE 5.1 – Schéma représentant l'outil AKISS

Nous allons dans les sections ci-dessous introduire les différents modules de l'outil AKISS original afin de mettre en exergue nos modifications pour adapter l'outil à notre nouveau modèle d'attaquant. Nous exhibons aussi quelques optimisations qui prennent quelques libertés avec la théorie sans impacter la correction. De cette manière, nous réduisons le nombre de traces que l'on va considérer dans l'analyse, sélectionnant celles qui sont représentatives et nous considérons également les positions les plus pertinentes pour placer les actions **guess** dans la trace. La nouvelle version de l'outil AKISS est disponible sur [AkG]. Nous détaillons ci-dessous la structure d'AKISS dont l'intuition est donnée dans la Figure 5.1. Tous les exemples associés s'attachent à la modélisation du protocole présenté en Figure 4.

5.1 Fonctionnement d'AKISS original

Nous présentons dans cette section l'outil AKISS avant toute modification de notre part. Toutes les phases et mécanismes d'AKISS ont été par la suite réutilisés et plus ou

moins modifiés afin de satisfaire notre nouveau modèle d'attaquant et les propriétés d'atteignabilité.

5.1.1 Modélisation

La phase de modélisation est réalisée par l'utilisateur. Il doit spécifier à l'outil AKISS les primitives cryptographiques qu'il choisi de considérer ainsi que le protocole et la propriété de sécurité qu'il veut vérifier.

Déclarations : AKISS nous permet de définir des symboles de fonctions, comprenant les constantes utilisées dans notre protocole (symboles d'arité 0), ou les noms publics qui seront considérés au même titre, et les fonctions correspondant aux primitives cryptographiques (voir Chapitre 1). Les comportements de ces fonctions sont décrits en déclarant des règles de réécriture comme le système $\mathcal{R}_{\mathsf{hash}}$, détaillé dans l'Exemple 5, modélisant le comportement de fonctions de projection. Nous pouvons également définir des noms privés, qui ne pourront être appris par l'attaquant qu'à travers les échanges publics entre les participants du protocole et les canaux publics ou privés sur lesquels ces échanges ont lieu. Il nous faudra également déclarer les variables `X` et `Y` qui vont être utilisées dans les règles de réécriture.

```
symbols h/1, pair/2, proj1/1, proj2/1, infoA/0, ack/0;
private rA;
channels C;
privchannels EA, EB;
var X, Y, XA1, XB1, XB2;

rewrite proj1(pair(X,Y)) -> X;
rewrite proj2(pair(X,Y)) -> Y;
```

Le système de réécriture que nous modélisons est celui décrit dans l'Exemple 5. Dans cet exemple, bien que `rA` est censé être faible, il est défini comme un nom privé, car l'outil AKISS original ne peut modéliser cette subtilité. Nous définissons également un canal public `C` où A et B pourront échanger des messages sur lesquels l'attaquant pourra exercer son pouvoir comme dans le Chapitre 1. A et B ont à leur disposition un canal hors-bande unidirectionnel. Pour représenter ce canal il nous faut au moins un canal privé (A peut envoyer des messages à B sur `EA` et B peut envoyer des messages à A sur `EB`). Les variables `XA1`, `XB1` et `XB2` sont nécessaires pour pouvoir réaliser autant d'actions **in**, comme détaillé dans la modélisation du protocole ci-dessous. Les règles de réécriture définies ici représentent les propriétés des fonctions de projection qui permettent d'accéder aux éléments d'une paire, ce sont les seules règles nécessaires pour représenter les propriétés des fonctions utilisées dans ce protocole.

Protocole : Pour représenter un protocole nous devons l'exprimer comme un processus, de manière similaire à la description du Chapitre 2. Ainsi, les protocole de la Figure 4 sera modélisé par le code ci-dessous.

```
P = A || B;

A = out(C, pair(infoA, h(pair(infoA, rA)))).in(EB,XA1).out(EA,rA);

B = in(C,XB1).(out(EB,ack) || in(EA, XB2).
        if h(pair(proj1(XB1),XB2)) = proj2(XB1) then 0;
```

Ici le protocole de la Figure 4 est modélisé comme un processus `P` qui exécute de manière concurrente les processus `A` et `B` qui représentent les participants du protocole. Les canaux hors-bande sont modélisés comme étant asynchrones, authentiques et confidentiels, une modélisation décrite dans la Section 2.3. Nous avions modélisé ce protocole dans l'Exemple 8.

Propriété de sécurité : AKISS ne vérifiait que des propriétés d'équivalence. Ainsi, nous pouvions vérifier si `rA` pouvait être deviné à l'aide d'une propriété d'équivalence [Bau05]. Pour cela l'idée est de vérifier l'équivalence entre deux processus identiques, en leur ajoutant une action, à l'un l'émission du nom faible et à l'autre l'émission d'un nom frais. Le fait que l'attaquant peut distinguer ces deux processus revient à dire qu'il est capable de trouver un test exhibant le fait que le nom faible n'est pas frais. S'il est capable d'exhiber un tel test, alors il est également capable de deviner ce nom faible. Nous modifions donc la modélisation de notre processus `P` en ajoutant `rA` aux noms privés :

```
private rA, r;
```

Nous définissons de nouveaux protocoles afin de vérifier si `rA` est devinable. Pour cela nous définissons l'opérateur de *phase*, `>>`. Cet opérateur s'applique à deux processus, `P >> Q`, et signifie que toute exécution de `P` peut être interrompue pour continuer sur l'exécution de `Q`. Nous modélisons nos processus ainsi :

```
revealRA = out(C,rA);
revealR = out(C,r);
P1 = P >> revealRA;
P2 = P >> revealR;
```

Et enfin on encode la propriété vérifiant si les traces de `P1` sont incluses dans celles de `P2` ou si l'on peut les distinguer. Cette inclusion est suffisante du fait de la construction des processus `P1` et `P2` : en effet, par construction, toute égalité dans `P2` est également vérifiée dans `P1`. L'inclusion est donc suffisante, ici, savoir si `rA` est devinable revient à vérifier s'il existe un test dans `P1` qui n'est pas possible dans `P2`.

```
includedft? P1 in P2;
```

Cette façon de modéliser si un nom faible est devinable correspond bien à notre Définition 4 mais elle ne nous permet pas de vérifier des propriétés de sécurité plus évoluées sur le protocole. En fait dans ce protocole `rA` est toujours devinable, mais cela n'implique pas nécessairement une faiblesse du protocole. Toute la sécurité repose sur le fait que l'on veut que l'attaquant ne puisse deviner `rA` que trop tard pour impacter l'authenticité de l'information envoyée par `A`.

5.1.2 Vérification

Depuis la modélisation présentée dans la section précédente, l'outil AKISS va générer des clauses de Horn afin d'appliquer une procédure de décision.

Génération de traces : AKISS commence par générer des traces depuis la modélisation de l'utilisateur (déclarations, protocole, propriété de sécurité). Pour générer ces traces l'outil AKISS va transformer les processus en ensembles de traces comme présenté dans le Chapitre 3 (excluant les spécificités du modèle considérant les noms faibles, les actions **guess** et les évènements). Ainsi les processus concurrents qui sont composés d'une séquence d'actions vont se transformer en un ensemble de traces représentant les entrelacements possibles. Concernant les canaux privés, on résout syntaxiquement les émissions/réceptions, on considère tous les appariements possibles d'envoi et de réception sur un même canal privé, (traitant également le cas ou aucun appariement n'est effectué), et on effectue une substitution de la variable de l'action de réception par le terme émis. Cette procédure a été expliquée plus en détail dans la Section 3.3 traitant de la traduction d'un processus vers un ensemble de traces et elle s'approche du comportement de la fonction trad. Les branches `else` des tests ont été introduites récemment dans AKISS [GK17]. Seuls les tests positifs, i.e. les branches `then` et non `else`, sont considérés dans l'outil AKISS d'origine.

Dans notre exemple précédent, deux entrelacements sont possibles pour le processus `P` (excluant les cas où les émissions/réception sur canaux privés interrompent l'exécution du protocole) :

```
1) out(C, h(pair(infoA, rA))).in(C,XB1).
       [h(pair(proj1(XB1),rA)) = proj2(XB1)]

2) in(C,XB1).out(C, h(pair(infoA, rA))).
       [h(pair(proj1(XB1),rA)) = proj2(XB1)]
```

Cet exemple n'exhibe pas une optimisation de l'outil AKISS d'origine afin de générer moins de traces. En effet, les tests étant des actions invisibles pour l'attaquant et AKISS vérifiant des propriétés d'équivalence, les actions de tests sont toujours groupées avec l'action qui les suivent. Nous exhibons ce comportement avec la modélisation suivante :

```
P = in(C,X).if X = t then out(C,ok);
Q = out(C,t);
```

Ainsi, le processus `P || Q` sera traduit par l'ensemble de traces suivant :

```
1) out(C,t).in(C,X).[X=t].out(C,ok)

2) in(C,X).out(C,t).[X=t].out(C,ok)

2) in(C,X).[X=t].out(C,ok).out(C,t)
```

Génération de la graine : Cet ensemble de traces permet de générer la graine, c'est-à-dire les clauses de base que va considérer AKISS avant la saturation. La génération de la graine est similaire à la méthode décrite dans la Section 4.2 sans les spécificités liées à la connaissance faible de l'attaquant. AKISS va générer les clauses représentant le système de réécriture et qui plus tard vont être utilisées lors de l'application de la

procédure de saturation. Par exemple, la clause suivante signifie que l'attaquant sait construire la paire $< x, y >$ si il connait les termes `x` et `y` :

```
knows(pair(X,Y), pair(x,y)) <= knows(X,x), knows(Y,y)
```

Les points du protocole atteignables sont représentés par les clauses `reach` qui définissent quels mondes peuvent être atteints et sous quelles conditions. Depuis la première trace de l'exemple précédent nous aurions, parmi d'autres, la clause `reach` suivante :

```
reach_out(C).in(C,x) <= knows(X,x)
```

Les actions d'émissions augmentent potentiellement la connaissance de l'attaquant, ainsi des clauses `knows` vont être générées, modélisant le fait que l'attaquant connaitra le terme émis s'il atteint ce point de la trace. Ces clauses seront détaillées dans la prochaine section. Afin de générer moins de clauses, les mondes dans lesquels les clauses s'appliquent (par exemple `out(C).in(C,x)` pour la clause `reach` ci-dessus) se terminent par une variable dans le cas où la clause s'applique pour tous les mondes avec un préfixe spécifique. Par exemple, dans notre cas nous avons que l'attaquant connait `pair(infoA,h(pair(infoA, rA)))` à partir du moment où il a écouté le premier message. Nous aurons la clause suivante :

```
knows_out(C).in(C,x).y(w1,pair(infoA,h(pair(infoA, rA))))<=knows(X,x)
```

où `y` peut être instancié par n'importe quel suffixe.

Saturation : La procédure de saturation, décrite dans la Section 4.3, permet à AKISS de générer toutes les clauses qui vont être utile ensuite pour appliquer la procédure de décision. La saturation va résoudre les clauses de connaissance de l'attaquant afin d'exhiber la connaissance effective de ce dernier. AKISS va également générer des clauses `identical` considérant toutes les représentations possibles des tests réalisables pendant l'exécution de cette trace. Cette étape va aussi permettre de calculer les tests qui sont effectivement atteignables et sous quelles conditions. Ce calcul est parallélisé pour chaque trace dans l'outil AKISS afin de considérer plusieurs traces de manière concurrente.

Procédure de décision : Pour décider si l'équivalence est satisfaite, l'algorithme de décision va vérifier que pour tous les points atteignables de la trace que l'on est en train d'analyser sont également atteignables pour toutes les traces du deuxième processus avec lequel on veut vérifier l'équivalence. Pour cela, la procédure de décision, va utiliser l'ensemble de clauses saturées. On vérifie également si les tests faisables dans la trace analysée sont eux aussi jouables dans l'ensemble de traces du second processus. On répète cette procédure pour toutes les traces du premier processus, puis on fait de même avec les traces du second processus. L'outil répond ensuite soit positivement si l'équivalence entre les deux processus est effective soit négativement exhibant un élément discriminant de l'équivalence.

5.2 Adaptation de l'outil AKISS à un attaquant qui devine les noms faibles

L'outil AKISS est tout à fait adapté pour définir facilement notre nouveau modèle d'attaquant. En effet les tests générés pendant la phase de saturation peuvent être adaptés afin d'exhiber une possibilité pour l'attaquant de deviner un nom faible. Nous modifions l'outil AKISS afin de considérer notre nouveau modèle d'attaquant, ces modifications impactent toutes les étapes de la chaine de vérification de l'outil. Nous avons présenté dans la section précédente les différentes étapes de vérification de l'outil AKISS original, nous détaillons dans cette section les modifications et optimisations effectuées afin d'améliorer l'efficacité de l'outil.

5.2.1 Modélisation avec des noms faibles

Déclarations : Nous pouvons, comme détaillé dans la section précédente, définir des symboles de fonctions, des règles de réécriture spécifiques, des canaux publics ou privés et des noms privés. Mais il nous faut aussi dorénavant représenter les noms faibles. Pour déclarer un nom faible `rA`, on écrira :

```
weak rA;
```

Le nom `rA` n'est plus modélisé comme un nom privé, il est maintenant considéré comme un nom faible. Tous les noms faibles sont bien évidemment privés et non connus de l'attaquant, sinon les deviner ne serait plus un enjeu pour la sécurité du protocole.

Protocole : Le protocole est aussi modélisé comme un processus exactement comme décrit dans la Section 2.1. Ici la modélisation du protocole est très similaire à celle de l'outil AKISS d'origine présentée dans la section précédente. Nous ajoutons cependant des évènements afin d'annoter le protocole et le nom `rA` est maintenant défini comme un nom faible.

Propriété de sécurité : Avec nos modifications nous ne vérifions pas des propriétés d'équivalence, cela donnera certainement lieu à d'autres travaux sur l'outil AKISS. Nous vérifions ici des propriétés d'atteignabilité (Section 3.4), et plus particulièrement nous vérifions des relations de correspondance. Ainsi nous avons envie de modéliser ces propriétés de correspondance comme présentées dans l'Exemple 11. Nous les modélisons donc de la manière suivante :

```
P = A || B;

A = out(c,pair(infoA, h(infoA, rA))).in(EB,XA1).begin(infoA).
        out(EA,rA);

B = in(c,XB1).out(EB,ack).in(EA.XB2).
        if h(proj1(XB1),XB2) = proj2(XB1) then end(proj1(XB1));

correspondence? P;
```

Nous ajoutons au protocole les évènements `begin` et `end`, comme décrit dans la Section 2.4, afin de pouvoir par la suite exprimé une propriété de correspondance. On va

vérifier ici si $P \models [end(x) \Rightarrow begin(x)]$. Dans l'outil, nous avons réservé les mots clés `begin` et `end` pour représenter ces propriétés de correspondance afin de par la suite les traduire en propriétés d'atteignabilité. Ceci n'est pas utilisé dans l'exemple ci-dessus mais nous avons également ajouté la gestion des branches `else` des tests.

5.2.2 Vérification en présence de noms faibles

Génération de traces : Les traces sont générées comme expliqué dans la Section 3.3. Cependant plusieurs optimisations ont été mises en place, ceci pour diminuer le nombre de traces générées, considérant des traces représentatives. Une de nos optimisations vient du fait que nous vérifions des propriétés d'atteignabilité uniquement, celles ci plus simple que les propriétés de correspondance nous permettent, quand deux actions doivent s'entrelacer, de ne considérer que les traces qui privilégie l'action d'émission de messages [CDE05]. Ainsi, nous pouvons diminuer drastiquement le nombre de traces. Dans l'exemple de l'outil AKISS d'origine nous avions deux traces représentatives maintenant une trace unique a besoin d'être considérée :

```
1) out(C, pair(infoA,h(pair(infoA, rA)))).in(C,XB1).
        [h(pair(proj1(XB1),rA)) = proj2(XB1)].[infoA != proj1(XB1)]
```

Le test négatif à la fin de la trace représente la propriété de correspondance que nous voulons vérifier, encodée comme une propriété d'atteignabilité. Sur des exemples plus conséquents la diminution du nombre de traces est plus flagrante.

Mais nous devons également placer des actions **guess** à tous les endroits de la trace afin de représenter la capacité de l'attaquant à deviner un secret faible. Ici aussi nous pouvons sélectionner des traces représentatives où les actions **guess** seront placées à des endroits plus pertinents. Sans optimisation, la trace précédente nous donne les 6 traces suivantes :

```
1) out(C, pair(infoA, h(pair(infoA, rA)))).in(C,XB1).
        [h(pair(proj1(XB1),rA)) = proj2(XB1)].[infoA != proj1(XB1)]

2) out(C, pair(infoA, h(pair(infoA, rA)))).guess(rA).in(C,XB1).
        [h(pair(proj1(XB1),rA)) = proj2(XB1)].[infoA != proj1(XB1)]

3) out(C, pair(infoA, h(pair(infoA, rA)))).in(C,XB1).guess(rA).
        [h(pair(proj1(XB1),rA)) = proj2(XB1)].[infoA != proj1(XB1)]

4) out(C, pair(infoA, h(pair(infoA, rA)))).in(C,XB1)..guess(rA).
        [h(pair(proj1(XB1),rA)) = proj2(XB1)].[infoA != proj1(XB1)]

5) out(C, pair(infoA, h(pair(infoA, rA)))).in(C,XB1).
        [h(pair(proj1(XB1),rA)) = proj2(XB1)].guess(rA).
        [infoA != proj1(XB1)]

6) out(C, pair(infoA,h (pair(infoA, rA)))).in(C,XB1).
        [h(pair(proj1(XB1),rA)) = proj2(XB1)].
        [infoA != proj1(XB1)].guess(rA)
```

Notre optimisation est basée sur 2 observations :

— L'attaquant ne peut deviner un nouveau nom faible que quand sa connaissance (frame) augmente. Réessayer de deviner un nom faible à partir de la même frame n'est en effet pas utile.

— L'attaquant n'a besoin de deviner un nouveau nom faible qu'au moment où il souhaite fournir à un des participants du protocole un nouveau terme.

Donc, il nous suffit de considérer les traces où les actions **guess** sont placées après une nouvelle action d'émission (**out**) et juste avant une action de réception (**in**). Le nombre de traces se réduit de manière très conséquente :

```
1) out(C, pair(infoA,h(pair(infoA, rA)))).in(C,XB1).
        [h(pair(proj1(XB1),rA)) = proj2(XB1)]

2) out(C, pair(infoA,h(pair(infoA, rA)))).guess(rA).
        in(C,XB1).[h(pair(proj1(XB1),rA)) = proj2(XB1)]
```

Ces optimisations nous permettent de limiter l'explosion du nombre de traces due à notre modèle, tout spécialement celle concernant les actions **guess**.

Génération de la graine : La génération de la graine (Section 4.2) est très similaire à celle effectuée par la version originale de l'outil. Cependant nous prenons en compte les clauses de connaissance faible. Ainsi nous générerons la clause suivante :

```
knows+_X(rA,rA) <=
```

L'attaquant connait faiblement `rA` il pourra construire des tests à l'aide de ce nom faible comme décrit dans le Chapitre 1.

Saturation : La procédure de saturation décrite dans la Section 4.3 utilise des règles de résolution assez différentes de la procédure originale car il lui faut séparer la connaissance forte, permettant de définir les points atteignables dans la trace, et la connaissance faible permettant de générer des tests afin de vérifier par la suite si les noms faibles utilisés sont effectivement devinables. Ces deux modes de connaissances doivent être utilisés pour vérifier nos propriétés d'atteignabilité décrites dans le Chapitre 3, et ceci est représenté dans la génération des clauses `ri` qui définissent les tests atteignables. Nos clauses `ri` à la différence de l'outil AKISS d'origine peuvent contenir plusieurs paires de recettes, une paire de recette par action **guess** présente dans la trace (leur génération est détaillée dans la Figure 4.3). La procédure de saturation est elle aussi traitée comme une tâche parallèle s'exécutant sur plusieurs processus afin de saturer plusieurs traces de notre ensemble en même temps.

Procédure de décision : Notre procédure ne vérifie pas d'équivalence de traces mais va vérifier si les traces sont atteignables avec notre nouveau modèle d'attaquant. Ainsi, notre procédure va maintenant vérifier si pour toutes les clauses `ri` les tests négatifs sont vérifiés et si les tentatives de l'attaquant pour deviner des noms faibles se vérifient (Algorithme 1). Dans la deuxième trace de notre exemple une clause `ri` typique que nous allons vérifier serait la suivante :

```
ri_out(C).guess(rA).guess(rA).in(C,pair(n',h(n',rA))).test=,test!=<=
```

Le test représenté par la clause est en effet un test qui permet de deviner `rA` et nous trouvons la trace d'exécution correspondant à une attaque. Elle est dans ce cas retournée à l'utilisateur. Si nous ne trouvons pas de trace d'attaque à la fin de notre procédure nous informons l'utilisateur que la propriété de sécurité qu'il voulait vérifier est satisfaite.

L'implémentation des comportements décrits ici est disponible en ligne, voir [AkG]. Dans le chapitre suivant nous étudierons les protocoles du standard ISO 9798-6 [ISO10] concernant les transfert de données à l'aide d'une interaction manuelle.

Chapitre 6

Études de cas

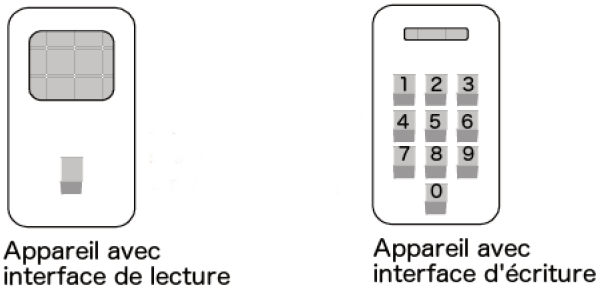
Sommaire

6.1 Protocoles utilisant des noms faibles 88
6.2 Protocoles utilisant des fonctions de hachage faibles . 92
6.3 Protocoles utilisant des noms faibles et des fonctions de hachage faibles 96
6.4 Autres protocoles 98

Nous nous intéressons ici à la vérification des protocoles du standard ISO/IEC 9798-6 :2010 sur les mécanismes utilisant des transferts de données manuels [ISO10]. Ce standard décrit 10 mécanismes permettant d'authentifier une donnée à l'aide d'une interaction manuelle. Ces protocoles s'attachent à la communication entre deux participants, A et B. Chaque participant va obtenir une donnée, d_A pour A et d_B pour B. Le but de chaque protocole est que les participants A et B parviennent à se convaincre qu'ils partagent la même donnée, si c'est effectivement le cas. Ces participants utilisent pour leurs échanges des appareils pouvant communiquer entre eux via un canal public, où l'attaquant peut donc écouter, intercepter et forger des messages, et ces appareils sont équipés d'écrans et de boutons afin de permettre une interaction avec l'utilisateur. Ces appareils peuvent soit disposer d'une interface de réception permettant à l'utilisateur de valider ou refuser des messages s'affichant à l'écran, soit d'une interface d'émission permettant à l'utilisateur d'entrer des messages de taille raisonnable et de lire des informations d'acquittement sur l'écran, voir Figure 6.1.

Dans ce chapitre nous analysons ces protocoles. Quand ces appareils communiquent grâce à l'aide active de l'utilisateur nous représentons cette communication par un échange sur un canal hors-bande. Nous considérons les canaux hors-bande utilisés pour représenter nos mécanismes, non confidentiels, qui est le cas le plus favorable à un attaquant (voir Section 2.3). Afin de vérifier les propriétés d'authentifications semblant être assurées par ces protocoles nous allons pouvoir utiliser l'outil AKISS modifié que nous avons détaillé dans le Chapitre 5. Nous allons décrire les parties importantes de la modélisation des protocoles ainsi que les résultats expérimentaux obtenus en portant particulièrement notre attention sur les protocoles représentatifs de méthodes caractéristiques. Cependant, tous les fichiers liés aux exemples présentés dans cette partie sont disponibles sur [AkG]. Les tableaux de résultats présentés dans ce chapitre sont renseignés à l'aide d'un calculateur de 20 cœurs physiques ayant une fréquence de

3GHz.



L'appareil de gauche dit *appareil avec interface de lecture* ne possède qu'un bouton permettant à l'utilisateur d'accepter le message affiché à l'écran. L'appareil de droite, dit *appareil avec interface d'écriture*, permet à l'utilisateur de lire des messages d'acquittement ainsi que d'écrire des messages courts utilisant le clavier numérique.

FIGURE 6.1 – Types d'appareils de communication

6.1 Protocoles utilisant des noms faibles

La propriété d'authentification de la donnée, sur laquelle les participants (appareils) se mettent ici d'accord, repose, pour les mécanismes 4 et 6, sur l'envoi d'un secret faible sur un canal hors-bande classique (voir Section 2.3). Dans notre cas c'est l'utilisateur humain qui va servir de médian et donc jouer le rôle du canal hors-bande. Nous détaillons dans cette section le mécanisme 4 (Figure 6.2). Ce protocole est une version très similaire au protocole que nous avons introduit dans l'introduction en Figure 5. L'appareil A est un appareil avec interface de lecture et B est un appareil avec interface d'écriture. Chaque appareil reçoit au début du protocole une donnée, d_A ou d_B et le protocole consiste en la vérification que ces données représentent bien la même valeur. Nous aurions pu représenter l'obtention des données de manière différente, par exemple A aurait pu générer une donnée d_A et l'envoyer via un canal public à B. L'appareil A, en vu d'authentifier la donnée reçue par B, va générer deux nonces, un nonce fort qui fera office de clé cryptographique : k, et un nonce faible : r. Ensuite A envoie sur un canal public, directement à B un premier message contenant le haché de sa donnée d_A, la clé k et le nom faible r : $\mathsf{hash}(d_A, k, r)$. Il va ensuite signifier à l'utilisateur humain qu'il est prêt à poursuivre le protocole, via un message sur son écran d'affichage (le message *ready* de la Figure 6.2). Une fois le premier message reçu, B signifie lui aussi à l'utilisateur qu'il est prêt à poursuivre le protocole. L'utilisateur, après avoir reçu confirmation que les deux appareils étaient prêt, enclenche la suite du protocole en envoyant un signal de démarrage à l'appareil A, (le message *start* dans la Figure 6.2), en appuyant sur un bouton de l'appareil par exemple. L'appareil A s'est maintenant engagé sur la valeur m, résultante de la fonction de hachage, et il sait, grâce à l'action de l'utilisateur, que B a bien reçu un engagement sur ce haché m. Il va pouvoir maintenant révéler à B le nom r via le canal hors-bande, c'est-à-dire qu'il va afficher la valeur de r sur son écran d'affichage et l'utilisateur va retranscrire cette valeur en la tapant sur l'appareil B. Puis A va envoyer sur un canal classique la clé k. L'appareil B peut maintenant vérifier que le haché m qu'il a reçu au début du protocole confirme

l'authenticité de sa donnée d_B en vérifiant s'il peut reconstruire ce haché avec d_B pour tester si $m = \mathsf{hash}(d_B, k, r)$. Si oui, alors il affiche *accept* si non il affiche *reject* et l'utilisateur transmet l'information à l'appareil A.

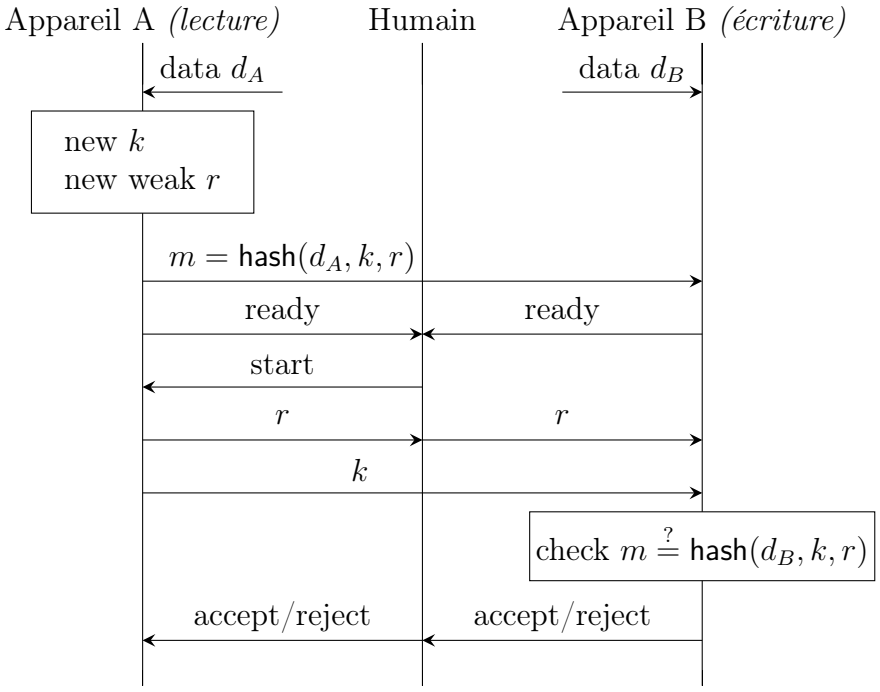


FIGURE 6.2 – Mécanisme d'authentification manuelle 4

Nous modélisons cette transcription du protocole dans AKISS. Pour vérifier les propriétés d'authentification qui s'expriment comme des propriétés de correspondance (Section 2.4) nous ajoutons les évènements `begin` et `end` afin de vérifier la propriété [`end(x)` ⇒`begin(x)`]. Nous considérons dans la Figure 6.4 la propriété de correspondance du point de vue de B, renvoyant aux exemples détaillés dans AKISS [AkG] pour le point de vue de A. La syntaxe de la modélisation est présentée dans le Chapitre 5. Nous voyons ici que r est défini de manière à marquer le fait que c'est un nonce faible. Sont définis 4 processus principaux, tout d'abord `A`, `B` et `H`, qui représentent respectivement les appareils A et B et l'utilisateur humain. Puis le processus P qui va représenter l'exécution parallèle de ces trois participants.

scénario		# traces	temps	statut
une session	Auth. A	8	< 1s	ok
	Auth. B	8	< 1s	ok
sans *start*	Auth. A	18	< 1s	attaque
	Auth. B	28	< 1s	attaque
k faible	Auth. A	22	< 1s	attaque
	Auth. B	22	< 1s	attaque
deux sessions	Auth. A	1416	< 5s	ok
	Auth. B	1416	< 5s	ok

FIGURE 6.3 – Résultats pour le mécanisme d'authentification 4

```
symbols h/3, ok/0, ready/0, start/0;

private k;
weak r;
channels CA, CB, CH;
privchannels OAH1, OAH2, OHA1, OHA2, OBH, OHB;
var X,Y, XA1, XA2, XA3, XB1, XB2, XB3, XB4, XH1, XH2, XH3, XH4;

A = in(CA,XA1).out(CA,h(XA1,k,r)).(out(OAH1,ready)
|| in(OHA1,XA2).if XA2=start then
(begin(XA1).out(OAH2,r) ||
out(CA,r).out(CA,k).in(OHA2,XA3).[XA3=ok]));

B = in(CB,XB1).in(CB,XB2).(out(OBH,ready)
|| in(OHB,XB3).in(CB,XB4).if XB2=h(XB1,XB4,XB3) then
out(OBH,ok).end(XB1));

H = (in(OAH1,XH1) || in(OBH,XH2)).if XH1=ready then
if XH2=ready then (out(OHA1,start) ||
in(OAH2,XH3).(out(OHB,XH3) ||
out(CH,XH3).in(OBH,XH4).if XH4=ok then out(OHA2,ok)));

P = A || B || H;

correspondence? P;
```

FIGURE 6.4 – Modélisation du mécanisme 4

Nous avons également vérifié si le protocole pouvait se simplifier en modifiant les propriétés de certains messages ou en retirant des synchronisations (les messages *ready* et *start* de la Figure 6.2). Le scénario *une session* correspond à la vérification pour une session du protocole, les lignes *Auth.A* et *Auth.B* indique chez quel participant nous positionnons l'évènement de fin de session. La sous-ligne *Auth.B* correspond donc à la vérification de l'exemple représenté en Figure 6.4. Pour vérifier cet exemple AKISS a généré 8 traces (colonne *# traces*) pour les deux sous-scénarios et enchainant sur la vérification cela lui a pris moins de 1 seconde (colonne *temps*). Cette vérification valide la propriété d'authentification comme étant effective, il n'y a pas d'attaques. Nous vérifions également le scénario sans synchronisation de l'utilisateur, c'est-à-dire

que l'appareil A n'attend pas un appui sur *start* de l'utilisateur avant de poursuivre le protocole. Nous pouvons voir ici que cela donne lieu à une attaque. L'attaque fonctionne ici de manière très similaire à celle de la Figure 4 présentée précédemment. En effet, sans synchronisation l'attaquant peut intercepter k avant que B ai reçu le haché, ainsi l'attaquant peut deviner r et construire un nouveau haché pour tromper B sur l'authenticité de la donnée. Nous considérons également le scénario où k est devenu un nom faible, dans ce cas nous considérons ces noms devinables comme conjointement faibles tel qu'expliqué dans la Section 1.4. Ainsi, l'attaquant peut construire un test $\mathsf{hash}(d_A, k, r) = m$ et deviner en même temps k et r. Comme AKISS travaille avec un nombre borné de sessions nous vérifions également le cas où nous exécutons deux sessions en parallèle.

Le mécanisme d'authentification 6, est très similaire au mécanisme 4 et utilise les mêmes particularités pour assurer l'authentification. Cependant dans ce cas les appareils considérés n'effectuent que des actions de lecture, comme décrit dans la Figure 6.1. L'utilisateur humain va donc devoir effectuer lui-même une comparaison. Dans le mécanisme 4, l'authenticité de r était assurée par une transmission sur un canal hors-bande, donc par la recopie de cette valeur par l'utilisateur de l'appareil A vers B. Dans le mécanisme 6, la transmission de r se fait via un canal classique, l'utilisateur ne pouvant écrire la valeur de r dans B ne disposant qu'une interface de lecture. Cependant, il va s'assurer ensuite que les appareils A et B étaient bien d'accord sur la valeur de r à utiliser. Nous pouvons noter ici qu'une erreur de l'utilisateur dans la comparaison finale peu se produire et aura davantage de chance d'entrainer de faux positifs. Ce mécanisme est modélisé dans la Figure 6.6 ci-dessous, et les résultats sont détaillés dans le tableau de la Figure 6.5 l'accompagnant. Nous testons les mêmes cas que pour le mécanisme 4. Nous remarquons que les petites différences entre ces protocoles entraine une augmentation du nombre de traces, ceci est dû à l'agencement différent des échanges et à l'émission supplémentaire. Ne parvenant pas à optimiser la génération de trace de la même manière, le nombre de traces augmente de manière conséquente et le temps de vérification pour deux sessions devient important.

scénario		# traces	temps	statut
une session	Auth. A	64	< 1s	ok
	Auth. B	52	< 1s	ok
sans *start*	Auth. A	136	< 1s	attaque
	Auth. B	367	< 2s	attaque
deux sessions	Auth. A	83806	13h	ok
	Auth. B	54677	45h	ok

FIGURE 6.5 – Résultats pour le mécanisme d'authentification 6

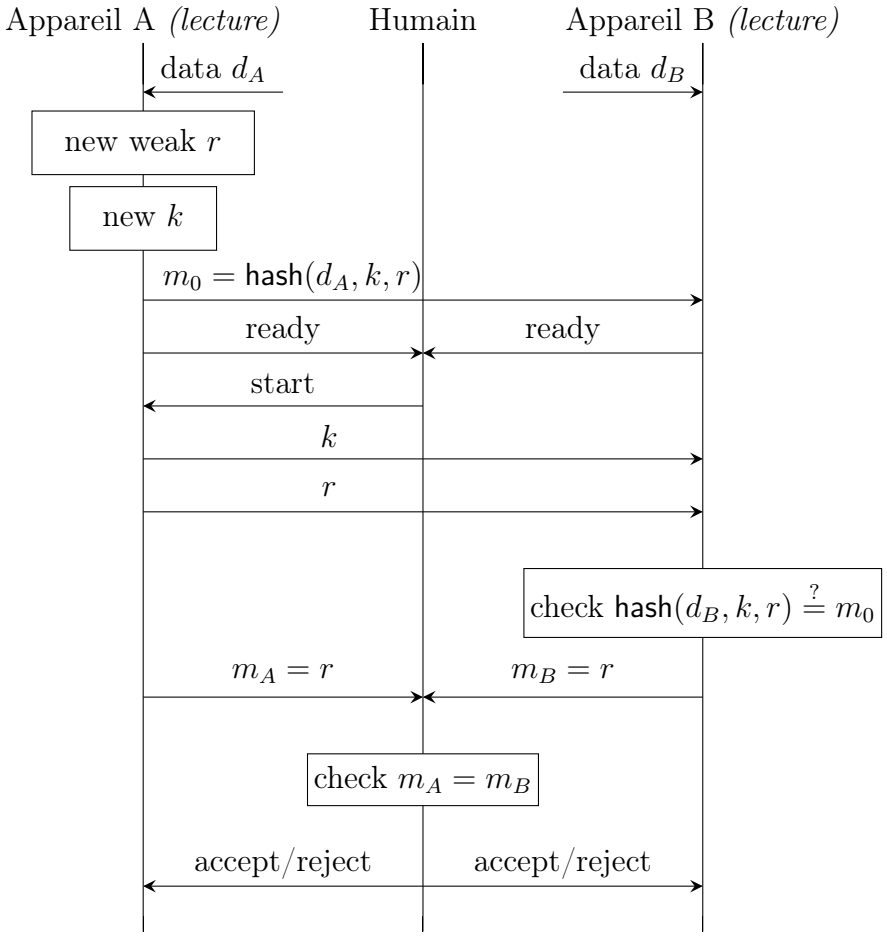


FIGURE 6.6 – Mécanisme d'authentification manuelle 6

L'outil AKISS modifié avec notre nouveau modèle d'attaquant nous a permis ici de considérer le nom faible r, de l'ajouter à la connaissance de l'attaquant quand nécessaire et ainsi de vérifier si les propriétés d'authentification de ces protocoles étaient soumises à une éventuelle attaque.

6.2 Protocoles utilisant des fonctions de hachage faibles

Les mécanismes d'authentification 3 et 5 ont en commun que la propriété d'authentification de la donnée agréée par les participants repose sur l'envoi d'une valeur résultante de l'application d'une fonction de hachage faible sur un canal hors-bande. Ce canal hors-bande est ici aussi considéré comme un canal hors-bande classique (Section 2.3), et va être représenté par l'interaction d'un utilisateur humain. Nous détaillons le mécanisme 3 qui concerne un appareil A qui possède une interface de lecture et l'appareil B qui possède une interface d'écriture. Ce protocole est décrit dans la Figure 6.9. Comme dans la section précédente la propriété d'authentification repose sur le fait que A et B doivent s'assurer que la donnée qu'ils reçoivent au début du protocole représente bien la même valeur. Pour authentifier la donnée de B, l'appareil A va générer un nonce

fort k, et envoyer un haché de k à B sur un canal public. Une fois envoyé A signifie à l'utilisateur qu'il est prêt à poursuivre le protocole. Une fois que B a reçu ce message il signifie lui aussi à l'utilisateur qu'il est prêt à poursuivre le protocole. Constatant que les deux appareils sont prêts à continuer. L'utilisateur signifie à l'appareil A qu'il peut poursuivre. Ainsi A va maintenant envoyer via le canal hors-bande, c'est-à-dire afficher à l'intention de l'utilisateur, le haché de sa donnée avec le nonce k, ce haché est le résultat de l'application de sh, une fonction de hachage faible. L'utilisateur va écrire cette valeur sur l'appareil B, pendant ce temps A aura envoyé à B le nonce k. L'appareil B va à la fois vérifier que le premier message m_1 qu'il a reçu correspond bien à ce nonce k et vérifier que le haché faible qu'il a reçu est égal au haché qu'il peut construire avec le nonce k reçu et sa donnée d_B. Si ces tests sont satisfaits, il signifie à l'utilisateur que la donnée est authentifiée et l'utilisateur reporte l'information à A.

Nous avons modélisé ce protocole dans AKISS, afin de vérifier cette propriété d'authentification. Pour cela nous l'avons exprimé comme une propriété de correspondance. Nous représentons ci-dessous (Figure 6.7) la modélisation de ce protocole pour AKISS (considérant l'authentification pour B). Remarquons que la faiblesse de la fonction de hachage est signifiée dans la théorie équationnelle (par des instructions `rewrite`) comme décrit dans la Section 1.2.

```
symbols ready/0, start/0, accept/0, sh/2, hash/1, bf/2;

private k;
channels CA,CB,CH;
privchannels OAH,OBH,OHA,OHB;
var X,Y,X1,Y1,XA1,XA2,XA3,XB1,XB2,XB3,XB4,XH1,XH2,XH3,XH4;

rewrite sh(bf(Y, sh(X1,Y1)),Y) -> sh(X1,Y1);
rewrite sh(X,bf(X, sh(X1,Y1))) -> sh(X1,Y1);

A = in(CA,XA1).out(CA,hash(k)).(out(OAH,ready)
       || in(OHA,XA2).if XA2=start then begin(XA1).
       (out(OAH,sh(XA1,k)) || out(CA,sh(XA1,k)).out(CA,k).
       in(OHA,XA3).if XA3=accept then 0 ));

B = in(CB,XB1).in(CB,XB2).(out(OBH,ready)
       || in(OHB,XB3).in(CB,XB4).if hash(XB4)=XB2 then
       if sh(XB1,XB4)=XB3 then (out(OBH,accept) || end(XB1) ));

H = (in(OAH,XH1) || in(OBH,XH2)).if XH1=ready then if XH2=ready then
       (out(OHA,start) || in(OAH,XH3).(out(OHB,XH3) || out(CH,XH3).
       in(OBH,XH4).if XH4=accept then out(OHA,accept) ));

P = (A || B || H);

correspondence? P;
```

FIGURE 6.7 – Modélisation du mécanisme 3

Nous étudions également des variations de ce protocole afin de vérifier la pertinence de l'enchainement de messages et de synchronisations. Le tableau en Figure 6.8 détaille ces vérifications. Nous vérifions tout d'abord que la propriété d'authentification est satisfaite à l'aide d'évènements pour vérifier une propriété de correspondance, pour A

et pour B. Nous vérifions également que l'action de synchronisation est nécessaire à la sécurité du protocole. Et nous vérifions aussi que c'est parce que la fonction de hachage est faible qu'une telle synchronisation est nécessaire. Le cas où nous considérons deux sessions en parallèle est également vérifié.

scénario		# traces	temps	statut
une session	Auth. A	3	< 1s	ok
	Auth. B	3	< 1s	ok
sans *start*	Auth. A	9	< 1s	attaque
	Auth. B	16	< 1s	attaque
sans *start* + h fort	Auth. A	9	< 1s	ok
	Auth. B	16	< 1s	ok
deux sessions	Auth. A	90	< 3s	ok
	Auth. B	72	< 3s	ok

FIGURE 6.8 – Résultats expérimentaux pour le mécanisme d'authentification 3

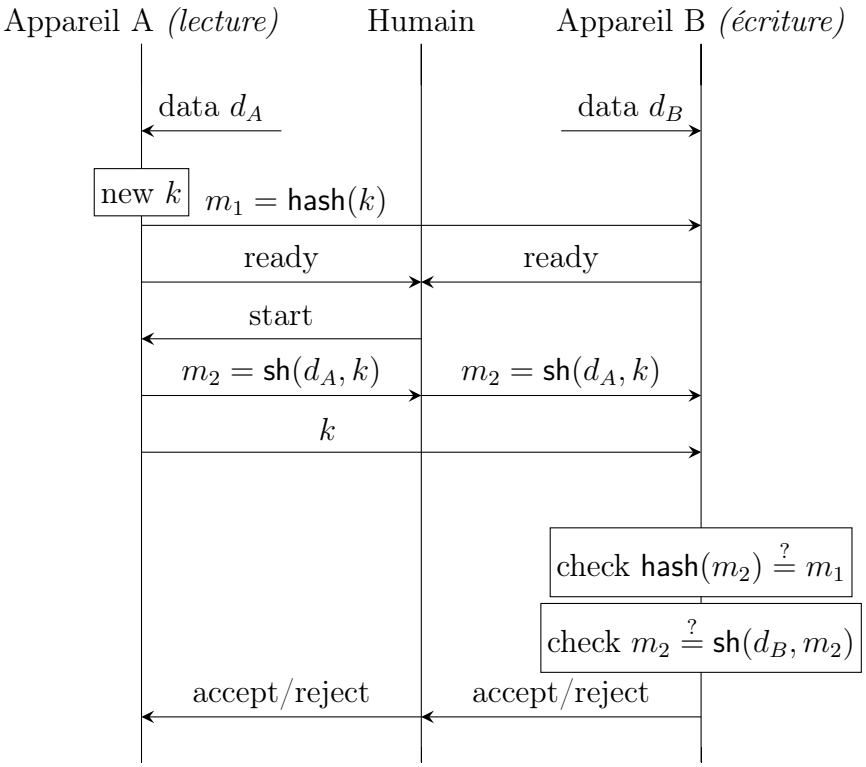


FIGURE 6.9 – Mécanisme d'authentification manuelle 3

Nous avons aussi vérifié ces propriétés d'authentification pour le mécanisme 5. Sa description est détaillée dans la Figure 6.11. Dans ce protocole, les deux appareils disposent d'une interface de lecture. Ainsi, l'utilisateur n'a plus les moyens de transmettre le haché m_2 de la Figure 6.9 à l'appareil B. Dans la Figure 6.11 on peut donc voir qu'il

va lire les messages m_A et m_B affichées respectivement par les appareils A et B, et les comparer lui-même. Il va donc assurer la fonction de vérifier que les hachés générés par A et par B sont bien identiques. Le fait que l'humain prenne une si grande part dans la validation de la donnée que vont utiliser les appareils peut être source de faux positifs. Les résultats obtenus sont décrits dans le tableau de la Figure 6.10.

scénario		# traces	temps	statut
une session	Auth. A	25	< 1s	ok
	Auth. B	18	< 1s	ok
sans *start*	Auth. A	55	< 1s	attaque
	Auth. B	106	< 1s	attaque
sans *start* + h fort	Auth. A	55	< 1s	ok
	Auth. B	106	< 1s	ok
deux sessions	Auth. A	3925	11h	ok
	Auth. B	1962	5h	ok

FIGURE 6.10 – Résultats expérimentaux pour le mécanisme d'authentification 5

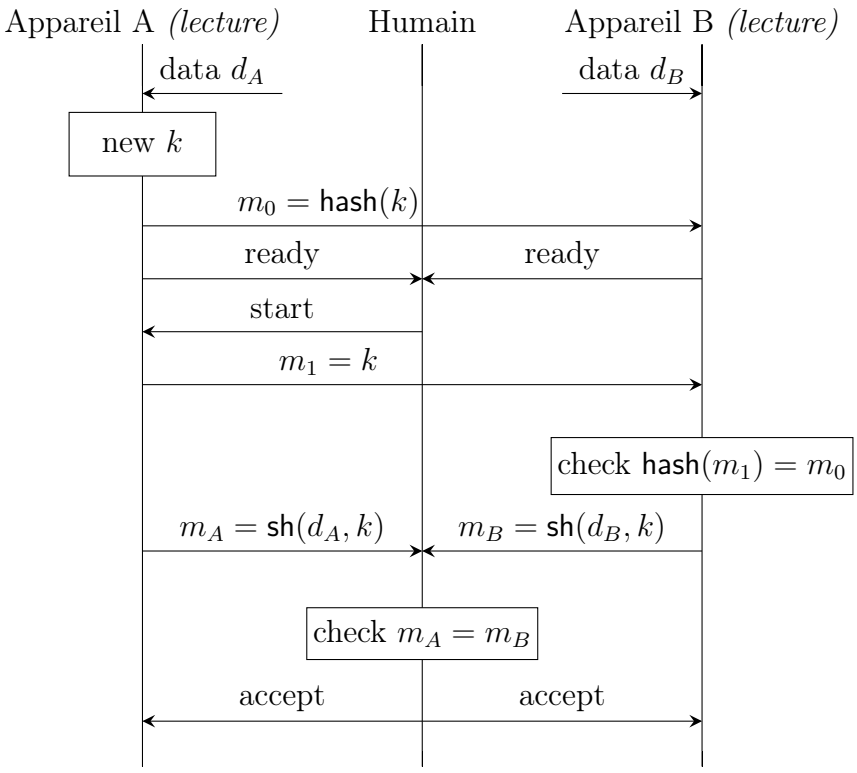


FIGURE 6.11 – Mécanisme d'authentification manuelle 5

Le fait que l'outil AKISS permet l'utilisation de théories équationnelles définies par l'utilisateur nous permet ici de modéliser le fait que les fonctions de hachage sont faibles

et ainsi de vérifier ce protocole tout en donnant à l'attaquant la capacité de trouver des collisions sur ces fonctions de hachage.

6.3 Protocoles utilisant des noms faibles et des fonctions de hachage faibles

Les mécanismes d'authentification 1 et 2 ont en commun que la propriété d'authentification de la donnée agréée par les participants repose à la fois sur une fonction de hachage faible, représentée dans la Figure 6.12 par sh, et d'un nom faible, représenté par k sur cette même figure. Nous nous intéresserons ici au mécanisme d'authentification 1 du standard [ISO10]. Dans ce protocole l'agent A est un appareil à interface de lecture et B est un appareil à interface d'écriture. Les deux appareils vont se mettre d'accord sur une donnée à utiliser. Dans la Figure 6.12, nous décrivons le protocole tel qu'il sera ensuite modélisé pour être vérifié par AKISS (voir exemples [AkG]). Les deux appareils vont considérer chacun une donnée, d_A pour A et d_B pour B.

Ce protocole, tout comme le protocole de la section précédente ne commence pas son exécution tant que les appareils n'ont pas reçu (ou généré) une donnée et signifié par un signal quelconque (par exemple un signal lumineux ou un texte à l'écran) à l'utilisateur humain qu'ils étaient prêts à démarrer la session. Quand cet utilisateur s'est assuré que les deux appareils sont prêts à démarrer une session du protocole il va démarrer la session en signifiant à l'appareil A de continuer l'exécution du protocole. Puis l'utilisateur retranscrit à B le message de A contenant un nonce faible r et le résultat de l'application d'une fonction de hachage faible : sh, dont le but est d'authentifier la donnée transmise à l'aide de la clé. Le canal sur lequel sont transmis la clé et le résultat de sh est un canal hors-bande avec les hypothèses classiques (Section 2.3). Le rôle de ce canal hors-bande est ici directement assuré par un utilisateur humain. Il devra écrire k et le résultat de notre fonction de hachage sh, qui va renvoyer un entier de petite taille, sur l'appareil B. Ici, la propriété d'authentification se veut assurée par l'envoi sur le canal hors-bande à la fois d'un nom faible k et d'un haché faible $\mathsf{sh}(r, d_A)$.

Comme dans les sections précédentes nous avons considéré plusieurs scénarios. Dans le dernier scénario nous vérifions que la propriété prouvée dans le cas *une session* est toujours effective dans le cas où l'on exécute *deux sessions* du protocole. On peut voir dans ce dernier cas un nombre de traces bien plus important même si le temps d'analyse reste raisonnable. Ce nombre de traces s'explique à la fois parce qu'il faut maintenant considérer l'entrelacement des actions des processus des deux sessions du protocole, mais aussi parce qu'en considérant deux sessions nous considérons également deux noms faibles. Comme nous le détaillons dans l'Exemple 17 et dans la Sous-Section 5.2.2, le positionnement des actions **guess** dans une trace a un impact conséquent sur l'ensemble de traces généré.

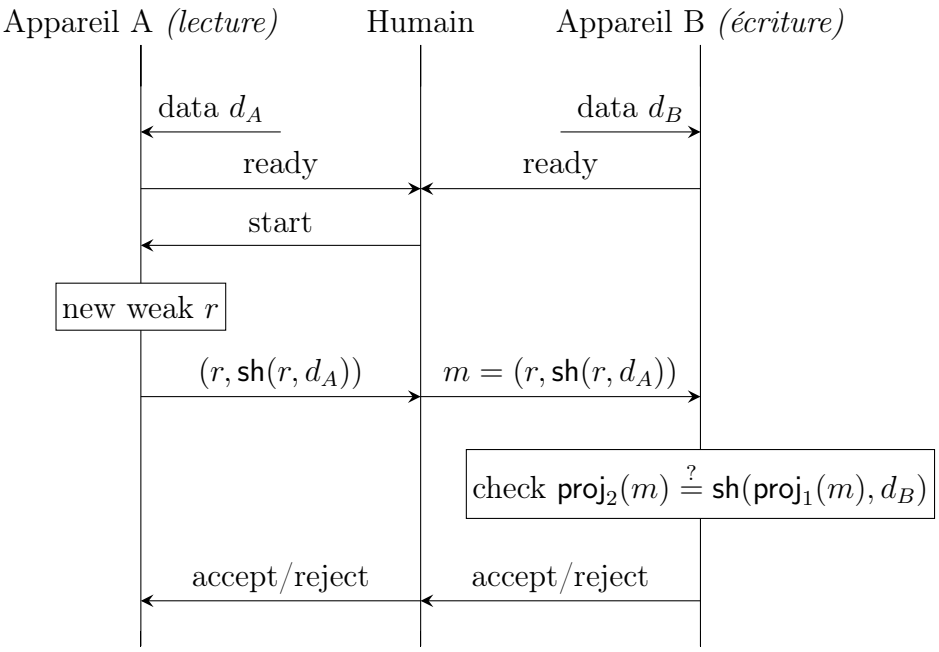


FIGURE 6.12 – Mécanisme d'authentification manuelle 1

scénario		# traces	time	status
une session	Auth. A	5	$<$ 1s	ok
	Auth. B	2	$<$ 1s	ok
sans *start*	Auth. A	8	$<$ 1s	attaque
	Auth. B	4	$<$ 1s	attaque
sans *start* + h fort	Auth. A	8	$<$ 1s	ok
	Auth. B	4	$<$ 1s	ok
deux sessions	Auth. A	410	$<$ 1m	ok
	Auth. B	46	$<$ 8s	ok

FIGURE 6.13 – Résultats pour le mécanisme d'authentification 1

Nous avons également vérifié le mécanisme 2 qui est détaillé dans la Figure 6.14 accompagné des résultats de la Figure 6.15. Dans ce mécanisme, nous considérons maintenant deux appareils de lecture. La différence de conception de ce protocole est donc la même différence que nous avons expliqué dans les sections précédentes. C'est-à-dire que l'utilisateur ne pouvant entrer de valeurs dans l'appareil B va devoir effectuer lui-même une comparaison. Ici, il va comparer à la fois le nom faible r et les hachés de ce nom faible avec la donnée considérée par les appareils. Dans le tableau de résultats de la Figure 6.15, nous observons les mêmes comportements que dans le tableau précédent, Figure 6.13.

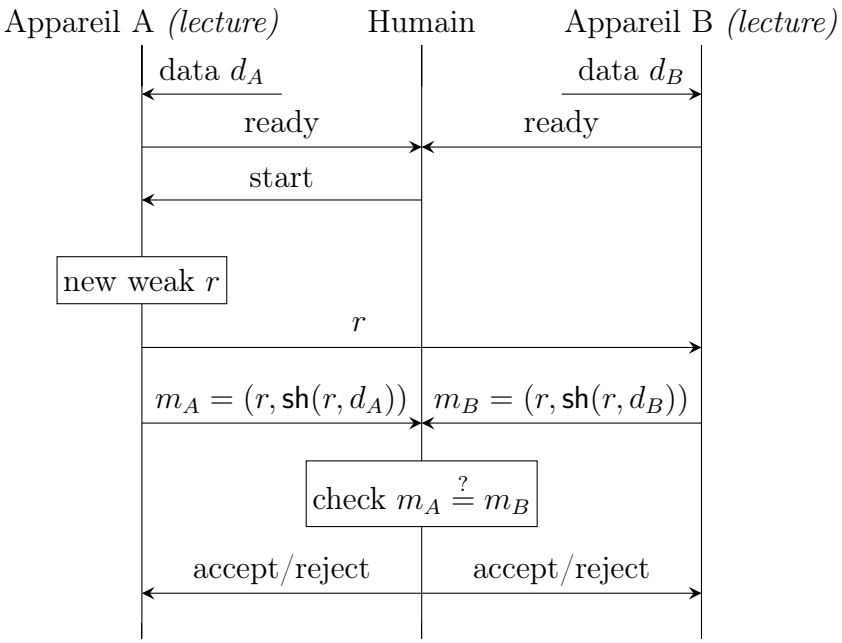


FIGURE 6.14 – Mécanisme d'authentification manuelle 2

scénario		# traces	temps	statut
une session	Auth. A	14	< 1s	ok
	Auth. B	12	< 1s	ok
sans *start*	Auth. A	16	< 1s	attaque
	Auth. B	18	< 1s	attaque
sans *start* + h fort	Auth. A	16	< 1s	ok
	Auth. B	18	< 1s	ok
deux sessions	Auth. A	513	23m	ok
	Auth. B	322	3m	ok

FIGURE 6.15 – Résultats pour le mécanisme d'authentification 2

Les protocoles de cette section font la synthèse des sections 6.1 et 6.2 concernant les capacités de l'attaquant à considérer pour vérifier de tels protocoles. Ainsi, il est important de pouvoir considérer à la fois les fonctions de hachage faibles et les secrets faibles.

6.4 Autres protocoles

Ces protocoles sont spécifiés dans le standard comme utilisant des fonctions de hachage de type MAC (*Message authentication code*) conformément au standard des mécanismes utilisant des chiffrements par blocs ISO/IEC 9797-1 [ISO11]. Les fonctions de hachage représentant l'application de l'algorithme MAC seront représentés ici par

la fonction hash s'appliquant à l'identité de l'émetteur, la donnée à hacher, une clé spécifique à l'algorithme, et un nonce faible aléatoire. Dans ces mécanismes nous ne considérons plus nos canaux hors-bande comme publics. Nous devons les considérer confidentiels car, le secret faible étant envoyé dès le début de la session, si l'attaquant pouvait l'apprendre à ce moment, le protocole serait trivialement attaquable.

Appareils possédant uniquement une interface d'écriture : Le protocole 7a est conçu pour des appareils utilisant uniquement une interface d'écriture. L'utilisateur humain ne peut donc pas ici comparer ou lire d'informations complexes qui auraient pu être affichées dans le cas où un des appareils possédait une interface de lecture. Le protocole 7a est modélisé en Figure 6.16. La sécurité de ce protocole repose notamment sur l'utilisation d'un nom faible r. Comme pour les protocoles précédents les appareils A et B vont devoir se mettre d'accord sur une donnée. Dans notre choix de modélisation chaque appareil reçoit sa donnée depuis l'environnement et va devoir vérifier si ces données correspondent à la même valeur. L'utilisateur va générer un nonce de petite taille, le nom faible r, et le transmettre aux appareils A et B qui sont donc pourvus d'une interface d'écriture. Une fois la valeur entrée, ils signifient qu'ils sont prêts à continuer le protocole et l'utilisateur va signifier à A qu'il peut poursuivre. L'appareil A va, via un canal public, envoyer à B un haché $\mathsf{hash}(A, d_A, k_A, r)$ de son identité, de sa donnée, d'un nonce fort qu'il aura généré, k_A, et du nom faible r. Une fois reçu B fera de même avec sa propre identité, sa donnée, un nonce fort, k_B, qu'il aura lui aussi généré, et le nom faible r. Puis A révèle à B son nonce fort. Avec cette information B peut vérifier sa donnée d_B par rapport au haché qu'il a précédemment reçu en construisant le haché : $\mathsf{hash}(A, d_B, k_A, r)$. Il va signifier à l'utilisateur s'il valide ou non ce test puis révéler son nonce fort k_B. L'appareil A va effectuer le même test pour sa donnée puis va signifier à l'utilisateur s'il accepte ou rejette ce test. L'utilisateur va prendre connaissance des deux acquittements avant de signifier aux deux dispositifs qu'ils sont d'accord ou non sur la donnée partagée.

Nous avons vérifié ce protocole avec AKISS afin de prouver que la propriété d'authentification est vérifiée. Nous avons, comme dans les sections précédentes également simplifié le protocole et vérifié si la propriété de sécurité était toujours satisfaite. L'outil AKISS développé au cours de cette thèse nous a permis de trouver une version du protocole comprenant une interaction de moins avec l'utilisateur tout en conservant les propriétés de sécurité. L'appareil B peut ne pas afficher (envoyer le message $m_{Ba/r}$) à l'utilisateur pour signifier si oui ou non il a accepté le message, l'appareil B ne continuant pas le protocole dans le cas contraire, nous n'atteindrons pas le cas où l'un des deux appareils accepte la session malgré une donnée falsifiée. Cependant cette action d'affichage, bien que n'impactant pas la sécurité du protocole reste nécessaire dans le cas pratique où l'on veut que l'utilisateur puisse vérifier l'état de B. De la même manière l'utilisateur n'a pas besoin de notifier à l'appareil A que B a bien accepté la donnée. Par contre il faut qu'il signifie à B si A a accepté la donnée.

Nous avons également vérifié différents scénarios comme précédemment (voir tableau de la Figure 6.17). Le cas où B n'envoie pas de message d'acquittement reste correct comme expliqué plus haut. Nous avons également vérifié le cas sans *start*, et cette fois ci, la suppression de l'acquittement du fait que les protocoles sont prêts n'impacte pas la sécurité du protocole. En effet, l'attaquant peut intercepter m_A, mais il n'obtiendra pas pour autant d'information sur la clé k_A, car A va attendre une valeur

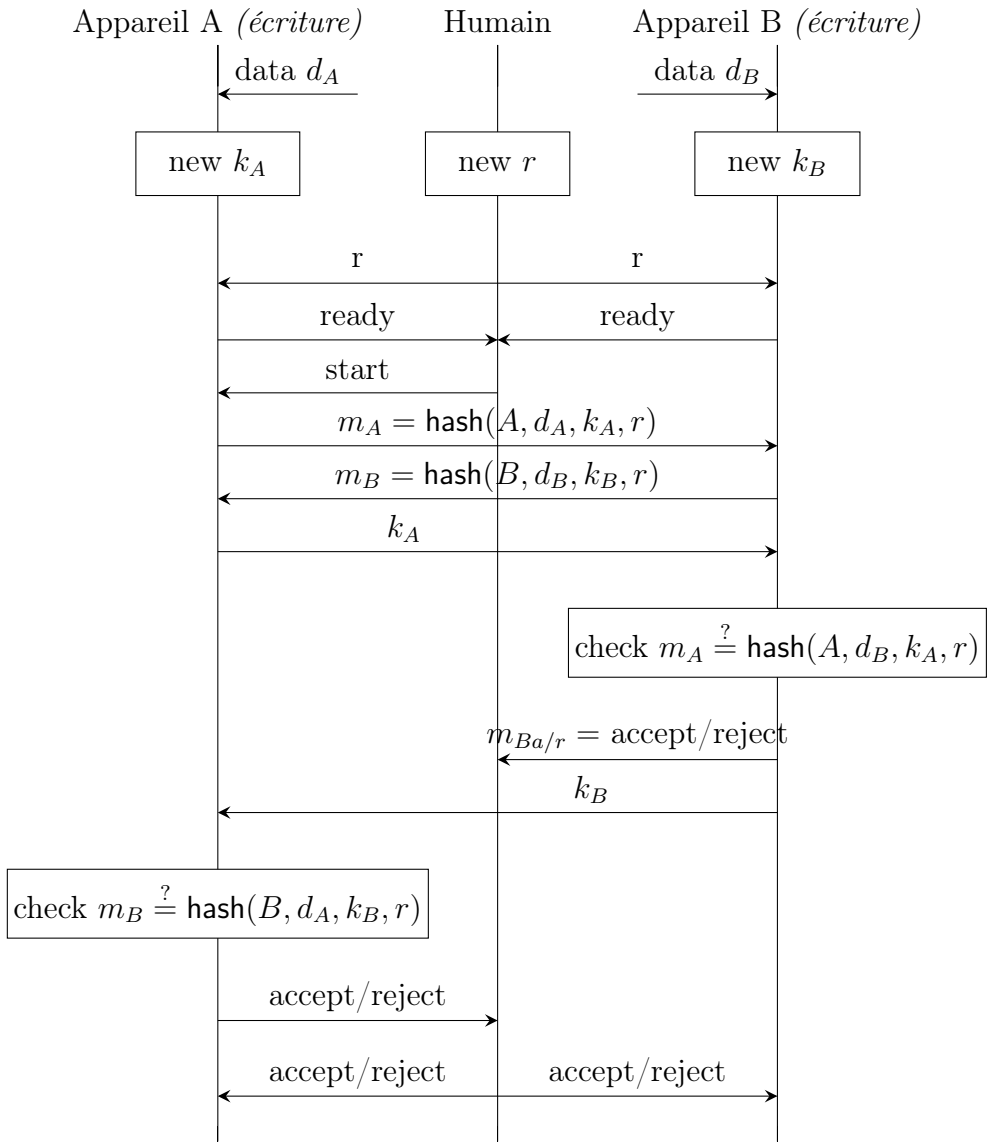


FIGURE 6.16 – Mécanisme d'authentification manuelle 7a

pour m_B avant de transmettre la clé. Cependant cette action reste utile en pratique pour que les appareils ne se synchronisent pas sans l'accord de l'utilisateur. Si une des clés de l'un des appareils est faible nous trouvons des attaques, car le haché envoyé par le participant correspondant peut être changé par l'attaquant sans que l'autre participant le discerne. Nous pouvons également voir que l'identité dans la fonction de hachage est indispensable. Sans elle, l'attaquant peut leurrer les participants. Quand A va envoyer $m'_A = \mathsf{hash}(d_A, k_A, r)$ à B, l'attaquant va transmettre le message mais ensuite quand B va transmettre $m'_B = \mathsf{hash}(d_B, k_B, r)$ l'attaquant pourra renvoyer à A, m'_A, par la suite l'attaquant interceptera k_B pour transmettre k_A à A et A pensera terminer sa session avec la bonne valeur pour k_B alors qu'il la terminera avec k_A. Nous n'avons pu vérifier l'exemple deux sessions qui nécessite un nombre trop important de traces à générer et a provoqué un dépassement de la mémoire disponible. Ce problème

pourrait peut-être être corrigé en faisant des optimisations mémoire dans l'outil AKISS.

scénario		# traces	temps	statut
une session	Auth. A	777	< 15s	ok
	Auth. B	915	< 15s	ok
sans $m_{Ba/r}$	Auth. A	926	< 3s	ok
	Auth. B	735	< 3s	ok
sans *start*	Auth. A	15423	< 4m	ok
	Auth. B	15701	< 4m	ok
k_A faible	Auth. A	3075	< 10s	attaque
	Auth. B	3599	< 10s	attaque
k_B faible	Auth. A	3075	< 10s	attaque
	Auth. B	3599	< 10s	attaque
hash sans identité	Auth. A	777	< 8s	attaque
	Auth. B	915	< 8s	attaque

FIGURE 6.17 – Résultats pour le mécanisme d'authentification 7a

Le mécanisme 7a permet d'authentifier nos données en utilisant uniquement des appareils d'écriture. De ce fait c'est l'utilisateur qui va générer un nom faible. AKISS modifié permet de prendre ce cas en compte, considérant l'utilisateur comme un participant actif du protocole. Ainsi, comme présenté dans les Figures 6.7 et 6.4 l'utilisateur est modélisé comme un processus.

Appareil à interface de lecture communiquant avec un appareil à interface d'écriture : Le mécanisme 8a utilise un appareil à interface de lecture et un appareil à interface d'écriture. Le mécanisme 8a, détaillé en Figure 6.18, se différencie par le fait que l'utilisateur n'a pas besoin ici de générer un nonce faible, c'est l'appareil A qui va s'en charger et l'afficher sur son interface. Comme pour le mécanisme 7a nous vérifions plusieurs scénarios, ces résultats sont présentés dans le tableau de la Figure 6.19.

Nous retrouvons dans le tableau de la Figure 6.19 des résultats similaires au mécanisme précédent.

scénario		# traces	temps	statut
une session	Auth. A	2542	< 15s	ok
	Auth. B	1946	< 15s	ok
sans *start*	Auth. A	27028	6m	ok
	Auth. B	40741	< 8m30	ok
k_A faible	Auth. A	9586	< 20s	attaque
	Auth. B	7202	< 15s	attaque
k_B faible	Auth. A	9586	< 20s	attaque
	Auth. B	7202	< 15s	attaque
hash sans identité	Auth. A	2542	< 15s	attaque
	Auth. B	1946	< 10s	attaque

FIGURE 6.19 – Résultats pour le mécanisme d'authentification 8a

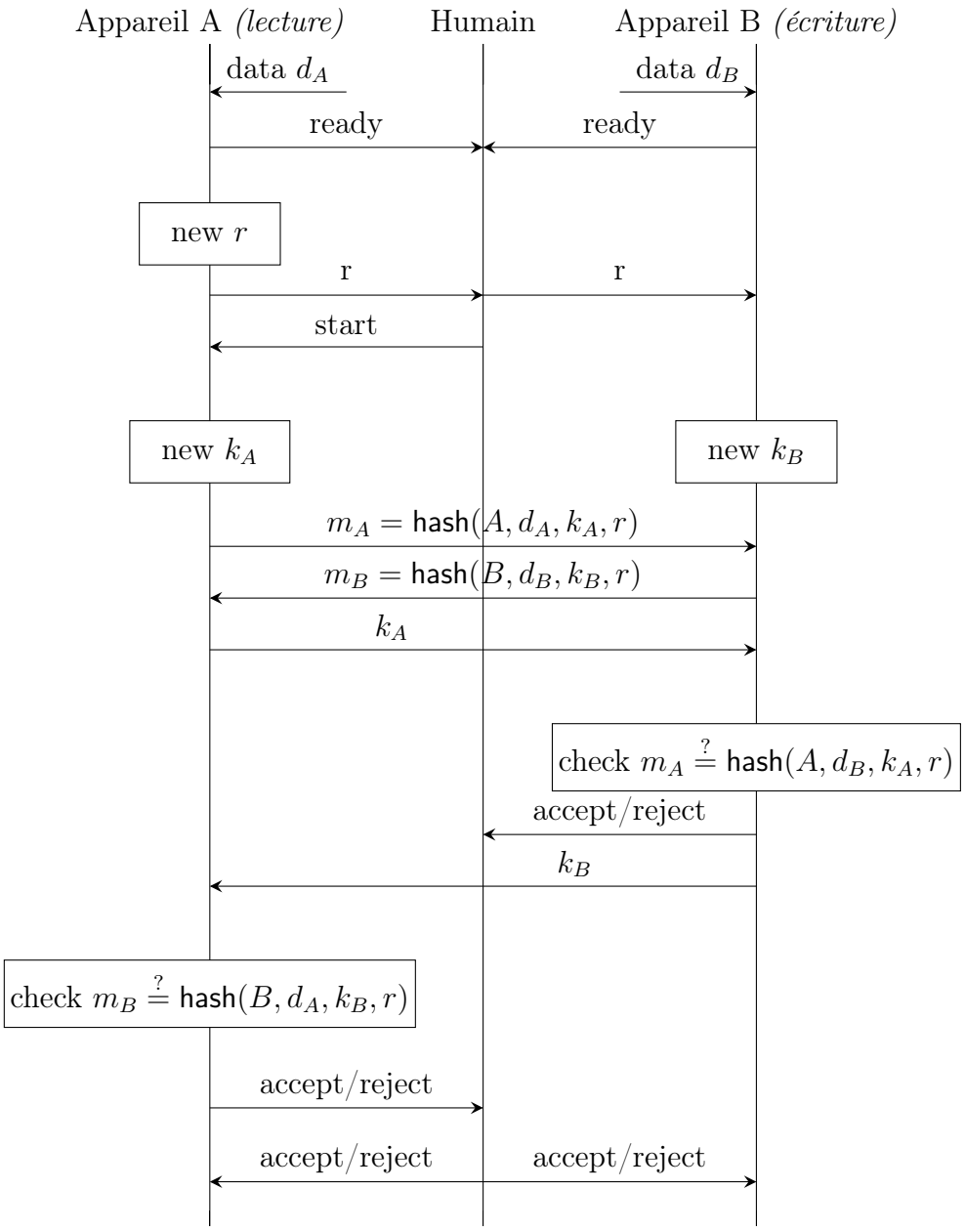


FIGURE 6.18 – Mécanisme d'authentification manuelle 8a

L'outil AKISS modifié que nous avons présenté dans le Chapitre 5 nous a permis de vérifier la majorité des protocoles du standard ISO9798-6 [ISO10]. Les seuls protocoles que nous n'avons pas vérifiés sont les mécanismes 7b et 8b : ce sont des variantes de 7a et 8a, mais qui opèrent en n tours, n étant un paramètre. De ce fait, ces protocoles sont hors de la portée de l'outil actuel qui ne permet pas de vérification paramétrée. Ces protocoles utilisaient des noms faibles et des fonctions de hachage faibles afin d'assurer des propriétés d'authentification. Notre implémentation qui nous autorise à modéliser à la fois des noms faibles et des fonctions de hachage faibles nous a permis de vérifier les protocoles de la Section 6.3 qui utilisent ces deux particularités. Ainsi, ces protocoles

du standard ISO/IEC 9798-6 ont été vérifiés jusqu'à deux sessions parallèlement, et sont donc sûrs sous ces hypothèses (voir les tableaux de résultat pour les détails des vérifications).

Conclusion

Cette thèse présente ma contribution au domaine de la vérification automatique de protocoles cryptographiques. Elle se concentre sur la vérification de propriétés d'authentification pour des protocoles utilisant de courtes chaines authentifiées. Afin de vérifier ces protocoles, nous avons proposé une procédure de décision étendant celle utilisée par l'outil AKISS [CcCK12]. Cette procédure est implémentée dans une nouvelle version d'AKISS [AkG], avec laquelle nous avons vérifié la plupart des protocoles du standard ISO/IEC 9798-6 [ISO10] ayant trait aux mécanismes de transfert manuel de données. Nous rappelons ici les contributions principales.

Un nouveau modèle symbolique pour raisonner sur les courtes chaines : Dans le Chapitre 1, prenant en compte l'état de l'art notamment concernant les définitions utilisées pour modéliser la vérification d'attaques par dictionnaire [CDK11, CDE05], nous avons exprimé une version de ces définitions adaptée à un attaquant pouvant deviner des secrets faibles (Définition 4). Nous inspirant de l'approche proposée par Chothia *et al.*, dans [CSS15] nous avons modélisé un attaquant pouvant trouver des collisions sur des fonctions de hachage faibles (Section 1.2). Dans le Chapitre 2, nous avons proposé une sémantique définissant un attaquant qui, devinant un secret faible, peut l'ajouter à sa connaissance afin de vérifier des propriétés d'authentification. Un tel attaquant est adapté à la vérification de protocoles hors-bande. En effet, dans ces protocoles, la sécurité n'est pas dépendante du simple fait qu'un attaquant peut deviner un secret faible ou trouver une collision sur une fonction de hachage faible : dans ces protocoles, en général de type *engagement avant connaissance*, le moment du protocole où l'attaquant peut deviner le secret faible ou trouver une collision va être déterminant quant à la sécurité du protocole. Ainsi, nous exprimons comment nous modélisons des propriétés d'authentification pour ce type de protocoles (Section 2.4). Dans le Chapitre 3, nous présentons une traduction de la modélisation du Chapitre 2 en un ensemble de traces. Ainsi, il nous suffit maintenant de vérifier des propriétés d'atteignabilité sur les traces.

Une procédure de décision pour un nombre borné de sessions : Dans le Chapitre 4, nous avons présenté une procédure pour décider de propriétés d'authentification sur des traces considérant un attaquant pouvant deviner des secrets faibles et trouver des collisions sur des fonctions de hachage faibles. À partir de clauses de Horn nous appliquons une étape de saturation que nous avons montré correcte et complète puis nous lui succédons l'application d'un algorithme dédié afin de vérifier l'atteignabilité de la dernière action d'une trace. Cet algorithme est lui aussi montré correct et complet. Cette procédure permet de vérifier si l'attaquant peut gagner des possibilités

de violation des propriétés d'authentification en exploitant les faiblesses inhérentes à l'utilisation de courtes chaines.

Une implémentation dans l'outil AKISS et la vérification de protocoles : Dans le Chapitre 5, nous détaillons les spécificités de l'implémentation basée sur ce travail. Ainsi, plusieurs optimisations ont été appliquées afin de réduire le nombre de traces générées de manière conséquente. Dans le Chapitre 6, nous présentons les protocoles du standard ISO/IEC 9798-6 :2010 sur les mécanismes de transfert manuel de données. Ces mécanismes, utilisant une interaction humaine, sont voués à ne transmettre que de courtes chaines. Ils ont la particularité d'utiliser des secrets faibles, des fonctions de hachage faibles ou une combinaison des deux. Ainsi, notre implémentation est la seule à notre connaissance permettant de vérifier de manière automatique de tels protocoles. Nous vérifions donc ces mécanismes et nous nous intéressons à quelques variations.

La vérification automatique de protocoles utilisant des secrets ou fonctions de hachage faibles n'entrait pas dans le cadre de vérification d'outils existants. Ainsi, avec nos travaux, nous pouvons vérifier de nombreux protocoles, et nous nous sommes attelés à la vérification de la plupart des protocoles du standard ISO/IEC 9798-6 :2010 [ISO10]. Bien d'autres protocoles de cette nature existent, e.g. 3D-Secure [3DS] largement utilisé dans le paiement en ligne. Ces travaux peuvent donc s'appliquer pour mener d'autres études de cas. En l'état, la vérification des protocoles du standard se limite à quelques sessions. En effet, le nombre de traces générés est trop important pour pouvoir mener à bien l'analyse. Afin de vérifier des scénarios plus complexes, et ainsi obtenir de meilleures garanties sur ces protocoles, il sera nécessaire d'intégrer des optimisations plus poussées dans la génération de traces. Une piste serait de paralléliser la procédure de génération de traces dans l'outil AKISS.

Des protocoles de vote électronique utilisent également de courtes chaines comme Du-Vote [GRCC15], Pretty Good Democracy [RT09] ou le protocole Norvégien [Gjø11]. Vérifier des propriétés d'authentification pour ces protocoles n'est pas suffisant pour s'assurer de leur bon comportement. Nos travaux ne peuvent donc pas s'appliquer en l'état à la vérification de protocoles de vote électronique. Une première difficulté est déjà de définir formellement ce que veut dire la propriété d'anonymat dans ce contexte. Ces propriétés d'anonymat s'expriment le plus souvent à l'aide d'une notion d'indistinguabilité, permettant d'assurer que deux situations sont équivalentes du point de vue de l'attaquant [AF01]. Dans un deuxième temps, il serait alors possible d'étendre notre procédure pour traiter des propriétés d'équivalence. Au vu des travaux récents et des difficultés pour traiter des propriétés d'équivalence dans les protocoles classiques, il s'agirait d'une extension non triviale. De plus, dans le cas de l'équivalence, de nombreuses optimisations qui permettent de réduire le nombre de traces générées ne seraient plus valides. Il faudrait dans ce cas approfondir les solutions qui permettent d'améliorer l'efficacité d'une telle procédure.

Bibliographie

[3DS] Visa 3-D Secure documentation. `https://technologypartner.visa.com/Library/3DSecure.aspx`.

[AAA+12] Alessandro Armando, Wihem Arsac, Tigran Avanesov, Michele Barletta, Alberto Calvi, Alessandro Cappai, Roberto Carbone, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Gabriel Erzse, Simone Frau, Marius Minea, Sebastian Mödersheim, David von Oheimb, Giancarlo Pellegrino, Serena Elisa Ponta, Marco Rocchetto, Michaël Rusinowitch, Mohammad Torabi Dashti, Mathieu Turuani, and Luca Viganò. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012*. Springer, 2012.

[ABB+02] Alessandro Armando, David A. Basin, Mehdi Bouallagui, Yannick Chevalier, Luca Compagna, Sebastian Mödersheim, Michaël Rusinowitch, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISS security protocol analysis tool. In *Computer Aided Verification, 14th International Conference, CAV 2002*. Springer, 2002.

[ABB+05] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification, 17th International Conference, CAV 2005*. Springer, 2005.

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella Béguelin, and Paul Zimmermann. Imperfect forward secrecy : How diffie-hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 2015*. ACM, 2015.

[ACC+08] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuéllar, and M. Llanos Tobarra. Formal analysis of SAML 2.0 web browser single sign-on : breaking the saml-based single sign-on for google apps.

In *Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering, FMSE 2008*. ACM, 2008.

[AES01] Advanced encryption standard (AES). *FIPS 197*, 2001.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2001*. ACM, 2001.

[AkG] Akiss repository - guess handling version. `https://github.com/LudovicRobin/akiss/`.

[Aki] Akiss repository. `https://github.com/akiss/akiss/`.

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In *Topics in Cryptology - The Cryptographers' Track at the RSA Conference 2005, CT-RSA 2005*. Springer, 2005.

[BAF05] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *20th Symposium on Logic in Computer Science LICS 2005*. IEEE Computer Society, 2005.

[Bau05] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. In *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005*. ACM, 2005.

[BDGK17] David Baelde, Stéphanie Delaune, Ivan Gazeau, and Steve Kremer. Symbolic verification of privacy-type properties for security protocols with XOR. In *30th Computer Security Foundations Symposium, CSF 2017*. IEEE Computer Society, 2017.

[BDS15] David A. Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 2015*. ACM, 2015.

[Bla01] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *14th Computer Security Foundations Workshop CSFW 2001*. IEEE Computer Society, 2001.

[Bla09] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4) :363–434, 2009.

[Bla16] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2) :1–135, 2016.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange : password-based protocols secure against dictionary attacks. In *Symposium on Security and Privacy, S&P 1992*. IEEE Computer Society, 1992.

[BMP00] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Advances in Cryptology - International Conference on the Theory and Application of Cryptographic Techniques, EUROCRYPT 2000*. Springer, 2000.

[BS01] Franz Baader and Wayne Snyder. Unification theory. In *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

[BTH01] Specification of the bluetooth system. Technical report, Bluetooth Special Interest Group, 2001.

[CB13] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. In *Principles of Security and Trust - 2nd International Conference, POST 2013*. Springer, 2013.

[CcCK12] Rohit Chadha, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. In *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012*. Springer, 2012.

[CCD17] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. A procedure for deciding symbolic equivalence between sets of constraint systems. *Inf. Comput.*, 255 :94–125, 2017.

[CD05] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property : How to get rid of some algebraic properties. In *Term Rewriting and Applications, 16th International Conference, RTA 2005*. Springer, 2005.

[CDE05] Ricardo Corin, Jeroen Doumen, and Sandro Etalle. Analysing password protocol security against off-line dictionary attacks. *Electr. Notes Theor. Comput. Sci.*, 121 :47–63, 2005.

[CDK11] Céline Chevalier, Stéphanie Delaune, and Steve Kremer. Transforming password protocols to compose. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[CGGI14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election verifiability for helios under weaker trust assumptions. In *19th European Symposium on Research in Computer Security, ESORICS 2014*. Springer, 2014.

[CKRT03] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR. In *18th Symposium on Logic in Computer Science LICS 2003*. IEEE Computer Society, 2003.

[CMAFE03] Ricardo Corin, Sreekanth Malladi, Jim Alves-Foss, and Sandro Etalle. Guess what ? here is a new tool that finds some new guessing attacks. In *Workshop on Issues in the Theory of Security, WITS 2003*, 2003.

[CS03] Hubert Comon-Lundh and Vitaly Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *18th Symposium on Logic in Computer Science, LICS 2003*. IEEE Computer Society, 2003.

[CS10] Tom Chothia and Vitaliy Smirnov. A traceability attack against e-passports. In *Financial Cryptography and Data Security, 14th International Conference, FC 2010*. Springer, 2010.

[CSS15] Tom Chothia, Ben Smyth, and Christopher Staite. Automatically checking commitment protocols in proverif without false attacks. In *Principles of Security and Trust - 4th International Conference, POST 2015*. Springer, 2015.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Information Theory*, 22(6) :644–654, 1976.

[Dis13] Stephen T. Dispensa. Multi factor authentication patent, 2013.

[DJ06] Stéphanie Delaune and Florent Jacquemard. Decision procedures for the security of protocols with probabilistic encryption against offline dictionary attacks. *J. Autom. Reasoning*, 36(1-2) :85–124, 2006.

[DKP12] Stéphanie Delaune, Steve Kremer, and Daniel Pasaila. Security protocols, constraint systems, and group theories. In *Automated Reasoning - 6th International Joint Conference, IJCAR 2012*. Springer, 2012.

[DKR08] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Composition of password-based protocols. In *21st Computer Security Foundations Symposium, CSF 2008*. IEEE Computer Society, 2008.

[DKR17] Stéphanie Delaune, Steve Kremer, and Ludovic Robin. Formal verification of protocols based on short authenticated strings. In *30th IEEE Computer Security Foundations Symposium, CSF 2017*. IEEE Computer Society, 2017.

[DRS08] Stéphanie Delaune, Mark Ryan, and Ben Smyth. Automatic verification of privacy properties in the applied pi calculus. In *Trust Management II - Proceedings of joint iTrust and PST Conferences on Privacy, Trust Management and Security, IFIPTM 2008*. Springer, 2008.

[Dup13] Hélène Dupuy. Le classement 2013 des pires mots de passe, 2013. `https://www.lesechos.fr/20/01/2014/lesechos.fr/0203255092785_le-classement-2013-des-pires-mots-de-passe.htm`.

[DY83] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2) :198–207, 1983.

[ESM10] Santiago Escobar, Ralf Sasse, and José Meseguer. Folding variant narrowing and optimal variant termination. In *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010*. Springer, 2010.

[FMS01] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of RC4. In *Selected Areas in Cryptography, 8th Annual International Workshop, SAC 2001*. Springer, 2001.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Symposium of Advances in Cryptology, CRYPTO 1984*. Springer, 1984.

[GD95] Patrick Gallagher and Acting Director. Secure hash standard (shs). *FIPS PUB*, pages 180–3, 1995.

[Gjø11] Kristian Gjøsteen. The norwegian internet voting protocol. In *E-Voting and Identity - 3rd International Conference, VoteID 2011*. Springer, 2011.

[GK17] Ivan Gazeau and Steve Kremer. Automated analysis of equivalence properties for security protocols using else branches. In *22nd European Symposium on Research in Computer Security, ESORICS 2017*. Springer, 2017.

[GMN04] Christian Gehrmann, Chris J Mitchell, and Kaisa Nyberg. Manual authentication for wireless devices. *RSA Cryptobytes*, 7(1) :29–37, 2004.

[GRCC15] Gurchetan S. Grewal, Mark Dermot Ryan, Liqun Chen, and Michael R. Clarkson. Du-vote : Remote electronic voting with untrusted computers. In *28th Computer Security Foundations Symposium, CSF 2015*. IEEE Computer Society, 2015.

[Hbd13] CVE-2014-0160. Heartbleed bug., December 3 2013.

[HM05] Changhua He and John C. Mitchell. Security analysis and improvements for IEEE 802.11i. In *Network and Distributed System Security Symposium, NDSS 2005*. The Internet Society, 2005.

[ISO10] ISO/IEC 9798-6 :2010 information technology – security techniques – entity authentication – part 6 : Mechanisms using manual data transfer, 2010.

[ISO11] ISO/IEC 9797-1 :2011 information technology – security techniques – message authentication codes (macs) – part 1 : Mechanisms using a block cipher, 2011.

[KT11] Ralf Küsters and Tomasz Truderung. Reducing protocol analysis with XOR to the xor-free case in the horn theory based approach. *J. Autom. Reasoning*, 46(3-4) :325–352, 2011.

[Low96] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using FDR. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, 2nd International Workshop, TACAS 1996*. Springer, 1996.

[Low97] Gavin Lowe. A hierarchy of authentication specification. In *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*, pages 31–44. IEEE Computer Society, 1997.

[Low04] Gavin Lowe. Analysing protocol subject to guessing attacks. *Journal of Computer Security*, 12(1) :83–98, 2004.

[NDS99] J.C. Mitchell N.A. Durgin, P.D. Lincoln and A. Scedrov. Undecidability of bounded security protocols. In *Formal Methods and Security Protocols Workshop FMSP 1999*, 1999.

[NR11] Long Hoang Nguyen and A. W. Roscoe. Authentication protocols based on low-bandwidth unspoofable channels : A comparative survey. *Journal of Computer Security*, 19(1) :139–201, 2011.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12) :993–999, 1978.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2) :120–126, 1978.

[RSN12] A.W. Roscoe, T. Smyth, and L. Nguyen. Model checking cryptographic protocols subject to combinatorial attack. 2012. Unpublished manuscript, available at `http://www.cs.ox.ac.uk/publications/publication5266-abstract.html`.

[RT01] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with finite number of sessions is np-complete. In *14th Computer Security Foundations Workshop, CSFW 2001*. IEEE Computer Society, 2001.

[RT09] Peter Y. A. Ryan and Vanessa Teague. Pretty good democracy. In Bruce Christianson, James A. Malcolm, Vashek Matyas, and Michael Roe, editors, *Security Protocols XVII, 17th International Workshop*. Springer, 2009.

[Shm04] Vitaly Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. In *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004*. Springer, 2004.

[SMCB12] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In *25th Computer Security Foundations Symposium, CSF 2012*. IEEE Computer Society, 2012.

[SSL] *OpenSSL - Cryptography and SSL/TLS toolkit*. `https://www.openssl.org`.

[VP17] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks : Forcing nonce reuse in WPA2. In *2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017*. ACM, 2017.

[ZDW06] Zhu Zhao, Zhongqi Dong, and Yongge Wang. Security analysis of a password-based authentication protocol proposed to IEEE 1363. *Theor. Comput. Sci.*, 352(1-3) :280–287, 2006.