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

Sécurité des protocoles cryptographiques : décidabilité et complexité

THÈSE

présentée et soutenue publiquement le 11 décembre 2003

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1

(spécialité informatique)

par

Mathieu TURUANI

Composition du jury

Président :	Jean-Pierre JOUANNAUD	
Rapporteurs :	Roberto AMADIO	Professeur de l'Université de Provence
	Hubert COMON LUNDH	Professeur ENS-Cachan
Examinateurs :	Jean-Pierre JOUANNAUD	Professeur Ecole Polytechnique
	Birgit PFITZMANN,	Professeur Docteur IBM Zurich
	Michael RUSINOWITCH,	Directeur de recherche INRIA (dir. de thèse)
	André SCHAFF	Professeur Université Henry Poincaré

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

Mis en page avec la classe thloria.

Sommaire

Table des figures

Liste des tableaux

1

Introduction

Sommaire

L'essor de l'informatique dans tous les secteurs de notre société s'est fait conjointement avec son utilisation de plus en plus massive dans toutes nos communications. D'une part, nous remplaçons de plus en plus une communication directe de personne à personne par une communication à distance au travers d'un système dont on n'a pas un contrôle absolu (téléphone, courrier électronique, etc...). D'autre part, nous laissons de plus en plus des systèmes automatiques effectuer des tâches de communication critiques sans aucune intervention humaine (typiquement un paiement électronique au lieu d'un échange d'espèces). Bien que beaucoup plus rapides et moins contraignants qu'une communication directe, ces nouveaux genres de communications n'assurent plus par nature les propriétés de sécurités nécessaires à toute communication importante. Par exemple, une tierce personne peut assez facilement écouter une communication téléphonique (problème du secret de la communication), ou envoyer un courrier électronique au nom de quelqu'un d'autre (SPAM, virus transmis par courrier électronique, et en fait problème d'identification de l'utilisateur d'un service).

En pratique, on résout généralement ces problèmes en chiffrant un message à transmettre pour que seul le destinataire officiel puisse en prendre connaissance, et en signant ce message pour prouver qu'il a été créé par l'envoyeur officiel. Cependant, bien qu'absolument nécessaires à toute communication importante, les algorithmes de chiffrement ne suffisent pas à sécuriser de manière satisfaisante une communication complexe entre plusieurs participants, comportant de nombreux échanges de messages, et désignée ici par un protocole de communication. Intuitivement, une attaque sur un protocole de communication est la possibilité pour un participant du protocole ou une personne extérieure d'obtenir une information qu'il ne devrait pas connaître, ou de se faire passer pour quelqu'un d'autre. Dans ce but, la personne malhonnête va exploiter toutes les caractéristiques du protocole pour détourner certains messages, les modifier dans la mesure du possible (i.e. sans "casser" de clef), et les renvoyer à d'autres personnes sous une fausse

identité. Ceci peut lui permettre d'obtenir, par exemple, une clef secrète qu'il n'aurait pas du connaître, et donc de modifier davantage encore les messages qu'il intercepte. Au final, la personne malhonnête peut dans certains cas détourner suffisamment le protocole pour obtenir une information critique. On parle alors d'attaque sur ce protocole, alors qu'à aucun moment la personne malhonnête (aussi appelée intrus ou intercepteur) n'a eu besoin de calculer une clef privée à partir d'une clef publique, ou de calculer un message en clair à partir d'un message chiffré sans la clef de déchiffrement. En résumé, il peut exister de nombreuses attaques sur un protocole même quand les fonctions de chiffrement sont parfaites.

On trouve dans la littérature de nombreuses attaques découvertes sur des protocoles réputés sûrs lors de leur création, même si l'on considère que les fonctions de chiffrement utilisées sont parfaites. L'exemple le plus connu est celui du protocole de Needham-Schröder (c.f. [74]), dont on a cru longtemps qu'il était sûr alors qu'en fait il permet de nombreuses attaques relativement simples (c.f. [57]). Une attaque sur un protocole de communication peut avoir de lourdes conséquences si elle n'est découverte qu'après la mise en service du protocole à grande échelle. En effet, à la différence d'un participant humain, un système automatique ne s'adapte pas de lui même dès qu'une faille est découverte mais nécessite au mieux un remplacement du logiciel, au pire un remplacement du matériel (par exemple le renouvellement de toutes les cartes bleues). Ceci peut prendre beaucoup de temps, et donc créer de nombreux problèmes.

1.1 Protocoles cryptographiques

1.1.1 Protocoles de communication

Un protocole de communication peut être décrit comme un échange (généralement complexe) de messages entre plusieurs participants. Dans la littérature, ces échanges de messages sont habituellement décrits par la liste des actions réalisées par chaque participant lors d'une exécution normale du protocole, c'est à dire sans aucune intervention de personne malhonnête. Par exemple, la figure 1.1 décrit un protocole très simple d'achat sur internet, heureusement non utilisé en pratique. Ce protocole est constitué de cinq échanges de messages entre trois participants : un client, un marchant, et une banque recevant les ordres de paiement. Ce protocole n'est qu'un exemple et ne doit pas être utilisé tel quel dans la réalité car il ne satisfait à aucune des propriétés de sécurité les plus évidentes que l'on attend d'un protocole de paiement. Par exemple, l'identité du client Alice n'est absolument pas vérifiée par la banque. Ainsi, n'importe qui pourrait envoyer un message à la banque en se faisant passer pour Alice et donner un ordre de paiement d'Alice vers lui-même.

Il est souvent intéressant d'étudier plusieurs instances d'un même protocole cryptographique. Pour cette raison, on appelle session de protocole un ensemble d'échanges de messages entre plusieurs participants formant un ensemble cohérent et pouvant être répété. Par exemple, la figure 1.1 présente une session de ce protocole d'achat. Vérifier plusieurs sessions d'un protocole permet de modéliser par exemple plusieurs clients se connectant en même temps à un serveur (plusieurs sessions en parallèle), ou un même client répétant plusieurs fois de suite une session donnée (plusieurs achats, par exemple). Pour différencier deux sessions différentes, on introduit généralement dans le protocole des nombres aléatoires générés par les agents, appelés nonces. Un nonce est une donnée de grande taille choisie au hasard par un agent. A cause de cette taille, on suppose qu'il est impossible pour tout autre agent de choisir par hasard un nonce de même valeur, comme il lui est impossible d'énumérer tous les nonces possibles. Dans le cas d'un achat sur internet, le numéro de commande (s'il est choisi au hasard) est un exemple de nonce : il permet d'identifier les achats d'un client donné, et il y a trop de numéros de commandes

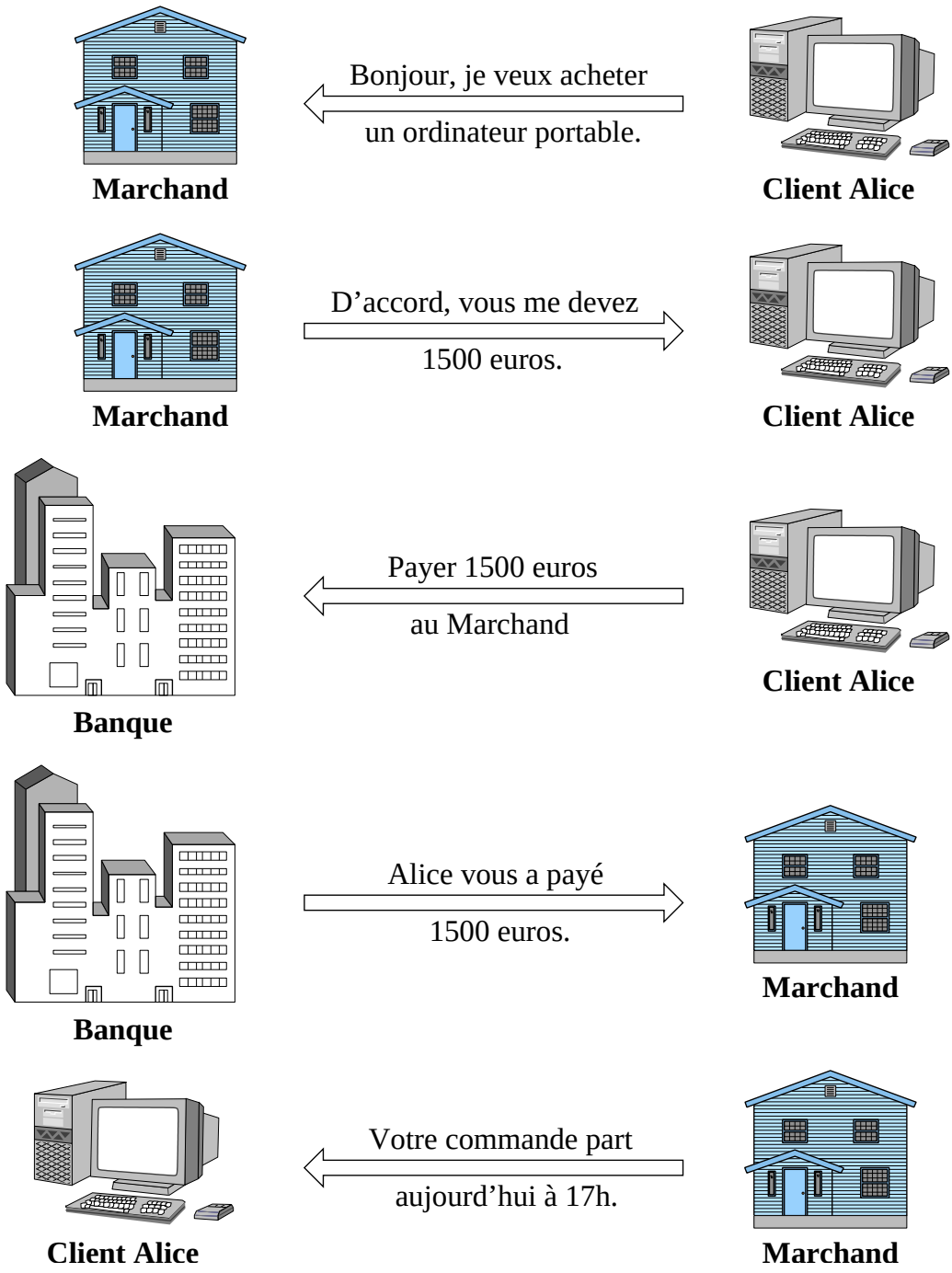


FIG. 1.1: Exemple de protocole d'achat très simple.

possibles pour qu'un agent extérieur voulant accéder aux commandes d'alice puisse choisir le bon numéro de commande.

Par extension, on peut modéliser des protocoles à nombre de sessions non borné. Pour un protocole client/serveur, cela revient à ne pas borner à priori le nombre de clients connectés, le nombre d'actions effectuées par les clients, etc... Ce type de situations peut être très difficile à vérifier, dans la mesure où chaque nouvelle session peut introduire de nouveaux nonces, et peut être exécutée d'une manière différente des sessions précédentes.

1.1.2 Primitives cryptographiques

Pour communiquer et créer des messages, les agents utilisent un certain nombre d'outils, ou primitives cryptographiques. Ces primitives sont essentiellement de trois types.

Concaténation :

Le premier type d'opérateur, et le plus simple, est l'opérateur de concaténation de messages, noté $\langle .., .. \rangle$. Ainsi, $\langle M_1, M_2 \rangle$ est la concaténation des messages M_1 et M_2, par exemple un montant à payer (1500 euros) suivi d'un numéro de commande, du nom du destinataire (le marchand), et du nom du client (Alice). En pratique, l'opérateur de concaténation vérifie souvent des propriétés comme l'associativité. Il serait donc intéressant de vérifier des protocoles en tenant compte de cette propriété. Cependant, en pratique un agent attendant un message M_1 suivi d'un message M_2 (i.e. $\langle M_1, M_2 \rangle$) connaît nécessairement la taille de M_1 et de M_2 (au pire, cette taille est inscrite dans le message reçu). Ainsi, il ne peut pas confondre $\langle M_1, \langle M_2', M_2'' \rangle \rangle$ (si $M_2 = \langle M_2', M_2'' \rangle$) avec $\langle \langle M_1, M_2 \rangle, M_3 \rangle$, ce qui limite l'usage de l'associativité de $\langle ... \rangle$. Dans toute la suite, nous considérerons que l'opérateur de concaténation n'est pas associatif.

Chiffrement (ou encryption) :

Le second type d'opérateur est l'opérateur d'encryption symétrique ou asymétrique, aussi appelé chiffrement. Rappelons qu'un chiffrement est l'application d'un algorithme (de chiffrement) sur un message et une clef (dite de chiffrement) de manière à ce qu'il soit impossible, ou au moins très difficile, de retrouver le message d'origine à partir du texte chiffré uniquement. L'opération inverse, appelée déchiffrement, consiste à appliquer une fonction de déchiffrement sur un texte chiffré et une clef (dite de déchiffrement, ou clef inverse), et produit le message d'origine. La signature fonctionne de manière similaire. Quand la clef de déchiffrement n'est pas calculable à partir de la clef de chiffrement, on parle d'encryption privée/publique, car ce type de chiffrement est souvent utilisé en gardant la clef de déchiffrement privée et en publiant la clef de chiffrement (et inversement pour une signature). A l'opposé, quand on peut aisément calculer la clef de déchiffrement à partir de la clef de chiffrement, ou quand ces deux clefs sont identiques, on parle d'encryption symétrique. Suivant la terminologie de [88], déchiffrement et décryptage désigneront tous deux l'action de déchiffrer un texte chiffré avec la clef adéquate.

Il existe en pratique de nombreux types d'algorithmes de chiffrement, basés sur différentes propriétés arithmétiques ou algébriques. Par exemple, RSA [80] est basé sur quelques propriétés simples en arithmétique (congruences, algorithme d'Euclide pour calculer la clef de déchiffrement, etc...). La difficulté à calculer la clef privée à partir de la clef publique repose sur la difficulté à factoriser des produits de grands nombres premiers. Comme algorithme de chiffrement ou signature à clef privée/publique, on peut également citer les algorithmes de Schnorr [89] et ElGamal [46], tous deux basés sur des logarithmes discrets, dont le standard DSA est une variante. Une liste très détaillée d'algorithmes de chiffrement symétriques ou asymétriques peut

être trouvée dans [88]. Bien que tous ces algorithmes aient des structures différentes, ils fournissent tous à peu près la même interface : étant donné un nombre K (ou un ensemble de nombres) appelé clef, et un message M, on note $\{M\}_K$ le chiffrement ou la signature de M par K. La différence entre chiffrement à clef symétrique ou asymétrique dépend uniquement de la nature de la clef. On ne fait ici aucune différence entre chiffrement et signature. En effet, $\{M\}_K$ peut être vu comme un texte chiffré si K est publique mais K^{-1} reste privé, et peut être vu comme une signature si K reste privé mais K^{-1} est publique (pour vérifier la signature, i.e. décrypter $\{M\}_K$, et dans le cas d'une encryption asymétrique). En outre, on ne spécifie à aucun moment l'algorithme de chiffrement utilisé. On applique souvent l'hypothèse de chiffrement parfait, selon laquelle un texte chiffré ne possède pas d'autre propriété que de pouvoir être déchiffré avec la clef adéquate. Cependant, ceci n'est pas toujours satisfaisant : il existe par exemple de nombreuses méthodes de cryptanalyse permettant de décrypter un message ou de calculer une clef (attaques à texte connu ou choisi pour calculer une clef privée, attaque sur la méthode de chiffrement pour retrouver le texte en clair, etc...)

Opérateur algébrique :

Le dernier type d'opérateur que nous considérerons sera un opérateur algébrique, c'est à dire un opérateur possédant des propriétés algébriques non triviales ne pouvant pas être ignorées lors de l'analyse de la sécurité d'un protocole. Ce sont en général des fonctions mathématiques assez simples à décrire mais bien plus compliquées à vérifier. Par exemple, certains protocoles utilisent l'opérateur de ou exclusif bit à bit sur des messages, noté $a \oplus b$ pour deux messages a et b. D'une certaine manière, on pourrait voir cet opérateur comme une encryption symétrique de a avec la clef b. Mais ce ne serait pas satisfaisant, car ce serait oublier les nombreuses propriétés du ou exclusif. En particulier, même si l'on ne peut pas calculer a avec seulement $a \oplus b$, on peut calculer a si l'on connaît $a \oplus b$, $b \oplus c \oplus d$, $c \oplus a$ et $d \oplus a$. Un autre exemple représentatif est celui de l'opérateur d'exponentiation, dont les propriétés algébriques sont utilisées dans de nombreux protocoles (de groupe, par exemple). La section 2.4 présentera des primitives cryptographiques avancées, telles que le xor de deux messages bit à bit, l'exponentiation, et l'encryption à clefs privées/publiques commutatives, ainsi que des propriétés algébriques très intéressantes de ces opérateurs pour la recherche de nouvelles attaques.

Pour illustrer l'utilisation des primitives cryptographiques, la figure 1.2 présente un protocole très simplifié de partage de clef de session. Dans cet exemple, $PubAlice$ et $PubBob$ sont respectivement les clefs publiques d'Alice et de Bob, c'est à dire que seule Alice connaît la clef inverse $PubAlice^{-1}$ permettant de déchiffrer un message $\{M\}_{PubAlice}$, et seul Bob connaît la clef inverse $PubBob^{-1}$ permettant de déchiffrer un message $\{M\}_{PubBob}$. Le protocole se déroule en deux étapes. Tout d'abord, le serveur crée une clef symétrique $ClefAB$ et la transmet chiffrée à *Alice* et *Bob* à l'aide de leurs clefs publiques respectives. Puis, Alice et Bob s'envoient mutuellement des données secrètes en les chiffrant à l'aide de $ClefAB$. Ce protocole peut sembler relativement sûr : tous les messages étant chiffrés, un espion écoutant les communication ne peut pas calculer la clef symétrique $ClefAB$, et ne peut donc pas décrypter les communications secrètes entre Alice et Bob chiffrées avec cette clef. Pourtant, ce protocole peut être très facilement attaqué par un intrus actif, et n'est donc pas du tout sûr ! Par exemple, il suffit pour l'intrus d'intercepter les deux messages envoyés par le serveur, et de les remplacer par deux messages de sa propre création. Ceci est possible, car les noms *Alice* et *Bob*, ainsi que les clefs de chiffrement sont des données publiques. L'intrus peut donc envoyer deux "faux" messages à Alice et Bob, sous l'identité du serveur, avec un clef $ClefIntrus$ connue à la place de $ClefAB$. Les deux principaux utilisent alors la clef $ClefIntrus$ pour chiffrer toutes leurs communications

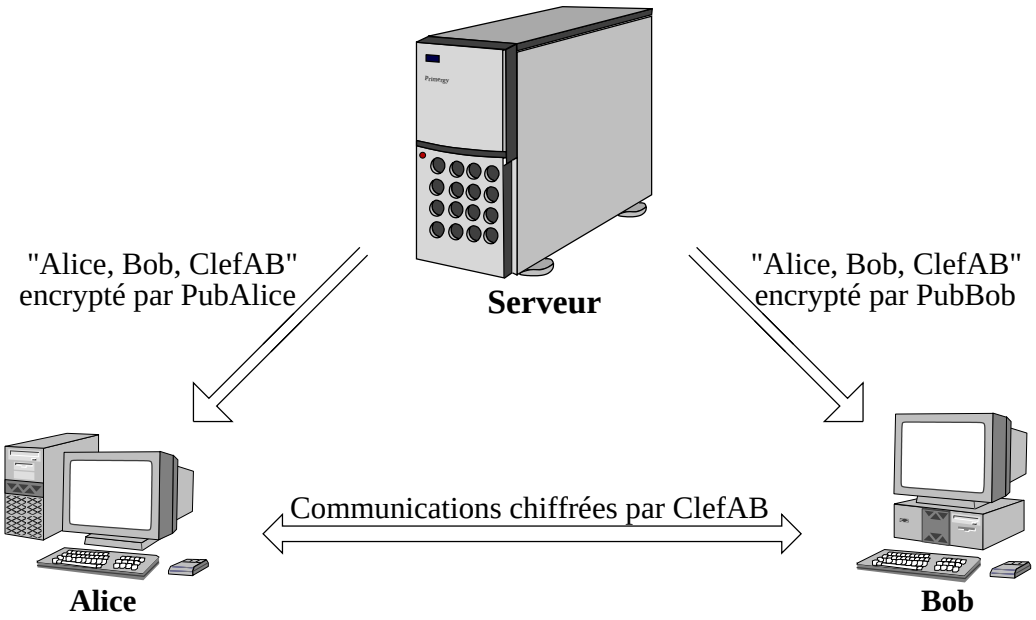


FIG. 1.2: Exemple de partage de clef simple.

secrètes, alors que cette clef est connue de l'intrus (puisque c'est lui qui l'a générée).

1.1.3 Agents (honnêtes) et intrus

Un protocole cryptographique est fondamentalement un protocole de communication entre personnes, machines, ou programmes, ayant un but de sécurité précis comme transmettre une donnée critique d'un point à un autre (par exemple des plans top-secrets), être certain de l'identité d'une personne ou d'une machine (par exemple pour accéder à un site Web confidentiel), ou garantir qu'un message a bien été créé par telle ou telle personne (par exemple un ordre de virement ou un paiement électronique). Dans tous les cas, un protocole est défini sur un certain nombre de participants, que l'on appelle agents. Ceux-ci peuvent être de deux types. Un agent honnête, ou principal, sera un participant "officiel" du protocole, c'est-à-dire une personne ou un programme ayant un comportement bien précis, prévu à l'avance, et décrit par la spécification du protocole qu'il exécute. A l'inverse un agent malhonnête, ou intrus, sera soit un participant non officiel du protocole, comme un espion, soit un participant officiel utilisant sa position avantageuse pour perpétrer des actions malhonnêtes. Dans ces deux cas, le ou les intrus doivent être capables d'intervenir dans les échanges de messages entre participants honnêtes pour mener à bien une attaque intéressante. Pour cela, on s'appuie souvent sur le modèle d'intrus de Dolev-Yao, initialement présenté dans [43] dans le cas des protocoles Ping-Pong. L'idée de ce modèle est de donner à l'intrus le plus de moyens possibles sans toutefois faillir à l'hypothèse de chiffrement parfait, c'est à dire sans permettre à l'intrus de décrypter un message sans connaître la clef de déchiffrement nécessaire. Bien que le modèle de protocoles que l'on utilise soit plus riche que le modèle de protocoles Ping-Pong de [43], on continue d'appeler intrus de Dolev-Yao un intrus de conception équivalente. Concrètement, tout intrus de Dolev-Yao peut intercepter, lire et retarder tout message transmis par un agent honnête sur un réseau publique. De plus, il peut décomposer tout message acquis de la sorte et utiliser les connaissances ainsi obtenues

pour construire de nouveaux messages. Enfin, il peut envoyer des messages sur le réseau sous n'importe quelle fausse identité. Ce sont les actions de base que nous considérerons dans le cas des protocoles sans opérateurs algébriques. Pour les protocoles avec opérateurs algébriques, nous étendrons les capacités de l'intrus de Dolev-Yao (c.f. section 2.4, modèle de Dolev-Yao étendu). Ce modèle d'intrus pouvant parfois sembler trop expressif, certaines restrictions seront discutées à la section 2.5.

1.2 Problème de l'insécurité de protocoles.

1.2.1 Propriétés à vérifier

Il existe de nombreuses propriétés de sécurité que l'on peut exiger d'un protocole. Sans vouloir donner une liste exhaustive de toutes les propriétés de sécurité considérées jusqu'ici, nous allons présenter certaines des propriétés les plus demandées sur des protocoles cryptographiques (c.f. [85] pour un liste assez complète de propriétés de sécurité).

Secret :

L'une des propriétés les plus connues est la propriété de secret, et se décline souvent en plusieurs versions. Intuitivement, le secret d'une donnée Sec est assuré dans un protocole quand il n'existe aucun moyen pour l'intrus de connaître Sec (i.e. en clair, non chiffré). Par exemple, le protocole minimaliste où $Alice$ envoie à Bob un message secret M chiffré par la clef publique de Bob assure naturellement le secret de M, puisque l'intrus ne peut pas décrypter le message transmis. La plupart du temps, cette notion du secret est amplement suffisante. Cependant, elle repose sur le fait que l'intrus ne peut pas énumérer toutes les valeurs de M (de même qu'il ne peut pas énumérer toutes les valeurs d'une clef). Ceci n'est pourtant pas toujours vrai, notamment dans le cas particulier des protocoles de vote électronique. En effet, si M représente le choix de vote d'un électeur, l'intrus connaissant tous les choix possibles ($Candidat1$, $Candidat2$, ...), il peut tous les chiffrer avec la clef publique de Bob, comparer le résultat avec le message transmis par $Alice$, et ainsi identifier le choix d'$Alice$. Cette version plus forte de la propriété de secret peut être formalisée sous la forme d'une équivalence observationnelle entre deux version du protocole utilisant deux atomes différents à la place de la donnée secrète (le secret est préservé ssi ces deux version sont indistinguables). Cette seconde formulation est surtout utilisée en spi-calcul (c.f. [2]). Par ailleurs, on peut adapter la propriété de secret à des données valables pendant un temps assez court. Ce sont des secret dits "à court terme", que l'intrus ne doit pas obtenir avant la fin de la session de protocole les ayant créés. En contre partie, ces données sont révélées à l'intrus à la fin de la session de protocole les ayant créées, et pourront donc être utilisées pour découvrir un des secrets à court terme de la session suivante.

Authentification :

Un second type de propriété de sécurité que doivent assurer les protocoles cryptographiques est l'authentification d'un participant. Informellement, il s'agit pour un agent d'être sûr de l'identité de son correspondant. Par exemple, une banque recevant un ordre de virement électronique voudra vérifier l'identité de son client. Pour cela, une méthode souvent utilisée consiste à envoyer au client un nonce encrypté par la clef publique du client. Si celui-ci parvient à décrypter ce message et à renvoyer le nonce, alors il possède la clef privée du client, ce qui prouve son identité. En outre, la clef privée n'a pas été dévoilée dans cette opération, pas même à la banque. On doit cependant faire attention à ce que l'intrus ne puisse pas utiliser le client comme oracle pour répondre au défis de la banque. Cette attaque très classique est décrite à la figure 1.3, avec N un nonce généré par la banque et réutilisé par l'intrus.

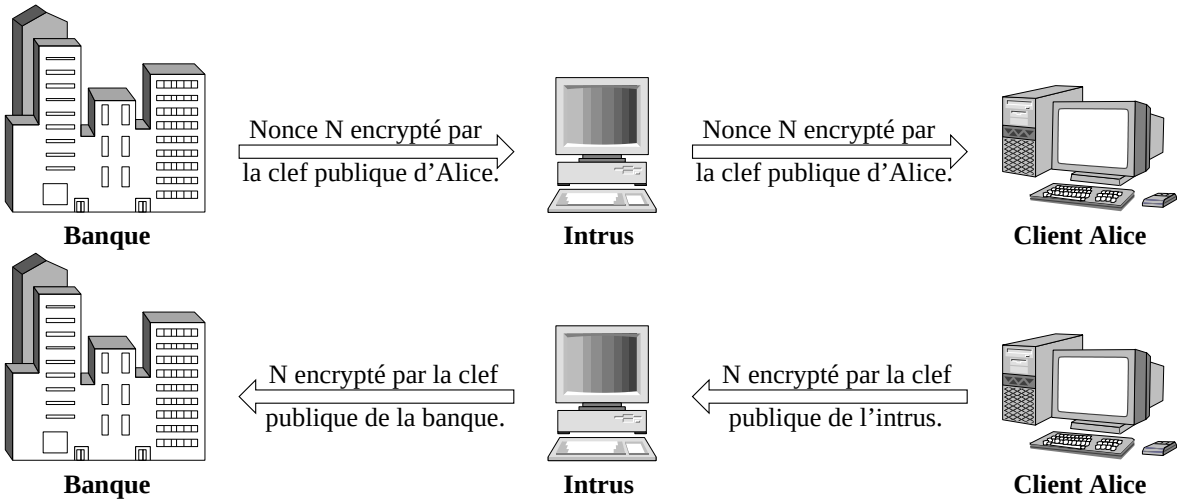


FIG. 1.3: Exemple d'attaque de l'intercepteur.

Le problème de cette propriété est qu'elle est très dépendante de ce que l'on entend par "vérifier une identité". En particulier, dans un média de communication ouvert où les messages peuvent avoir plusieurs intermédiaires, on ne communique jamais directement avec son correspondant. Ceci conduit à de nombreuses définitions différentes de la propriété d'authentification (voir par exemple [87] pour une liste de définitions). Par exemple, est-ce une attaque que de pouvoir rejouer avec a en se faisant passer pour b une ancienne communication entre deux principaux a et b, sans pouvoir la modifier ? Une formalisation intéressante d'une propriété d'authentification, issue de CAPSL (c.f. [67, 40]), consiste à identifier une donnée devant être transmise correctement d'un principal à un autre. Par exemple, si Alice crée une donnée Na et la transmet à Bob, on dit que Bob authentifie Alice grâce à Na si l'on peut certifier que la donnée Na reçue par Bob sera toujours la donnée Na transmise par Alice dans la même session. On remplace ainsi une propriété un peu floue sur les flux de messages par une propriété précise sur les connaissances de Bob, en se rapprochant des propriétés d'intégrité des données transmises (non modification). Il est intéressant de remarquer que pour l'une des nombreuses définitions de l'authentification, V. Bernat a prouvé qu'il existe une propriété de secret dont la validité assure la propriété d'authentification. Il serait très intéressant de pouvoir réduire certaines propriétés d'authentification et des propriétés de secret.

Disponibilité :

Les propriétés de secret et d'authentification sont les propriétés de sécurité les plus classiques sur les protocoles cryptographiques. Cependant, il existe d'autres propriétés tout aussi intéressantes mais nettement plus difficiles à formaliser proprement. Par exemple, on veut quelque fois vérifier des propriétés de disponibilité, comme la résistance (d'un serveur de pages Web, par exemple) aux attaques de type dénie de service. Ce type d'attaques est relativement simple à mettre en place. Elles consistent à saturer un serveur donné de requêtes jusqu'à ce qu'il ne puisse plus répondre à aucune (ou pire, qu'il plante). En pratique, ce type d'attaques correspond souvent à une inadéquation entre la capacité de traitement du serveur et le débit théorique maximal de requêtes qu'il peut recevoir (nettement supérieur au débit moyen), ou pire à un bogue dans le logiciel du serveur. De telles attaques ont déjà été perpétrées à grande échelle sur internet, notamment sur le site Yahoo. De plus, la présence d'un agent malhonnête

pas toujours nécessaire, comme pour la saturation des serveurs de paiement par Carte Bleue une veille de Noël à cause du trop grand nombre de clients. Le problème est qu'une propriété de disponibilité est difficilement formalisable (de manière générique). Il existe cependant plusieurs pistes pour vérifier certains aspects de la disponibilité. Par exemple, on peut spécifier un protocole en explicitant le temps de calcul nécessaire à chaque opération effectuée par le serveur, puis vérifier que l'intrus ne peut pas à moindre frais (i.e. avec peu de puissance de calcul) provoquer de longs calculs sur le serveur, ce qui le saturerait.

Anonymat :

Enfin, il existe plusieurs tentatives de formalisation d'autres propriétés spécifiques à certains protocoles ou certains contextes. Par exemple, une propriété d'anonymat consiste à ne pas pouvoir identifier les actions réalisées par un agent donné, ou inversement à ne pas pouvoir retrouver quel agent est à l'origine d'une action donnée. Par exemple, un utilisateur d'internet désire en général rester anonyme, c'est à dire éviter que les sites qu'il visite puissent l'identifier, pour éviter tout traçage commercial (ou en général non gouvernemental) des sites qu'il visite. Cependant, on veut en même temps qu'une organisation de contrôle puisse identifier chaque utilisateur d'internet en cas d'action frauduleuse sur le réseau, d'attaque sur un serveur, d'émission de SPAM, etc... Le même problème s'applique également aux téléphones portables, pour qu'un intrus ne puisse pas localiser et/ou identifier les utilisateurs, mais pour que la police puisse le faire. Des définitions formelles de propriétés d'anonymat n'ont été proposées que très récemment (voir par exemple [92], publié en 2002), et sont encore difficiles à vérifier.

1.2.2 État de l'art des méthodes de vérification.

Parmi les différentes propriétés de sécurité que nous venons de voir, nous allons nous concentrer sur le secret. Le problème est donc, étant donné un protocole cryptographique, de déterminer si une donnée particulière reste secrète on non. Malheureusement, si l'on ne pose aucune restriction sur les protocoles que l'on considère, ce problème est indécidable (S.Even et O.Goldreich, [47]). J. Mitchell *et al.* ont montré dans [45] que ce problème est indécidable même si l'on borne à priori la taille des messages transmis (vers et par les principaux). R. Amadio et W. Charatonik ont ensuite restreint davantage ce modèle de protocoles en ne permettant l'utilisation que d'une seule primitive cryptographique, à savoir l'encryption (c.f. [3]). Le problème de décision de la sécurité est encore indécidable dans ce cas. Une autre restriction consiste à borner le nombre de nonces utilisables dans une attaque. Cependant, plusieurs codages de problèmes indécidables comme PCP sont encore possibles dans ce cas (voir par exemple la thèse de V. Cortier [35]). Pour décider si un protocole donné est sûr, on doit donc soit se contenter d'une méthode partielle (semi-décision ou approximation), soit restreindre plus fortement le modèle de protocoles cryptographiques. Nous allons décrire différentes méthodes utilisées pour vérifier ces protocoles. Pour plus de précision, on peut se reporter aux survols réalisés par H. Comon et V. Shmatikov [33] (méthodes symboliques), G. Delzanno et P. Ganty [38] (méthodes interactives et approximations), ou C. Meadows [62, 64] (méthodes formelles et invariants). On trouve également des survols assez proches dans [21, 86]. Excepté pour le dernier groupe, les méthodes décrites ici n'utilisent pas d'opérateurs algébriques.

Méthodes d'analyse finie :

Une première idée pour décider la sécurité d'un protocole cryptographique consiste à restreindre drastiquement le modèle de protocole. En bornant à la fois le nombre de sessions et le nombre d'utilisations de l'opérateur d'encryption par l'intrus, on parvient à borner le nombre d'états de protocole accessibles à partir de l'état initial. On peut alors représenter le protocole

sous la forme d'une machine à nombre d'états finis (automate ou autre), et la propriété à vérifier sous la forme d'une formule de logique temporelle. Des outils de model-checking comme FDR [81], Murφ [71] ou Brutus [30] peuvent alors trouver certaines attaques sur le protocole. Cependant, ceci ne permet pas de prouver la sécurité d'un protocole dans le cas général. On peut néanmoins définir une classe restreinte de protocoles dans laquelle tout protocole non sûr admet une attaque à nombre de sessions et nombre d'opérateurs d'encryption utilisés par l'intrus bornés (c.f. Lowe [59]). Parmi les restrictions de [59], on trouve en particulier l'impossibilité pour un principal de créer une nouvelle clef à partir de connaissances acquises, ou de transmettre un message reçu qu'il n'a pas pu totalement décomposer. Ce sont des restrictions relativement fortes, qui ne sont pas respectées par des protocoles comme SSL ou Kerberos. Du point de vue théorique, on peut également borner à priori la taille des messages échangés et le nombre de nonces générés (c.f. [45]). Par une simple énumération des connaissances de l'intrus, le problème de la sécurité de protocoles est alors décidable en temps exponentiel. Malheureusement, le nombre de cas à énumérer rend cette méthode inutilisable en pratique.

Méthodes interactives ou de semi-décision :

Si l'on ne veut pas restreindre le modèle de protocoles cryptographiques, on peut utiliser des techniques qui soit ne terminent pas toujours, soit nécessitent l'intervention de l'utilisateur pour prouver certains lemmes. Par exemple, l'outil d'analyse de protocoles NRL [63] implémente une méthode de preuve de sécurité pour les protocoles cryptographiques sans aucune restriction. En contrepartie de cette généralité, l'utilisateur doit prouver "à la main" certains lemmes assurant ensuite la terminaison d'une exploration symbolique des différents états du protocole. Par contre, si l'on veut une méthode totalement autonome on doit utiliser une procédure de semi-décision comme celle d'Athéna [90] ou de CASRUL [19]. Ces deux méthodes utilisent des explorations symboliques des états du protocole, l'une en arrière du but de sécurité vers l'état initial du protocole (Athéna), l'autre en avant (CASRUL). En tant que méthodes de semi-décision, elles terminent toujours sur des protocoles non sûrs (si une attaque existe alors elle sera trouvée), mais peuvent boucler indéfiniment sur certains protocoles sûrs (donc sans prouver la sécurité du protocole).

Méthodes par approximation :

Les méthodes de semi décision ont l'avantage d'être automatiques, mais elles risquent de ne donner aucun résultat valable dans de nombreux cas. Pour tenter de pallier à ce défaut, et toujours sans restreindre le modèle de protocoles, plusieurs méthodes d'exploration symbolique ont été développées, utilisant cette fois des approximations des ensembles d'états considérés pour assurer la terminaison sur le plus de protocoles possible. Un premier exemple d'approximation est celle présentée par D. Monniaux, T. Genet et F. Klay respectivement dans [73] et [49]. L'idée de ces travaux est de modéliser une approximation supérieure des connaissances de l'intrus à l'aide d'automates d'arbre, puis d'étendre cette modélisation aux états de protocole accessibles grâce à un nombre non borné de sessions parallèles. L'approximation réalisée sur les connaissances de l'intrus peut être assez grossière. Typiquement, si un agent envoie un message contenant deux copies $\langle x, x\rangle$ d'une même connaissance acquise x, l'approximation risque d'utiliser deux messages différents à la place de x, comme $\langle a, b\rangle$ avec $a \neq b$, au lieu de deux messages identiques. Ceci conduit naturellement à de fausse attaques. Cependant, la procédure termine (modulo quelques heuristiques), et elle permet de prouver la sécurité de certains protocoles. Dans le même but, B. Blanchet a proposé dans [11, 12] une méthode basé sur les clauses de Horn (i.e. Prolog). Parmi les approximations réalisées, on trouve une modélisation particulière des nonces et surtout des pas de protocole non ordonnés, c'est à dire qu'une attaque peut utiliser les actions des agents

dans n'importe quel ordre même s'il n'est pas possible en pratique. Ceci termine assez souvent, prouve la sécurité de certains protocoles, mais peut également donner de fausses attaques.

Méthodes de décision restreignant la structure des messages :

Nous avons vu que l'on peut soit avoir une procédure de décision dans un modèle de protocole très restreint (cas fini) ou une procédure de semi décision dans le cas général. Cependant, on peut obtenir des procédures de décision pour des modèles de protocole plus riches que dans le cas fini. Le premier exemple est celui des protocoles ping-pong [42]. En effet, dans ces protocoles les messages échangés sont uniquement construits sur l'encryption à clef atomique non variable. Typiquement, la concaténation et les clefs complexes sont interdites. Cette limitation du modèle de protocoles permet à Dolev et Yao de montrer que la sécurité des protocoles ping-pong, avec un nombre non borné de sessions, est décidable [42] (et même polynomial). Cependant, cette restriction est généralement trop forte pour des protocoles concrets, et d'autres travaux ont par la suite montré la décidabilité du problème pour des classes de protocoles nettement plus riches. Par exemple, V. Cortier a montré dans sa thèse (c.f. [31]) que ce problème est décidable pour un modèle de protocoles sans nonces[1] tel qu'au plus une connaissance acquise soit recopiée à chaque pas de protocole (i.e. à chaque réception/envoi de message par un agent). Ce résultat a l'avantage de pouvoir être étendu aux protocoles exploitant les propriétés algébriques de l'opérateur ou exclusif, mais ne permet pas de décrire des protocoles complexes comme SET. On peut également restreindre autrement le modèle de protocoles cryptographiques pour atteindre une classe décidable. Par exemple, un système de typage fort peut à lui seul garantir la décidabilité du problème (c.f. [59]), même avec un nombre infini de nonces. Dans la même ligne, un schéma de marquage sur les protocoles permettant d'interdire l'unification de sous messages chiffrés différents rend le problème décidable même avec un nombre infini de nonces (c.f. [79]).

Méthodes de décision à nombre de sessions borné :

Ici, nous nous intéressons essentiellement à un autre type de restriction sur les protocoles cryptographiques permettant de décider leur sécurité. Il s'agit des protocoles à nombre de sessions borné. Par contre, nous pouvons considérer un modèle de protocoles où les messages transmis sont de taille non bornée, sont non typés, etc... Différentes méthodes de décision du problème de l'insécurité ont été développées pour ce modèle de protocoles. Par exemple, une première méthode a été proposée par A. Huima dans [53], basée sur la réécriture. Par la suite, R. Amadio *et al.* ont présenté dans [4, 6] une méthode de décision basée sur des algèbres de processus (similaire aux spi-calcul), pour des protocoles à clefs atomiques. Les états du protocole sont représentés de manière symbolique par des termes représentant les connaissances de l'intrus et des contraintes sur les messages envoyés par l'intrus aux agents honnêtes. Toujours dans le cas d'un nombre borné de sessions, Y. Chevalier et L. Vigneron proposent dans [25] une procédure de décision cette fois pour des clefs quelconques (n'importe quel message composé peut être une clef). Dans cette thèse, nous étudierons notamment la complexité du problème de la sécurité de protocoles cryptographiques dans un modèle équivalent (i.e. nombre de sessions borné, messages et clefs quelconques). Plus récemment, M. Boreale [16] présente pour une variante du spi-calcul une procédure de décision basée sur une représentation symbolique des traces du protocole. Partant d'une trace issue de la spécification du protocole, celle-ci est raffinée pour décomposer les connaissances de l'intrus en éléments irréductibles, choisir (de manière non déterministe) la manière dont il crée les messages envoyés aux principaux, etc... Cependant, cette méthode ne travaille que sur des clefs atomiques, et aucune complexité n'est donnée.

Méthodes avec opérateurs algébriques :

[1] Un nombre borné de nonces est ici équivalent à aucun nonce : on peut les remplacer par des atomes.

Les méthodes ci-dessus semblent satisfaisantes pour les protocoles cryptographiques sans opérateurs algébriques. Pourtant, de nombreux protocoles utilisent de tels opérateurs, dont on ne peut pas ignorer les propriétés. Dans ce travail, nous montrerons entre autres que le problème de décision de la sécurité de protocoles cryptographiques à nombre de sessions borné est co-NP-complet[2], que l'on utilise (ou pas) l'opérateur de ou exclusif, l'opérateur d'exponentiation (prop. de Diffie-Hellman), ou un opérateur d'encryption commutative. Pour les protocoles avec ou exclusif, H. Comon et V. Shmatikov présentent dans [34] une procédure de décision DEXPTIME pour un modèle un peu plus général que le nôtre, mais dont les protocoles permis en plus des nôtres ne peuvent à priori pas être réalisés en pratique. Pour les protocoles avec exponentiation, il existe plusieurs tentatives infructueuses. Par exemple, dans [69] J.Millen et V.Shmatikov proposent une réduction de ce problème à une résolution de systèmes d'équations quadratiques. Malheureusement, résoudre ces systèmes est en général indécidable. Un problème similaire a également été étudié par M.Boreale et M.G. Buscemi dans [17]. Ils proposent une procédure correcte et complète par rapport à une sémantique opérationnelle, dont il n'est malheureusement pas clair qu'elle prenne en compte toutes le propriétés de Diffie-Hellman de l'exponentiation. De plus, la méthode nécessite une borne à priori sur le nombre de facteurs des produits (dans les exposants), et les auteurs ne montrent pas qu'elle existe.

Les travaux présentés ci-dessus prennent en compte les propriétés de certains opérateurs algébriques. Cependant, vérifier un protocole face à une modélisation des propriétés algébriques de ses opérateurs, fût-elle relativement complète, ne suffit naturellement pas. Il faut de plus assurer que la modélisation est correcte pour assurer la sécurité d'un protocole concret. Dans cette voie, M. Backes, B. Pfitzmann, M. Waidner ont récemment présenté un travail visant à combiner d'une part les méthodes formelles d'analyse de protocoles et d'autre part les méthodes de cryptanalyse étudiant les propriétés de chaque opérateur [10]. Le but est d'établir une équivalence entre la sécurité d'un protocole concret et la sécurité de sa modélisation formelle. En particulier, ce travail a conduit à une preuve formelle et cryptographique de la sécurité du protocole à clefs publiques de Needham-Schroeder-Lowe [9].

1.3 Contenu de la thèse et plan

Dans cette thèse, nous nous intéressons au problème de décision de l'insécurité des protocoles cryptographiques pour des modèles d'intrus plus réalistes que le modèle d'intrus initialement proposé par Dolev-Yao. Le problème est d'autant plus complexe que l'on veut donner à l'intrus tous les moyens dont il pourrait disposer en pratique dans le pire des cas, c'est à dire que l'on va compléter le modèle d'intrus de Dolev-Yao de manière à relâcher certaines limitations importantes de l'hypothèse du chiffrement parfait. Nous équiperons ainsi l'intrus avec différentes règles de déduction algébriques (xor, exponentielle, etc...) ou non-algébriques (règles préfixe). Détaillons les différentes parties de ce travail :

Formalisation des protocoles, chapitre 2 :

Nous commencerons par définir le modèle de protocoles cryptographiques utilisé dans la plus grande partie de cette thèse. Pour lier ce modèle aux spécifications de protocoles que l'on trouve habituellement dans la littérature, nous commencerons par définir un modèle de référence, dit modèle Alice-Bob, et préciserons ses lacunes. Nous définirons alors le modèle utilisé ici, dit modèle par rôles, tout d'abord sans opérateurs algébriques puis étendu avec de tels opérateurs.

[2] autrement dit, le problème de l'insécurité de protocoles est NP-complet.

Nous en profiterons pour poser toutes les définitions de base utilisées dans la suite de la thèse. Enfin, nous examinerons deux extensions très utiles en pratique de ce modèle, l'une augmentant les capacités de déduction des principaux, et l'autre modélisant des canaux de communication différents.

Vérification avec ou sans opérateurs algébriques, chapitres 3, 4, et 5 :

Nous présenterons alors différents résultats de décidabilité, et de complexité, du problème de l'insécurité de protocoles cryptographiques à nombre de sessions borné. En particulier, toutes les méthodes présentées ici utilisent des clefs symétriques non atomiques. Pour tous les cas envisagés, nous donnons à la fois une procédure de décision et nous prouvons qu'elle a une complexité optimale. Ce travail fait l'objet de publications en conférences, à la fois pour le cas de base [82] (plus journal [83]), pour le xor [22], et pour l'exponentielle [24]. Les travaux publiés avant [82] ne traitaient pas les clefs non atomiques, excepté Y. Chevalier et L. Vigneron dans [25], et aucun n'a considéré la complexité du problème. Plus précisément, nous montrerons que le problème de l'insécurité de protocoles cryptographiques est NP-complet (et donc décidable) pour des protocoles utilisant (ou pas) un opérateur de ou exclusif, un opérateur d'exponentiation, ou un opérateur d'encryption commutative. Dans tous les cas, nous donnerons à l'intrus les moyens d'utiliser efficacement ces nouveaux opérateurs et leurs propriétés algébriques.

Nous commencerons par détailler le cas des protocoles sans opérateurs algébriques. Ceci permet de présenter la structure générale de preuve utilisée avant d'ajouter les nombreux problèmes inhérents à l'ajout d'opérateurs algébriques. En outre, nous présentons différents codages de problèmes NP-complets dans l'insécurité de protocoles cryptographiques. Ceci permet d'explorer les sources de NP-complétude de ce problème. Nous présentons notamment un codage original de 3-SAT dans une classe de protocoles sans concaténation, sans clef composée, à variables de type atomique, et avec un ordre d'exécution fixé (donc exécution déterministe).

Nous présenterons ensuite deux extensions du résultat précédent de décidabilité et de complexité du problème de l'insécurité de protocoles cryptographiques à nombre de sessions borné, le premier avec l'opérateur "ou" exclusif, le second avec l'opérateur d'exponentiation. Dans les deux cas, nous montrerons en fait un résultat plus général de décidabilité et de complexité avec un intrus décrit pas des règles dites d'oracle. On peut alors prouver d'un côté que l'insécurité est NP-complète face à des règles d'oracle, et d'un autre côté que les règles d'intrus pour le xor ou l'exponentielle que l'on veut ajouter sont bien des règles d'oracle. Ce schéma de preuve est assez général, et il peut être réutilisé pour d'autres résultats (comme la combinaison de modèles du chapitre 7).

Vérification avec encryption commutative, chapitre 6 :

Dans le travail précédent sur l'exponentielle, l'intrus peut inverser tout terme connu (pour l'opérateur produit), avant de l'utiliser comme exposant. Le cas où cette inversion n'est pas possible sera également considéré. Par analogie avec l'algorithme RSA, nous parlerons alors d'encryption commutative. Ce point peut changer sensiblement le comportement de l'intrus, mais nous montrerons que l'on peut tout de même adapter les preuves du cas avec exponentiation au cas avec encryption commutative.

En outre, nous montrerons que le problème de l'insécurité de protocoles est aussi NP-complet dans le cas des protocoles ping-pong avec encryption commutative, donc sans concaténation et avec des clefs atomiques fixes, mais avec un nombre non borné de sessions.

Combinaison de modèles, chapitre 7 :

Nous étudierons ensuite la possibilité de combiner deux modèles de protocoles : le modèle de protocoles à nombre fini de sessions mais avec des tailles de messages non bornées, et un modèle de protocoles à nombre infini de sessions mais avec des tailles de messages bornées (c.f. [45]). Cette modélisation est intéressante dans la mesure où l'on a les avantages des deux modèles. D'un côté, le modèle à sessions infinies permet de ne pas spécifier à l'avance le nombre de sessions à vérifier (mais au prix de messages plus simples), et d'un autre côté le modèle à sessions bornées permet des structures de messages plus précises. Pour combiner ces deux modèles, l'idée sera d'étendre les capacités de l'intrus dans le modèle par rôles (sessions bornées) de manière à modéliser un nombre infini de sessions à messages bornés. Nous montrerons que le problème de l'insécurité de protocoles cryptographique dans ce modèle "combiné" est dans DEXPTIME.

Outil de vérification, chapitre 8 :

Enfin, nous présenterons un outil de vérification de protocoles (Atsé) développé dans le cadre du projet européen AVISS. Dans le cas des protocoles cryptographiques à nombre fini de sessions et sans opérateur algébrique, cet outil est correct et complet. De plus, il est actuellement étendu pour supporter l'opérateur ou exclusif et ses propriétés. A terme, il supportera également l'opérateur d'exponentiation.

2

Formalisation des protocoles cryptographiques

Sommaire

Informellement, un protocole cryptographique spécifie les échanges de messages entre un ensemble d'agents, honnêtes ou non. Le but de ce chapitre est de définir formellement les éléments de base que nous utiliserons. Il s'agira essentiellement des primitives cryptographiques et des

protocoles cryptographiques, ainsi que des intrus et des notions d'attaques. On fixe ainsi le cadre dans lequel on va spécifier et vérifier des protocoles cryptographiques.

Nous commencerons par préciser le genre de protocoles cryptographiques que nous voulons étudier au travers de la notion d'environnement hostile. Ceci nous conduira naturellement à deux modèles de protocoles cryptographiques, à savoir le modèle Alice-Bob très utilisé dans la littérature mais dont l'interprétation peut être relativement floue, et le modèle par rôles, équivalent mais beaucoup plus précis, que nous utiliserons tout au long de ce travail. Certaines restrictions de ces modèles seront considérées à la section 2.5. Ces deux modèles de protocoles cryptographiques supposent que les primitives cryptographiques utilisées par les principaux sont parfaites, c'est à dire qu'elles ne permettent aucune autre action que l'encryption et la décryptage d'un message quand la clef nécessaire est connue. Bien que cette propriété soit techniquement fausse en théorie (par exemple, on peut en principe calculer une clef privée de chiffrement à partir de la clef publique correspondante), les autres propriétés des primitives cryptographiques sont souvent considérées comme inutilisables en pratiques à partir du moment où les algorithmes de chiffrement sont utilisés correctement (typiquement, avec des clefs de taille suffisante). Cependant, certains protocoles (souvent des protocoles de groupe) exploitent certaines propriétés algébriques d'autres opérateurs que les opérateurs de chiffrement. Il peut s'agir par exemple de l'opérateur de ou exclusif entre deux messages (bit à bit), de l'exponentiation d'un message par un autre, ou d'un opérateur d'encryption à clefs privées/publiques commutatives. La vérification de ce genre de protocoles nécessite donc de prendre en compte les propriétés algébriques de ces opérateurs, et de permettre à l'intrus d'en faire usage. Ainsi, la section 2.4 présentera formellement ces primitives cryptographiques avancées ainsi que des propriétés algébriques très intéressantes de ces opérateurs pour la recherche de nouvelles attaques.

2.1 Environnement hostile.

Tous les intrus que nous allons définir dans ce chapitre seront des extensions de l'intrus de Dolev-Yao (c.f. [43]), c'est à dire qu'ils en posséderons tous au moins les capacités. L'avantage de la notion d'intrus de Dolev-Yao est qu'elle fournit à l'intrus tous les moyens nécessaires pour intervenir efficacement dans un protocole. En effet, pour n'importe quelle transmission de donnée (ou envoi de message) entre deux principaux, l'intrus de Dolev-Yao est capable de stopper le message (pour qu'il n'atteigne pas son destinataire) et de le lire (pour acquérir des connaissances supplémentaires). De plus, il est capable d'analyser en détail tous les messages qu'il a interceptés (par exemple décrypter un message avec une clef obtenue à un autre moment), de créer n'importe quel message à l'aide de ses connaissances acquises ou déduites, et enfin de l'envoyer à n'importe quel principal sous n'importe quelle identité (pour se faire passer pour un agent honnête). Dans ces conditions, il devient inutile de séparer les notions de média de communication (le réseau physique servant à transporter les messages, par exemple Internet), et n'importe quel intrus construit sur le modèle de Dolev-Yao. De plus, avec des intrus aussi performants, attaquer un protocole à l'aide d'un seul intrus ou de plusieurs ayant des capacités similaires ne fait aucune différence, puisqu'ils pourront tous intercepter les mêmes messages, construire les mêmes messages, communiquer avec les mêmes principaux, et partager leurs connaissances en s'envoyant mutuellement des messages. Ainsi, on considère qu'un intrus unique a réussi à prendre le contrôle total du média de communication. En conséquence, on considère que tous les principaux communiquent en réalité uniquement avec l'intrus : tous les messages émis sont ajoutés aux connaissances de l'intrus, et seul ce dernier construit et envoie des messages aux principaux.

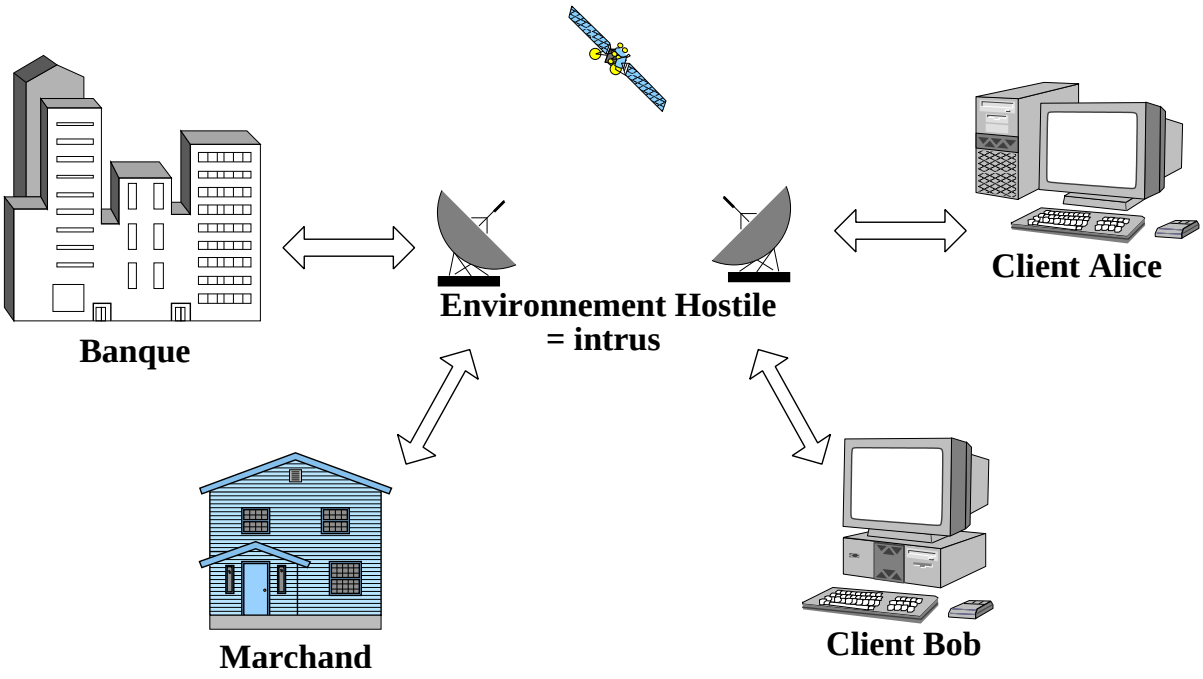


FIG. 2.1: Organisation générale d'un protocole.

2.2 Modèle Alice-Bob de référence

Dans la littérature, on trouve souvent les protocoles cryptographiques écrits dans une syntaxe simple et intuitive, bien qu'assez imprécise. Par exemple, J.Clark et J.Jacob en utilisent une de manière intensive dans leur état de l'art sur les protocoles d'authentification [28]. Elle est commune à la plupart des publications sur les protocoles de sécurité. Nous allons donc décrire le modèle de protocole sous-jacent à cette syntaxe. Ce sera notre modèle de référence, ou modèle Alice-Bob pour rappeler sa structure.

2.2.1 Syntaxe des protocoles

La syntaxe du modèle Alice-Bob est très simple. En effet, elle ne décrit pas précisément toutes les actions des agents, mais se contente de donner une trace d'exécution du protocole dans le cas où aucun intrus ne vient perturber le bon déroulement des échanges de messages. Étant donné un ensemble de noms d'agents $Agents$, de clefs publiques/privées $ClefsAsym$, de nonces $Nonces$, et d'atomes $Atomes$ (incluant les clefs symétriques atomiques), tels que $Agents \subseteq Atomes$ et $\forall k \in ClefsAsym$, $k^* \in ClefsAsym$ est la clef inverse de k, un message envoyé (ou reçu) par un agent est construit selon la grammaire suivante :

$$\begin{aligned} MessageAB \ ::= \quad & Agents \,|\, ClefsAsym \,|\, Nonces \,|\, Atomes \\ & |\, \langle MessageAB, MessageAB \rangle \,|\, \{MessageAB\}_{MessageAB} \end{aligned}$$

où $\langle .., .. \rangle$ est l'opérateur de création d'une paire (ou concaténation), et $\{..\}_{..}$ est l'opérateur d'encryption. Pour simplifier la syntaxe, on notera $a, b, ..z$ le message $\langle a, \langle b, \langle ..., z \rangle \rangle \rangle$. On remarque immédiatement que l'on ne fait aucune différence syntaxique entre les différents types d'encryption (symétrique et asymétrique). En fait, ces deux types d'encryption sont identifiés par la clef

utilisée : si $K \in ClefsAsym$, alors $\{..\}_K$ est une encryption privée/publique dont la clef inverse est K^*, et si $b \in MessageAB \backslash ClefsAsym$ alors $\{..\}_b$ est une encryption symétrique de clef b. A partir de là, une session de protocole dans le modèle Alice-Bob est une suite d'échanges de messages de la forme :

$$A \rightarrow B \ : \quad M$$

avec $A, B \in Agents$ et $M \in MessageAB$. Un tel échange représente l'envoi du message M par l'agent A à l'agent B. Un protocole sera un ensemble de connaissances initiales pour chaque agent (i.e. une liste de messages connus dès le début) et un ensemble de sessions différentes exécutées en parallèle (ou d'instances de session selon [19]) et répétées séquentiellement. Le nombre de répétitions peut être soit non borné, soit fixé par le protocole lui même. Ici, on ne considérera toujours qu'un nombre fini de sessions et de répétitions. Cette syntaxe ne décrit pas comment l'agent A va construire le message M, ni quelles parties du message M l'agent B pourra reconnaître ou calculer. Généralement, on suppose qu'un agent honnête recevant un message effectuera le maximum de vérifications possibles, en fonction de ses connaissances accumulées et d'un système de déduction de connaissances (i.e. un ensemble de règles décrivant les opérations d'encryption, de décryptage, etc... qu'un agent peut réaliser). Dans ce modèle, les nonces ont un rôle assez particulier. A la différence des autres atomes ou clefs asymétriques immuables, les nonces sont générés à chaque (nouvelle) session, et sont donc supposés uniques pour chaque session et chaque itération de session. Enfin, le but de l'intrus est d'obtenir un atome particulier, $Secret \in Atomes$, ou de manière duale le but du protocole est de conserver secret cet atome même s'il est transmis d'un principal à un autre.

2.2.2 Capacités de l'intrus

L'intrus possède deux type de capacités. D'une part, comme décrit à la section 1.1.3, l'intrus est capable d'intercepter tous les messages transmis par les principaux. Ceci est intrinsèquement lié au modèle par rôle que nous décrirons à la section 2.3.1, et sera formellement décrit avec la notion d'attaque à la section 2.3.5. Ainsi, le modèle Alice-Bob ne sert qu'à décrire le fonctionnement normal du protocole, et pour décrire des attaques on a besoin d'un modèle plus précis tel que le modèle par rôles. D'autre part, l'intrus, comme les principaux, est capable d'effectuer des déductions de connaissances sur les messages interceptés. Pour les primitives cryptographiques de base, il s'agit de la fabrication d'une paire $\langle a, b \rangle$ connaissant a et b, d'une encryption symétrique $\{a\}_b$ connaissant a et $b \notin ClefsAsym$ ou d'une encryption asymétrique $\{a\}_k$ connaissant a et $k \in ClefsAsym$, ainsi que de la décomposition d'une paire $\langle a, b \rangle$ pour calculer a et b, du décryptage d'une encryption symétrique $\{a\}_b$ connaissant b pour calculer a, ou du décryptage d'une encryption asymétrique $\{a\}_k$ (respectivement $\{a\}_{k^*}$) connaissant k^* (resp. k) pour calculer a.

2.2.3 Limitations du modèle

Le principal problème de ce modèle est qu'il ne décrit pas assez précisément des actions des principaux, et en particulier les parties des messages reçus dont ils peuvent vérifier la structure, notamment parce que cela dépend d'une part du modèle de déductions de connaissances utilisé et d'autre part de l'implémentation (paranoïaque ou laxiste) du protocole. Par exemple, considérons le protocole suivant :

$$\begin{array}{ll} Alice \rightarrow Bernard : & \{\{a\}_b\}_K \\ Bernard \rightarrow Alice : & \{Secret\}_{\{a\}_b} \end{array}$$

où le principal *Bernard* connaît $a, b \in Atomes$ et $K \in ClefsAsym$ dès le départ, mais l'intrus ne connaît que K. On a alors deux interprétations possibles : soit *Bernard*, après avoir calculé $\{a\}_b$ à partir de $\{\{a\}_b\}_K$ vérifie qu'il s'agit bien de l'encryption de a par b avant d'encrypter le secret avec $\{a\}_b$, auquel cas il n'y a pas d'attaques puisque l'intrus ne peut pas connaître $\{a\}_b$, soit *Bernard* ne vérifie pas ça et encrypte le secret directement avec $\{a\}_b$ qu'il vient de calculer, auquel cas l'intrus peut fabriquer n'importe quel terme $\{I\}_K$ avec I connu quelconque (même une encryption pour le cas typé) et récupérer le secret grâce à $\{Secret\}_I$. Pour cette raison, le modèle par rôle de la section suivante différenciera les messages émis et reçus par les principaux en utilisant des variables pour préciser les parties d'un message reçu qu'un principal ne calcule pas (soit parce qu'il ne peut pas le faire, soit parce qu'il choisit de ne pas le faire).

2.2.4 Exemple.

Pour illustrer le modèle de protocoles Alice-Bob, nous allons décrire le protocole d'échange de clefs Otway-Rees (c.f. [88] page 64). Le but de ce protocole est de générer une clef K_{AB} utilisée pour toutes les communications futures entre les principaux A et B. Cette clef sera générée par un tiers de confiance, à savoir un serveur S, avec lequel A et B partagent déjà des clefs K_{AS} et K_{BS} respectivement. Voici une session de ce protocole :

$$\begin{array}{ll} A \to B : & N,\, A,\, B,\, \{N_A,\, N,\, A,\, B\}_{K_{AS}} \\ B \to S : & N,\, A,\, B,\, \{N_A,\, N,\, A,\, B\}_{K_{AS}},\, \{N_B,\, N,\, A,\, B\}_{K_{BS}} \\ S \to B : & N,\, \{N_A,\, K_{AB}\}_{K_{AS}},\, \{N_B,\, K_{AB}\}_{K_{BS}} \\ B \to A : & N,\, \{N_A,\, K_{AB}\}_{K_{AS}} \\ A \to B : & \{Secret\}_{K_{AB}} \end{array}$$

avec N un numéro de session, N_A et N_B des nonces générés par A et B, et K_{AS}, K_{BS} des clefs symétriques (atomiques) . Ici, chaque principal A et B crée une demande de clef temporaire pour S, à savoir $\{N_A,\, N,\, A,\, B\}_{K_{AS}}$ et $\{N_B,\, N,\, A,\, B\}_{K_{BS}}$, et le serveur crée deux messages $\{N_A,\, K_{AB}\}_{K_{AS}}$ et $\{N_B,\, K_{AB}\}_{K_{BS}}$ pour envoyer la clef K_{AB} à A et B. A la fin, les deux principaux A et B devraient partager une clef secrète K_{AB}, et A peut alors envoyer *Secret* à B encrypté par K_{AB}. Une attaque possible sur ce protocole utilise le fait que les connaissances acquises par les principaux ne sont pas typées : le principal A recevant le message $\{N_A,\, K_{AB}\}_{K_{AS}}$ vérifiera la présence de son nonce N_A mais ne peut pas vérifier que ce qu'il interprète comme étant K_{AB} est bien une clef symétrique et non un message construit de toutes pièces par l'intrus. En l'occurrence, l'intrus peut envoyer le message $\{N_A,\, N,\, A,\, B\}_{K_{AS}}$ au principal A à la place de $\{N_A,\, K_{AB}\}_{K_{AS}}$, et le principal A identifiera $\langle N,\, A,\, B\rangle$ comme étant la clef K_{AB}. Du coup, l'intrus peut intercepter le message $\{Secret\}_{\langle N,\, A,\, B\rangle}$ envoyé par A, et calculer le secret puisqu'il connaît N, A, et B.

2.3 Modèle par rôles

2.3.1 Spécification

De nombreux modèles de spécification de protocoles cryptographiques se rencontrent dans la littérature. En particulier, chaque manière de résoudre le problème de l'insécurité des protocoles cryptographiques a tendance à introduire une nouvelle manière de modéliser les protocoles cryptographiques adaptée à la méthode.

Ici en revanche, nous n'avons pas besoin d'introduire un modèle de protocoles fondamentalement différent des modèles classiques précédents. En particulier, le modèle présenté dans

cette section est basé sur une notation directe, rôle par rôle, des échanges de messages effectués par chaque principal (ou agent honnête), dans le style de [20]. Cela nous permet d'utiliser un modèle classique, universellement reconnu, tout en adaptant et clarifiant sa notation pour éviter toute traduction ou interprétation inutile. Les actions des différents principaux seront modélisées par une liste partiellement ordonnée de pas de protocole, représentant les différents choix possibles effectués par les principaux lors de l'exécution du protocole. Chaque pas de protocole, représentant une action élémentaire d'un principal, associera à un message reçu la réponse correspondante émise par le principal. L'ensemble des messages acceptés par un principal à un pas de protocole donné sera spécifié par un filtre, i.e. les messages attendus comportent des variables représentant les connaissances nouvellement ou précédemment acquises par les principaux. Par comparaison avec le modèle de référence de la section précédente, cela revient à :

- Regrouper les actions effectuées par chaque principal au sein d'un rôle.
- Définir un ensemble de variables représentant toutes les connaissances acquises par les principaux lors de l'exécution du protocole. Comme on n'utilise qu'un nombre fini de sessions, que ce soit en parallèle (un rôle joué plusieurs fois en même temps) ou en séquence (un rôle rejoué différemment plusieurs fois les unes après les autres), et que l'on a un nombre fini de rôles, on n'a besoin que d'un nombre fini de variables que l'on peut choisir et fixer dans la spécification du protocole. On supposera que les domaines des variables sont globaux, i.e. toutes les variables sont (potentiellement) utilisables dans n'importe quel rôle, par opposition à des variables locales à chaque rôle. Comme les variables représentent les connaissances des principaux, utiliser une variable dans la spécification d'un principal différent de celui qui l'a reçue permet de modéliser une connaissance partagée entre deux principaux, sans intervention possible de l'intrus.

Cette approche est assez naturelle car elle décrit clairement, pas à pas, les actions des principaux sans entrer dans le détail des opérations réellement effectuées par un programme implémentant le protocole. De plus, des outils comme CAPSL [40] ou CASRUL [54] ont été créés pour traduire automatiquement des spécifications de protocoles du modèle de référence (le plus couramment utilisé dans la littérature) vers une spécification syntaxiquement équivalente à celle-ci. Nous pouvons maintenant décrire formellement ce modèle de protocoles cryptographiques.

Nous rappelons les notions de terme et de terme clos. (c.f. HandBook of Theoretical Computer Science, chap. Logic Programming, par K.R. Apt). Tous les symboles de fonctions auront une arité fixe (2) et un ordre sur les arguments significatif. Cependant, certains symboles particuliers comme $\oplus$ ou $\cdot$ seront écrits (et lus) modulo l'associativité et la commutativité de ces deux opérateurs, ce qui revient (de manière équivalente) à étendre la notion de termes pour avoir deux symboles de fonction $\oplus$ et $\cdot$ d'arité non bornée et sans ordre sur leurs arguments (représentés par un ensemble ou un multiensemble au lieu d'une liste). Par mesure de simplicité, les notations n-aires $\oplus$ et $\cdot$ ne seront que des raccourcis syntaxiques pour deux opérateurs associatifs-commutatifs. Comme dans le modèle Alice-Bob, on suppose l'existence de certains atomes :

$$\begin{array}{rcl} Noms & : & \text{Noms des agents, Nonces, Clefs symétriques, etc...} \\ Clefs & : & \text{Clefs publiques et privées, telles que } k \in Clefs \text{ ssi } k^* \in Clefs \\ Atomes & : & Noms \cup Clefs \text{ i.e. l'ensemble des atomes du protocole} \end{array}$$

L'ensemble $Noms$ contient tous les atomes, et $Clefs$ contient un ensemble d'atomes, avec $Clefs \cap Noms = \emptyset$, tel qu'il existe une fonction involutive de $Clefs$ dans $Clefs$, qui à tout $k \in Clefs$ associe son inverse $k^* \in Clefs$. Ainsi, pour toute clef K, K^* désigne l'inverse de K,

même si $K = P^*$ (et donc $K^* = P^{**} = P$)[3]. L'ensemble *Atomes* est fini. En particulier, on n'a qu'un nombre fini de nonces. Ceci n'est en rien restrictif. En effet, comme on ne considère qu'un nombre fini de sessions (en parallèle ou en séquence), le nombre total de nouvelles connaissances générées par tous les principaux lors d'une exécution complète du protocole est nécessairement fini, borné lors de la spécification du protocole. Lors de la spécification d'un protocole, on pourra ainsi indifféremment n'utiliser un nonce que dans le rôle l'ayant créé, ou dans plusieurs rôles en même temps pour modéliser une connaissance initiale partagée par plusieurs principaux. Ceci nous permet de considérer les nonces comme des connaissances initiales de principaux, i.e. des atomes dans la spécification, et non comme des connaissances réellement générées lors de l'exécution du protocole. A ces atomes, on ajoute un ensemble de variables à domaine global :

$$Var \quad : \quad \text{Variables du protocole, i.e. connaissances des principaux.}$$

Les variables représentent, du point de vue des principaux, les sous-termes des messages reçus qu'un principal n'a pas pu décomposer. Nous verrons à la section 2.5.1 une manière d'étendre un peu cette notion de variables sans perdre les résultats de complexité des chapitres 3 à 6.

En plus des atomes et des variables, les messages échangés lors de l'exécution du protocole sont construits grâce aux opérateurs habituels : la concaténation (ou création d'une paire), notée $\langle \_, \_ \rangle$, et les opérateurs d'encryption symétrique $\{\_\}^s_\_$ et publique/privé $\{\_\}^p_\_$. On distingue ainsi les deux types d'opérateurs à l'aide d'un exposant ($\_^s$) ou ($\_^p$). A la différence du modèle Alice-Bob de référence, cela nous permet de ne pas dépendre du type de clef utilisée pour déterminer l'opérateur d'encryption, et donc d'éviter toute confusion. Ainsi, n'importe quel terme, qu'il soit dans *Noms*, dans *Clefs*, ou même composé, pourra être utilisé comme clef symétrique pour l'opérateur $\{\_\}^s_\_$. En contre-partie, on limite l'utilisation de l'encryption publique/privé aux seules clefs dans $Clefs$. Cela nous donne les grammaires suivantes définissant les règles de formation des messages et des termes :

$$\begin{array}{rcl} Messages & ::= & Atomes \,|\, \langle Messages, Messages \rangle \,|\, \{Messages\}^s_{Messages} \,|\, \{Messages\}^p_{Clefs} \\ Terme & ::= & Var \,|\, Atomes \,|\, \langle Terme, Terme \rangle \,|\, \{Terme\}^s_{Terme} \,|\, \{Terme\}^p_{Clefs} \end{array}$$

Tout message est également un terme, et tout terme sans variable est également un message, aussi appelé terme clos. Pour ne pas surcharger les notations, on notera par $m_1, ..., m_n$ l'ensemble de messages ou de termes $\{m_1, .., m_n\}$. Par extension, on notera E_1, E_2 l'union $E_1 \cup E_2$ de deux ensembles de messages ou de termes E_1 et E_2. De même, pour un ensemble de messages ou de termes M et un message ou un terme t, on notera E, t l'ensemble $E \cup \{t\}$. On notera également $Var(t)$ l'ensemble des variables apparaissant dans un terme t. La notion de sous-termes est classique, et décompose toutes les parties d'un terme :

Définition 2.3.1.1 *Sous-termes.*

Pour tout terme t, on note $STermes(t)$ l'ensemble des sous-termes de t défini récursivement par :

- $STermes(t) = \{t, t^*\}$ *si* $t \in Clefs$
- $STermes(t) = t$ *si* $t \in Atomes \backslash Clefs$ *ou* $t \in Var$.
- $STermes(t) = \{t\} \cup STermes(u) \cup STermes(v)$ *si* $t = \langle u, v \rangle$, $t = \{u\}^s_v$, *ou* $t = \{u\}^p_v$ *avec* $\{u, v\} \subset Terme$.

[3]Le choix du symbole $..^*$ n'est pas complètement aléatoire : le symbole $'$ sera utilisé pour décrire des variantes de différents atomes, et $..^{-1}$ décrira l'inverse dans un groupe multiplicatif.

Ceci s'étend naturellement aux ensembles de termes : $STermes(E) = \bigcup_{t \in E} STermes(t)$ pour tout ensemble de termes E. Nous représentons les termes de manière optimisée comme des graphes acycliques dirigés (DAG) :

Définition 2.3.1.2 *Représentation* DAG *d'un ensemble de termes.*

Soit E un ensemble de termes. La représentation DAG de E est un graphe étiqueté $(\mathcal{V}, \mathcal{E})$ tel que :

- *L'ensemble des noeuds $\mathcal{V}$ est l'ensemble $STermes(E)$ des sous-termes de E.*
- *L'ensemble des arêtes étiquetées est :*

$$\left\{ u \overset{gauche}{\longrightarrow} a \text{ ou } u \overset{droite}{\longrightarrow} b \,|\, u \in \mathcal{V}, \text{ et } u = \langle a, b \rangle \text{ ou } u = \{a\}_b^s \text{ ou } u = \{a\}_b^p \right\}$$

On remarque aisément que la représentation DAG est unique. De plus, on constate aisément que si n est le nombre d'éléments de $STermes(E)$, noté $\#STermes(E)$, alors la représentation DAG de E a au plus n noeuds et $2n$ arêtes, et donc de taille linéaire en n. On définit donc, pour simplifier :

Définition 2.3.1.3 *Taille* DAG d'un ensemble de termes.

Soit E un ensemble de termes. La taille DAG *de E, notée $|E|_{dag}$, est le nombre de sous-termes distincts de E, i.e. $\#STermes(E)$.*

Le coefficient constant entre la taille DAG et la taille de la représentation DAG n'a aucune importance pour les résultats de complexité de ce travail. Par abus de notation, on notera $|t|_{dag} = |\{t\}|_{dag}$ pour tout terme t. De manière à dénoter aisément les connaissances des principaux et les messages envoyés lors d'une exécution d'un protocole, on utilise des substitutions :

Définition 2.3.1.4 *Substitutions.*

Une substitution est une fonction de Var dans $Terme$, notée $[x \leftarrow \sigma(x), ..., z \leftarrow \sigma(z)]$ pour $\{x, .., z\} = Dom(\sigma) \subset Var$, $\sigma(x) \in Terme$ pour toute $x \in Dom(\sigma)$, et $\sigma(x) = x$ pour toute $x \in Var \backslash Dom(\sigma)$.

Une substitution close σ est une substitution telle que $\sigma(x) \in Messages$ pour tout $x \in Var$ (i.e. $Dom(\sigma) = Var$ dans ce cas).

On dénote l'application d'une substitution σ à un terme t par $\sigma(t)$ ou $t\sigma$. Cela revient à remplacer chaque variable x de t par sa valeur $\sigma(x)$. Par exemple, on a $\langle x, y \rangle [x \leftarrow u] = \langle u, y \rangle$.

Même si selon cette définition une substitution peut être définie de manière récursive (par exemple $\sigma(x) = y$ et $\sigma(y) = \langle x, y \rangle$), toutes les substitutions considérées par la suite seront supposées idempotentes, i.e. $t\sigma = (t\sigma)\sigma$ pour tout terme t, et donc non récursives. En particulier, une substitution close étant une élimination complète des variables, son application à un terme est nécessairement idempotente. On étend logiquement ces notations à des ensembles de termes. Par exemple, $E\sigma$ est l'ensemble $\{t\sigma \,|\, t \in E\}$. De même, on étend la taille DAG aux substitutions :

$$|[x \leftarrow \sigma(x), .., z \leftarrow \sigma(z)]|_{dag} = |\sigma(x)|_{dag} + .. + |\sigma(z)|_{dag}$$

On peut à présent décrire formellement le modèle par rôles de protocoles cryptographiques :

Définition 2.3.1.5 *Pas de Protocole.*

Un pas de protocole est une paire de termes $R, S \in Terme$, noté $R \Rightarrow S$.

L'exécution d'un pas de protocoles $R \Rightarrow S$ par un principal A dont les connaissances sont décrites par une substitution σ consiste en deux étapes :

- Réception : Le principal reçoit de la part de l'intrus (ou d'un autre principal par l'intermédiaire de l'intrus) un message t tel qu'il existe une substitution σ' vérifiant $t = R\sigma'$, et compatible avec σ : pour tout $x \in Var$, $\sigma'(x) = \sigma'(\sigma(x))$. Le calcul de σ' est unique. De cette manière, σ' décrit les connaissances acquises par le principal à la réception de t, plus ses connaissances précédentes.
- Envoi : Le principal met à jour ses connaissances, qui deviennent σ', et envoie le message $S\sigma'$ à l'intrus (ou à un autre principal par l'intermédiaire de l'intrus).

Les seules connaissances significatives des principaux sont les valeurs des variables. Vérifier qu'un protocole est correctement spécifié, ou créer une spécification de protocole satisfaisante (au sens habituel, i.e. exécutable en pratique) est laissé à des outils externes, comme par exemple HLPSL2IF du projet Européen AVISS[7]. Avec plusieurs pas de protocoles, on peut définir un protocole :

Définition 2.3.1.6 *Protocole.*

Un protocole est un triplet $(\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}\}, <_\mathcal{I}, S)$ *où* S *est un ensemble fini de messages,* $<_\mathcal{I}$ *est un ordre partiel sur l'ensemble fini* $\mathcal{I}$*, et pour tout* $\iota \in \mathcal{I}$*,* $R_\iota \Rightarrow S_\iota$ *est un pas de protocole tel que :*

$$(C_1) : \text{ Pour tout } \iota \in \mathcal{I} \text{ et } x \in Var(S_\iota), \text{ il existe } \iota' \leq_\mathcal{I} \iota \text{ tel que } x \in Var(R_{\iota'})$$

avec $\leq_\mathcal{I}$ *la clôture réflexive de* $<_\mathcal{I}$ *(i.e.* $\iota \leq_\mathcal{I} \iota'$ *ssi* $\iota <_\mathcal{I} \iota'$ *ou* $\iota = \iota'$*).*

Dans cette définition, S est l'ensemble des connaissances initiales de l'intrus, et $\mathcal{I}$ est un ensemble d'indices permettant d'identifier tous les pas de protocoles de la spécification. La condition C_1 nous assure que tout principal utilisant une variable (x) dans un envoi de message en connaisse la valeur, i.e. il existe un pas de protocole précédent (ι') où cette variable a été instanciée. Enfin, $<_\mathcal{I}$ décrit toutes les exécutions possibles du protocole, c'est-à-dire tous les ordonnancements admissibles des pas de protocoles. Ainsi, si l'on spécifie par exemple deux rôles A et B possédant chacun trois pas d'indices respectifs A_1, A_2, A_3 et B_1, B_2, B_3 avec $\mathcal{I} = \{A_1, A_2, A_3, B_1, B_2, B_3\}$, on demandera que l'ordre partiel vérifie $A_1 <_\mathcal{I} A_2 <_\mathcal{I} A_3$ et $B_1 <_\mathcal{I} B_2 <_\mathcal{I} B_3$. De cette manière, on peut décrire tous les ordonnancements possibles entre les pas de plusieurs principaux. Par exemple, si l'on a quatre pas a, b, c et d, on peut définir l'ordre partiel $a <_\mathcal{I} b <_\mathcal{I} d$ et $a <_\mathcal{I} c <_\mathcal{I} d$ pour définir un rôle exécutant les pas b et c dans n'importe quel ordre, mais après a et avant d. On cherchera des attaques quel que soit le comportement du principal à ce moment là, i.e. qu'il choisisse d'exécuter b ou c en premier. Enfin, on définit l'ensemble des ordres d'exécutions d'un protocole, i.e. les ordonnancements admissibles de pas de protocole.

Définition 2.3.1.7 *Ordre d'exécution.*

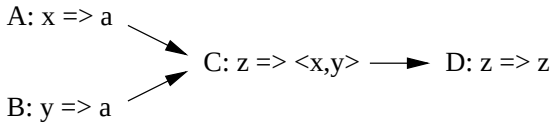
Étant donné un protocole $P = (\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}\}, <_\mathcal{I}, S)$*, un ordre d'exécution* π *sur* P*, de domaine* $\mathcal{J}$*, est un ordre total sur* $\mathcal{J} \subset \mathcal{I}$ *compatible avec* $<_\mathcal{I}$ *et stable par prédécesseur, i.e. pour tous* $a \in \mathcal{I}$ *et* $b \in \mathcal{J}$*, si* $a <_\mathcal{I} b$ *alors* $a \in \mathcal{J}$ *et* $a\pi b$*.*

On peut également voir π comme une bijection de son domaine $\mathcal{J}$ dans $\{1, ..\#\mathcal{J}\} \subset \mathbb{N}$, avec l'ordre naturel sur les entiers. On décrit ainsi tous les ordres admissibles par la spécification de protocole. De plus, on notera $\#\pi = \#\mathcal{J}$. Pour fixer les idées, un ordre d'exécution totale, correspondant à une exécution (totale) du protocole menant tous les principaux au terme de leurs pas de protocole respectifs, est un ordre d'exécution de domaine $\mathcal{J} = \mathcal{I}$, i.e. un ordre total sur $\mathcal{I}$ compatible avec $<_\mathcal{I}$: si $a <_\mathcal{I} b$ pour $\{a, b\} \subset \mathcal{I}$, alors $a\pi b$. En rassemblant un ordre d'exécution et une substitution close, on obtient :

Définition 2.3.1.8 *Exécution de protocole.*

Une exécution d'un protocole P est un couple (π, σ) composé d'un ordre d'exécution π sur P et d'une substitution close σ sur Var.

On peut se demander pourquoi, dans la définition d'ordre d'exécution, on demande que $\forall a \in \mathcal{I}$, $\forall b \in \mathcal{J}$, si $a <_{\mathcal{I}} b$ alors $a \in \mathcal{J}$. Il y a deux raisons à cela. La première et la plus évidente est d'imposer à toute exécution de commencer "au début" du protocole. La seconde raison tiens du fait que l'on doit être sûr que toute variable utilisée dans l'envoi d'un message a été précédemment reçu. Par exemple, considérons le protocole suivant à quatre pas, d'ordre $A <_{\mathcal{I}} C$, $B <_{\mathcal{I}} C$ et $C <_{\mathcal{I}} D$, et de connaissances initiales de l'intrus $S = \{Init\}$:



Dans ce protocole, toute exécution utilisant le pas C exécutera les pas A et B avant C, ce qui rend l'utilisation des deux variables x et y possible dans ce dernier pas. Par contre, si l'on avait autorisé $A - C$ comme exécution, la variable y du pas C n'aurait pas été définie.

2.3.2 Exemple

Le protocole bien connu de Needham Schroeder [74] étant un très bon exemple pour présenter une spécification de protocoles cryptographiques, nous allons le présenter dans notre formalisme. Des protocoles plus complexes seront présentés à la section 2.4.7. Il s'agit ici d'une variante du protocole d'origine. On rappelle que les nonces sont supposés appartenir aux connaissances initiales des principaux censés les générer. De plus, dans cette variante du protocole un principal A voulant communiquer avec B va lui envoyer sa clef publique à la place de son nom dans la version d'origine. On obtient la spécification suivante, pour deux principaux A et B, avec $\mathcal{I} = \{A_1, A_2, B_1, B_2\}$, et avec $\iota : R \Rightarrow S$ représentant le pas de protocole $R_\iota \Rightarrow S_\iota$:

$$\begin{array}{llllllll} (& A_1 : & Init & \Rightarrow & \{\langle N_A, K_A\rangle\}^p_{K_B} & B_1 : & \{\langle x_2, x_3\rangle\}^p_{K_B} \Rightarrow & \{\langle x_2, N_B\rangle\}^p_{x_3} \\ & A_2 : & \{\langle N_A, y_1\rangle\}^p_{K_A} & \Rightarrow & \{y_1\}^p_{K_B} & B_2 : & \{N_B\}^p_{K_B} \Rightarrow & End \quad) \end{array}$$

avec l'ordre $<_{\mathcal{I}}$ suivant : $A_1 <_{\mathcal{I}} A_2$ et $B_1 <_{\mathcal{I}} B_2$, avec les connaissances initiales de l'intrus $S = \{Init, A, B, K_A, K_B\}$, et avec $Clefs = \{K_A, {K_A}^*, K_B, {K_B}^*\}$. Tous les autres atomes, comme N_A et N_B, sont dans $Noms$. On se rend bien compte sur cet exemple que les domaines de variables sont globaux, d'où l'utilisation de variables différentes pour les deux principaux. Nous verrons à la section 2.3.5 une attaque sur ce protocole.

2.3.3 Modèle d'intrus de Dolev-Yao

Dans le modèle de Dolev-Yao [43], l'intrus est capable d'intercepter, mémoriser et envoyer les messages sous n'importe quelle fausse identité. De plus, il possède au moins les mêmes capacités de calcul que les principaux, i.e. décomposer les couples, encrypter et décrypter des messages quand il en possède la clef. Pour simplifier le modèle, et comme introduit à la section 1.1.3, on considère que notre intrus intercepte toujours tous les messages envoyés par les principaux (divert automatique). Ce n'est pas une restriction, car l'intrus peut toujours au moins intercepter un message et le renvoyer sans modification au destinataire théorique. Ainsi, dans ce modèle toutes les communications entre les principaux sont transmises par un environnement hostile, représenté par l'intrus. On peut définir plusieurs extensions du modèle de Dolev-Yao, en fonction

	Règles de décomposition		Règles de composition	
Paire :	$L_{p_1}(\langle a,b\rangle)$:	$\langle a,b\rangle \rightarrow a$	$L_c(\langle a,b\rangle)$:	$a,\, b \rightarrow \langle a,b\rangle$
	$L_{p_2}(\langle a,b\rangle)$:	$\langle a,b\rangle \rightarrow b$		
Enc. asymétrique :	$L_{ad}(\{a\}^p_K)$:	$\{a\}^p_K\,,\, K^* \rightarrow a$	$L_{ac}(\{a\}^p_K)$:	$a,\, K \rightarrow \{a\}^p_K$
Enc. symétrique :	$L_{sd}(\{a\}^s_b)$:	$\{a\}^s_b\,,\, b \rightarrow a$	$L_{sc}(\{a\}^s_b)$:	$a,\, b \rightarrow \{a\}^s_b$

TAB. 2.1: Règles de l'intrus de Dolev-Yao

des capacités que l'on veut donner à l'intrus. Pour pouvoir décrire précisément les capacités accordées à l'intrus, on simule ses actions (de modification des messages) par des règles de déduction sur des ensembles de messages, appelées règles d'intrus.

Définition 2.3.3.1 *Règle d'intrus.*
Une règle d'intrus est une une règle de déduction $M \rightarrow t$, avec M un ensemble de messages et t un message.

Une règle d'intrus $L = M \rightarrow t$ peut être appliquée à un ensemble E de messages quand $M \subseteq E$, et le résultat est l'ensemble de messages E, t. On notera $E \rightarrow_L E, t$ l'application de la règle L à l'ensemble E. Étant donné que E, t est un ensemble de messages, on peut chaîner cette notation pour décrire l'application des règles L_1 à L_n sur l'ensemble E, avec : $E \rightarrow_{L_1} E, t_1 \rightarrow_{L_2} ... \rightarrow_{L_n} E, t_1, .., t_n$. On étend la notation $\rightarrow_L$ à des ensembles de règles d'intrus : si $\mathcal{L}$ est un ensemble fini ou infini de règles d'intrus, on note $\rightarrow_{\mathcal{L}}$ la relation binaire correspondante sur des ensembles finis de messages. Ainsi, $E \rightarrow_{\mathcal{L}} E'$ ssi il existe $L \in \mathcal{L}$ telle que $E \rightarrow_L E'$. Enfin, $\rightarrow^*_{\mathcal{L}}$ est la fermeture réflexive et transitive de $\rightarrow_{\mathcal{L}}$. Un intrus est alors formé par plusieurs règles d'intrus :

Définition 2.3.3.2 *Intrus.*
Un intrus $\mathcal{L}$ (ou environnement hostile de type $\mathcal{L}$) est un intrus disposant d'un ensemble $\mathcal{L}$ de règles d'intrus.

On fera souvent la confusion entre un intrus et l'ensemble des règles d'intrus qui le définissent. Les capacités de base de l'intrus de Dolev-Yao [43], et les règles d'intrus correspondantes, sont retranscrites dans la Table 2.1. Pour tous messages a et b et toute clef K dans ce tableau, la notation $L_{..}(..)$ décrit un ensemble d'instances de règles d'intrus. On répartit ces règles dans différents ensembles :
- $L_d(t) = L_{p_1}(t) \cup L_{p_2}(t)$ si $t = \langle \_, \_ \rangle$ et $L_d(t) = L_{ad}(t) \cup L_{sd}(t)$ si $t = \{\_\}_\_$.
- $L_d = \bigcup_a L_d(a)$, et $L_c = \bigcup_a L_c(a)$.

Ceci nous donne :

Définition 2.3.3.3 *Intrus DY (de Dolev-Yao)*
L'intrus DY est l'intrus disposant de l'ensemble $\mathcal{L}_{DY}$ de règles d'intrus $L_d \cup L_c$.

La définition d'un intrus nous permet de décrire l'ensemble (infini) des messages qu'un intrus peut calculer à partir d'un ensemble fini de messages. Étant donné un intrus décrit par un

ensemble de règles $\mathcal{L}$, on note

$$forge_{\mathcal{L}}(E) = \left\{E' \mid E \rightarrow_{\mathcal{L}}^{*} E'\right\}$$

l'ensemble de tous les messages accessibles à partir de E avec les règles de déduction $\mathcal{L}$. Lorsque l'intrus a pour but de construire un message particulier à partir d'un ensemble de messages, on veut pouvoir décrire pas à pas toutes les règles d'intrus utilisées. Pour cela :

Définition 2.3.3.4 *Dérivations.*

Soit $\mathcal{L}$ un intrus, soit E un ensemble de messages, et soit t un message. Une dérivation D de longueur $n \in \mathbb{N}$, partant de E et de but t est une séquence non répétitive d'applications de règles d'intrus

$$E \rightarrow_{L_1} E, t_1 \rightarrow_{L_2} \ldots \rightarrow_{L_n} E, t_1, .., t_n$$

avec pour tout $i \in \{1..n\}$, $t_i \in Messages$ et $L_i \in \mathcal{L}$, telle que $t_i \neq t_j$ si $i \neq j$.

Pour toute règle d'intrus L, on notera $L \in D$ quand il existe $i \in \{1, , n\}$ tel que $L = L_i$.

Il est important de remarquer que l'on interdit aux dérivations d'être (inutilement) répétitives, i.e. de créer deux fois le même terme. Comme chaque règle d'intrus crée au plus un seul nouveau message, ce n'est en rien restrictif. Pour le moment, on ne définit que l'intrus *DY* (de base), mais nous verrons aux chapitres suivants d'autres intrus, tous construits en ajoutant des règles supplémentaires à l'intrus de base.

2.3.4 Interprétation du modèle Alice-Bob dans le modèle par rôles

En pratique, le programme CASRUL [19] permettant de traduire automatiquement toute spécification de protocole cryptographique écrite dans une syntaxe équivalente au modèle Alice-Bob vers un Format Intermédiaire, contenant un équivalent du modèle par rôles (voir [19, 26] pour une présentation de cet outil et une formalisation de la traduction réalisée). Nous ne présenterons donc pas ici formellement de traduction du modèle Alice-Bob vers le modèle par rôles. En revanche, nous allons donner quelques exemples pour guider l'intuition.

La principale difficulté dans l'interprétation du modèle Alice-Bob consiste à identifier quelles connaissances possède un principal donné à un moment donné de l'exécution d'un protocole. En effet, un principal ne peut généralement pas décomposer entièrement tous les messages qu'il reçoit au cours d'une exécution. Ceci peut servir à l'intrus pour construire une attaque : il pourra peut-être remplacer un sous terme d'un message non identifiable par un principal par un autre. Par exemple, considérons le protocole suivant :

$$\begin{array}{ll} A \rightarrow B : & \langle \{a\}_b^s , \{c\}_d^s \rangle \\ B \rightarrow A : & \langle \{a\}_b^s , c \rangle \end{array}$$

Avec A connaissant initialement a, b, c et d, et B ne connaissant initialement que d. Comme B ne connaît pas b à la réception de $\langle \{a\}_b^s , \{c\}_d^s \rangle$, il ne peut pas calculer a, i.e. $\{a\}_b^s$ reste non décomposable pour lui à ce moment de l'exécution du protocole. En revanche, B connaît d, il peut donc décomposer $\{c\}_d^s$ (il vérifie au passage que c'est bien une encryption par la clef d), et obtient ainsi c. Pour modéliser ces actions de B dans le modèle par rôles, on utilise deux variables x et y, représentant respectivement ce que B pense être $\{a\}_b^s$ et c. Ainsi, le message attendu par B dans le modèle par rôles sera $\langle x, \{y\}_d^s \rangle$. En effet, quelles que soient les valeurs de x et de y, le principal B acceptera à ce point de l'exécution n'importe quel message ayant cette structure. En réponse, le principal B doit envoyer le message $\langle \{a\}_b^s , c \rangle$. Il ne connaît pas

a, mais il croit connaître $\{a\}_b^s$ et c. Il va donc émettre le message $\langle x, y\rangle$. En résumé, le premier pas de B dans le modèle par rôles sera :

$$B_1 \ : \quad \langle x, \{y\}_d^s\rangle \ \Rightarrow \ \langle x, y\rangle$$

De plus, les connaissances de B ont changées. A présent, il connaît l'atome d et les valeurs des variables x et y, qu'il pense être $\{a\}_b^s$ et c. On peut remarquer d'emblée que face à un intrus, la variable x n'a que peu de chances d'avoir cette valeur. En effet, x a été transmis non crypté, l'intrus peut le remplacer par n'importe quel terme qu'il peut construire. En parallèle de cela, le principal A exécute lui aussi un premier pas de protocole. Il s'agit simplement de l'envoi du premier message, i.e. $\langle \{a\}_b^s\,, \{c\}_d^s\rangle$, que le principal A peut construire à partir de ses connaissances. On a donc, dans le modèle par rôles :

$$A_1 \ : \quad \mathit{Init} \ \Rightarrow \ \langle \{a\}_b^s\,, \{c\}_d^s\rangle$$

Le message reçus par A est Init pour permettre à l'intrus de lancer l'exécution du protocole (et car syntaxiquement on a besoin d'un message reçu à chaque pas pour éviter les cas particuliers). On peut remarquer ici que non seulement un protocole écrit dans le modèle Alice-Bob doit être interprété, mais en plus on doit vérifier qu'il est réellement exécutable par les différents principaux. Par exemple, si A ne connaissait pas a, la traduction devrait échouer sur la création du terme $\langle \{a\}_b^s\,, \{c\}_d^s\rangle$ par A. Enfin, il nous reste à donner le second pas de protocole exécuté par A. Il s'agit de la réception du message $\langle \{a\}_b^s\,, c\rangle$ émis par B. Comme A connaît tous les atomes composant de messages, on a :

$$A_2 \ : \quad \langle \{a\}_b^s\,, c\rangle \ \Rightarrow \ \mathit{End}$$

L'atome End émis par A n'est là que pour éviter d'avoir une syntaxe différente pour le dernier pas de protocole, et ne doit donc pas interférer avec la recherche d'attaques. On choisis donc un atome absent de la spécification du protocole dans le modèle Alice-Bob. Ces trois pas de protocole (A_1, A_2 et B_1) forment alors un protocole dans le modèle par rôles, avec l'ordre partiel $A_1 <_{\mathcal{I}} A_2$ et par exemple les connaissances initiales de l'intrus $S_0 = \{\mathit{Init}\}$. Pour tester la résistance d'un protocole face à la perte de telle ou telle donnée, ou peut fournir à l'intrus des données publiques, on peut compléter les connaissances initiales à l'intrus.

En outre, on peut également donner certains rôles à l 'intrus. Par exemple, supposons dans le protocole précédent que A soit honnête mais que B ne le soit pas. On a alors les mêmes pas de protocole A_1 et A_2 pour A, mais on remplace le pas de B (ici B_1) par des connaissances initiales supplémentaires pour l'intrus : S_0 devient $\{\mathit{Init}, d\}$ puisque l'intrus jouant B doit connaître au moins les connaissances initiales de B. Nous n'avons plus besoin de spécifier le pas B_1, car l'intrus peut exécuter n'importe quelle déduction de connaissance ou construction de message réalisée par un principal honnête, à partir du moment où il possède les connaissances nécessaires. On obtient donc le protocole formé par les deux pas A_1 et A_2, l'ordre partiel $A_1 <_{\mathcal{I}} A_2$, et les connaissances initiales $S_0 = \{\mathit{Init}, d\}$.

Il est intéressant de remarquer que dans le modèle par rôles, nous n'avons pas besoin de conserver explicitement les connaissances des principaux. Comme on sait que le protocole est exécutable (c'est garanti par la traduction avec CASRUL et en fait aussi par la définition de protocole dans le modèle par rôles), on n'a pas besoin de vérifier qu'un principal possède toutes les connaissances (en particulier les atomes) nécessaires à la création d'un message envoyé ou à la décomposition d'un message reçus. Nous n'avons donc besoin que de conserver les connaissances acquises par les principaux, c'est à dire les valeurs des variables. De plus, on peut remarquer que

même si un principal de peut pas décomposer une connaissance donnée, il peut quand même en tester l'égalité avec un message reçus par ailleurs. Par exemple, pour modéliser le protocole suivant :

$$\begin{array}{ll} A \to B : & \langle \{a\}_d^s, a \rangle \\ B \to A : & a \end{array}$$

avec les mêmes connaissances initiales que précédemment, i.e. B connaît d. Ici, B va calculer a à partir de $\{a\}_d$ et vérifier qu'il est égal au second membre de la paire. On spécifie ceci de le modèle par rôles avec le pas de protocole suivant :

$$B_1 \ : \quad \langle \{x\}_d^s, x \rangle$$

avec une nouvelle variable x. Dans le cas d'une encryption asymétrique, le traducteur (CASRUL) doit en plus trouver la manière dont le principal va effectuer ses vérifications. Typiquement, si B reçoit $\langle \{a\}_k^p, a \rangle$ et qu'il connaît k^*, il va décrypter $\{a\}_k^p$ pour obtenir a, mais s'il ne connaît que k il va calculer $\{a\}_k^p$ à partir de a (membre de droite de la paire) et de k, puis vérifier que le résultat est bien égal au membre de gauche de la paire.

Enfin, dans certains cas le modèle par rôles semble ne pas suffire pour modéliser des protocoles Alice-Bob (mais c'est une illusion). Considérons par exemple le protocole suivant, du point de vue de B :

$$\begin{array}{ll} A \to B : & \{a\}_b^s \\ B \to A : & \{a\}_b^s \\ A \to B : & b \\ B \to A : & a \end{array}$$

avec B sans connaissances initiales (et A connaissant a et b). Le premier pas de protocole pour B consiste simplement à acquérir une nouvelle connaissance non décomposable $\{a\}_b^s$, et à la renvoyer, i.e. :

$$B_1 \ : \quad x \Rightarrow x$$

avec x une nouvelle variable. Pour le moment, le principal B n'a aucun moyen de décomposer x (ni de vérifier que c'est bien une encryption). Cependant, B reçoit ensuite la clef b de la part de A, qu'il met dans une nouvelle variable y, et il peut maintenant décomposer la connaissance précédemment acquise. Cela revient à définir une variable z dont la valeur est le résultat de la décomposition de x avec la clef y. Seul problème, nous de disposons pas d'un tel opérateur de décryptage, qui compliquerait (inutilement, en fait) la recherche d'attaques. A la place, nous définissons le pas de protocole suivant :

$$B_2 \ : \quad y \Rightarrow z \quad [x = \{z\}_y^s]$$

où $[x = \{z\}_y^s]$ représente l'unification du terme x avec le terme $\{z\}_y^s$, i.e. l'égalité $\sigma(x) = \{\sigma(z)\}_{\sigma(y)}^s$ pour que σ soit une attaque. Ceci est une extension du modèle de protocole par rôles. Néanmoins, nous verrons à la section 2.5.1 que l'on peut ramener une telle extension au modèle par rôles défini précédemment.

2.3.5 Formalisation du problème de l'insécurité

Après avoir défini les notions d'exécution de protocole et d'intrus, on peut définir les attaques. Pour construire une attaque sur un protocole P, l'intrus commence par choisir un ordre d'exécution sur P, puis tente de construire tous les messages demandés par les principaux (dans

l'ordre choisi) de manière à obtenir le message spécial $Secret$ à la fin. Ces messages sont construits à partir des connaissances initiales de l'intrus et des messages envoyés par les principaux, avec les règles de réécriture de l'intrus choisi. Le modèle de protocole que l'on considère ne définit en fait qu'une seule session de protocole, puisque toute boucle ou duplication de rôle est exclue. Néanmoins, pour traiter un nombre fini n de sessions séquentielles (même principal) ou parallèle (principal dupliqué), il suffit de définir une spécification de protocole comportant n répétitions d'une même session (les unes après les autres ou en même temps), en prenant soin de renommer les variables. En revanche, un outil de vérification comme Atsé pourrait être tenté d'éviter ces duplications. La notion d'attaque que l'on considère ici est assez standard dans le cas d'un nombre fini de sessions (c.f. [82]).

Définition 2.3.5.1 *Attaque*

Soit $P = (\{R_\iota \Rightarrow S_\iota \mid \iota \in \mathcal{I}\}, <_\mathcal{I}, S_0)$ *un protocole (sur Atomes et* Var*), et soit* $\mathcal{L}$ *un intrus. Alors une attaque sur* P *avec l'intrus* $\mathcal{L}$ *est une exécution* (π, σ) *de* P*, avec* π *de domaine* $\mathcal{J}$ *vu comme une bijection de* $\mathcal{J}$ *dans* $\{1, .., \#\mathcal{J}\} \subset \mathbb{N}$*, telle que :*

$$\begin{array}{rcl} \forall i \in \{1, .., \#\mathcal{J}\}, \quad R'_i\sigma & \in & forge_\mathcal{L}(S_0, S'_1\sigma, ..., S'_{i-1}\sigma) \\ Secret & \in & forge_\mathcal{L}(S_0, S'_1\sigma, ..., S'_{\#\mathcal{J}}\sigma) \end{array}$$

avec $R_\iota = R'_{\pi(\iota)}$ *et* $S_\iota = S'_{\pi(\iota)}$ *pour tout* $\iota \in \mathcal{J}$*.*

Une attaque (π, σ) *sur* P *avec l'intrus* $\mathcal{L}$ *est dite minimale quand* $\sum_{x \in Var} |\sigma(x), Init|_{dag}$ *est minimale parmi toutes les attaques sur* P *avec* $\mathcal{L}$*.*

La présence de l'atome $Init$ dans la mesure d'attaque minimale est purement technique, et nous permettra de remplacer certains sous termes de σ par $Init$. On peut remarquer que l'on ne cherche que des attaques de secret. Nous pensons que les autres attaques classiques peuvent être traitées de la même manière. Par exemple, les attaques d'authentification nécessitent d'atteindre un point du protocole où un principal B croit posséder une connaissance envoyée par A, i.e. non corrompue pendant le transfert. Ici, cela se traduit par atteindre un point de l'exécution d'un protocole où $\sigma(x) \neq t$, pour une variable x et un message t. Or ce test se vérifie très facilement (sur des messages, i.e. des termes clos), et ne pose donc pas plus de problème que pour une recherche de secret.

Étant donné un intrus $\mathcal{L}$, le problème de décision que l'on souhaite résoudre est alors cet ensemble de protocoles :

$$Insecure(\mathcal{L}) \ := \ \{\, P \mid \text{il existe une attaque sur } P \text{ avec l'intrus } \mathcal{L} \,\}$$

En particulier, le cas de base de l'insécurité face à l'intrus de Dolev-Yao est $Insecure(DY)$.

2.3.6 Extension de Dolev-Yao par cryptage involutif

L'intrus (et les attaques) de base que l'on vient de définir reposent sur une notion d'opérateur d'encryption ne possédant strictement aucune propriété utilisable par l'intrus. Mais même si c'est le but de tout opérateur concret, il est quelque fois possible ou nécessaire d'avoir quelques propriétés supplémentaires. Par exemple, on peut vouloir tenter de gommer la différence entre les opérateurs d'encryption et de décryptage. Cela se traduit par le fait qu'un message M encrypté successivement par une clef privée et la clef publique correspondante est suffisamment proche du message M d'origine pour que l'intrus puisse le calculer. Cela donne un règle de la forme $\{\{M\}^p_K\}^p_{K^*} \rightarrow M$. De même avec deux encryptions par une clef symétrique ($M = \{\{M\}^s_t\}^s_t$). Pour modéliser ceci, on définit de nouvelles règles d'intrus, décrites dans la Table 2.2.

Règles de décomposition		Règles de composition	
$L_s(\{\{a\}_b^s\}_b^s):$	$\{\{a\}_b^s\}_b^s \to a$	$L_r(\{\{a\}_b^s\}_b^s):$	$a \to \{\{a\}_b^s\}_b^s$
$L_s(\{\{a\}_K^p\}_{K^*}^p):$	$\{\{a\}_K^p\}_{K^*}^p \to a$	$L_r(\{\{a\}_K^p\}_{K^*}^p):$	$a \to \{\{a\}_K^p\}_{K^*}^p$

TAB. 2.2: Règles de l'intrus involutif.

Définition 2.3.6.1 *Intrus Involutif (Dolev-Yao avec $\{..\}_{..}^{..}$ involutif)*

Soient L_s et L_r les ensembles de règles décrites par la Table 2.2.

L'intrus Inv (pour involutif) est l'intrus disposant de l'ensemble $\mathcal{L}_{Inv}$ de règles d'intrus $\mathcal{L}_{DY} \cup L_s \cup L_r$.

Les notions d'ensemble de messages constructibles par l'intrus ($forge_{L_{Inv}}$) et d'attaque avec ces règles découlent des définitions des sections précédentes. Ceci permet de trouver des attaques dans des protocoles sûrs en présence de l'intrus de Dolev-Yao. Par exemple, considérons le protocole suivant :

$$\begin{array}{lllll} \{ & A_1 : & \{x\}_{K^*}^p & \Rightarrow & x \\ & A_2 : & \{\{y\}_P^p\}_K^p & \Rightarrow & y \quad \} \end{array}$$

Les connaissances initiales de l'intrus sont $S = \{\{Secret\}_P^p\}$, et l'ordre sur les pas de protocole est $A_1 <_{\mathcal{I}} A_2$, pour $\mathcal{I} = \{A_1, A_2\}$. Les atomes K et P sont dans $Clefs$, et tous les autres atomes sont dans $Noms$. Ce protocole n'a pas d'attaque face à l'intrus de base, car ce intrus ne peut construire aucun message encrypté par K^*. En revanche, l'intrus de Dolev-Yao étendu peut exécuter l'attaque suivante :

$$\begin{array}{rcl} \pi & = & <_{\mathcal{I}} \\ \sigma(x) & = & \{\{Secret\}_P^p\}_K^p \\ \sigma(y) & = & Secret \end{array}$$

En effet, le premier message attendu par le principal A est obtenu avec la règle d'intrus

$$\begin{array}{rcl} L_r(\{\{\{Secret\}_P^p\}_K^p\}_{K^*}^p) & = & \{Secret\}_P^p \to \{\{\{Secret\}_P^p\}_K^p\}_{K^*}^p \\ & & \text{puisque} \quad \{\{\{Secret\}_P^p\}_K^p\}_{K^*}^p = \sigma(\{x\}_{K^*}^p) \end{array}$$

et le second message attendu par A n'est autre que le premier message envoyé par A, i.e. $\sigma(x)$.

Nous verrons à la section 3.6.1 que la recherche d'attaques dans ce modèle n'est pas plus difficile que dans le cas de base. Mais ce modèle peut encore être amélioré, puisque les règles d'intrus L_s et L_r ne peuvent être appliquées qu'au sommet d'un message : on pourrait aussi considérer le cas où elles sont applicables à n'importe quel sous-terme.

2.3.7 Extension de Dolev-Yao par règles préfixe

La simplification des opérateurs de chiffrement, i.e. de l'encryption, (au moins en tête de messages) est loin d'être la seule propriété des opérateurs cryptographiques que l'on souhaite

Règles de décomposition	**Règles de composition**
néant	$L_{prefix}(\{M\}^s_K): \quad \{\langle .. \langle M, M_1\rangle, ..M_n\rangle\}^s_K \rightarrow \{M\}^s_K$

TAB. 2.3: Règles de l'intrus préfixe.

étudier. Une autre propriété intéressante est la propriété de décomposition préfixe de l'opérateur d'encryption, utilisé dans les attaques de "Cipher Block Chaining", ou chaînage de blocs d'encryption. Étant donnés trois messages M_1, M_2 et K, cette propriété de l'encryption consiste à pouvoir calculer simplement $\{M_1\}^s_K$ à partir de $\{\langle M_1, M_2\rangle\}^s_K$. Ceci est rendu possible par certains opérateurs d'encryption consistant à crypter les différents morceaux (blocs) d'un message les uns après les autres en les chaînant les uns aux autres. Ceci permet d'éviter que l'intrus ne puisse remplacer ou permuter des blocs. Mais cette méthode reste vulnérable dans certains cas à un intrus coupant le message chiffré entre deux blocs (ici M_1 et M_2). Nous voulons donc donner à l'intrus la capacité de "couper" les blocs de chiffrement qu'il connaît pour en construire d'autres. Les règles d'intrus correspondantes sont décrites dans la Table 2.3, pour tous messages M, M_1, ..., M_n. On note L_{prefix} l'ensemble des règles d'intrus décrites dans ce tableau.

Définition 2.3.7.1 *Intrus Préfixe.*
L'intrus Préfixe est l'ensemble $\mathcal{L}_{prefixe}$ des règles d'intrus $\mathcal{L}_{DY} \cup L_{prefixe}$.

Toutes les définitions précédentes de messages constructibles par l'intrus ($forge$) et d'attaque sont évidement également valables pour cet intrus. Bien que de telles déductions de connaissances ne soient pas toujours possibles par l'intrus (en particulier, la taille du message M doit correspondre à la taille d'un bloc d'encryption), il est assez difficile en pratique de prévoir à priori si une telle décomposition sera possible et quelles en seront les conséquences. Ainsi, l'intrus préfixe correspond au pire des cas, quand toutes les décompositions sont possibles. Il est raisonnable de demander qu'un protocole soit sûr face à l'intrus préfixe, car cela garantit sa sûreté face à toutes les attaques de ce type (déduction préfixe). Pour illustrer ceci, nous présentons une (autre !) variante du protocole de Needham-Schroeder [74][4]. Cette version est très intéressante, car elle a d'abord été considérée comme correcte dans l'étude de Clark & Jacob [28], mais ensuite une étude approfondie de Pereira & Quisquater [77] a montré qu'elle est en fait vulnérable à une attaque de type préfixe, quand l'opérateur d'encryption est du type chaînage de blocs chiffrés décrit plus haut et utilise des blocs élémentaires de la même taille que tous les atomes utilisés dans le protocole. La spécification de ce protocole est la suivante. Pour fixer les idées, on donne d'abord la version dans le modèle Alice-Bob, puis la version dans le modèle par rôles.

1. $A \rightarrow S: \quad A,\ B,\ N_A$
2. $S \rightarrow A: \quad \left\{N_A,\ B,\ K_{AB},\ \{K_{AB},\ A\}^s_{K_{BS}}\right\}^s_{K_{AS}}$
3. $A \rightarrow B: \quad \{K_{AB},\ A\}^s_{K_{BS}}$
4. $B \rightarrow A: \quad \{N_B\}^s_{K_{AB}}$
5. $A \rightarrow B: \quad \{N_B - 1\}^s_{K_{AB}}$
6. $B \rightarrow A: \quad \{Secret\}^s_{K_{AB}}$

[4]Peut-être le protocole bogué le plus étudié de l'histoire des protocoles cryptographiques... Mais ses qualités pédagogiques sont remarquables.

Le but de ce protocole est de permettre au principal A de transmettre une donnée secrète *Secret* à B, en utilisant un serveur connu/partagé S. Le principal A commence par récupérer auprès de S une clef temporaire K_{AB} devant être partagée entre A et B pour cette session (uniquement). A transmet ensuite cette nouvelle clef K_{AB} à B. A et B vérifient alors mutuellement leurs identités, puis B considère qu'il peut maintenant transmettre à A le message *Secret* encrypté par la clef temporaire K_{AB} en toute sécurité. Pour attaquer ce protocole grâce aux règles préfixe, nous allons considérer deux sessions concurrentes de ce protocole, avec les principaux A,B, et S. Les deux principaux A et B joueront tous deux les deux rôles de ce protocole, et S jouera le serveur dans les deux sessions. Dans le modèle par rôle, ceci nous donne la spécification suivante :

$$
\begin{array}{lrcl}
A_1: & \mathit{Init} & \Rightarrow & A,\, B,\, N_A \\
S_1: & A,\, B,\, x_{N_A} & \Rightarrow & \left\{x_{N_A},\, B,\, K_{AB},\, \{K_{AB},\, A\}^s_{K_{BS}}\right\}^s_{K_{AS}} \\
A_2: & \{N_A,\, B,\, x_{K_{AB}},\, x_{K_{AB}K_{BS}}\}^s_{K_{AS}} & \Rightarrow & x_{K_{AB}K_{BS}} \\
B_1: & \left\{x'_{K_{AB}},\, A\right\}^s_{K_{BS}} & \Rightarrow & \{N_B\}^s_{x'_{K_{AB}}} \\
A_3: & \{x_{N_B}\}^s_{x_{K_{AB}}} & \Rightarrow & \{x_{N_B}-1\}^s_{x_{K_{AB}}} \\
B_2: & \{N_B-1\}^s_{x'_{K_{AB}}} & \Rightarrow & \{\mathit{Secret}\}^s_{x'_{K_{AB}}} \\
A_4: & \{\mathit{Secret}\}^s_{x_{K_{AB}}} & \Rightarrow & \mathit{End}
\end{array}
$$

$$
\begin{array}{lrcl}
B'_1: & \mathit{Init} & \Rightarrow & B,\, A,\, N'_B \\
S'_1: & B,\, A,\, x_{N'_B} & \Rightarrow & \left\{x_{N'_B},\, A,\, K_{BA},\, \{K_{BA},\, B\}^s_{K_{AS}}\right\}^s_{K_{BS}} \\
B'_2: & \{N'_B,\, A,\, x_{K_{BA}},\, x_{K_{BA}K_{AS}}\}^s_{K_{BS}} & \Rightarrow & x_{K_{BA}K_{AS}} \\
A'_1: & \left\{x'_{K_{BA}},\, B\right\}^s_{K_{AS}} & \Rightarrow & \{N'_A\}^s_{x'_{K_{BA}}} \\
B'_3: & \left\{x_{N'_A}\right\}^s_{x_{K_{BA}}} & \Rightarrow & \left\{x_{N'_A}-1\right\}^s_{x_{K_{BA}}} \\
A'_2: & \{N'_A-1\}^s_{x'_{K_{BA}}} & \Rightarrow & \{\mathit{Secret}\}^s_{x'_{K_{BA}}} \\
B'_4: & \{\mathit{Secret}\}^s_{x_{K_{BA}}} & \Rightarrow & \mathit{End}
\end{array}
$$

Les atomes considérés sont tous dans Noms, et l'ordre sur les pas de protocole est simplement l'ordre habituel rôle par rôle : $A_1 <_{\mathcal{I}} A_2 <_{\mathcal{I}} A_3 <_{\mathcal{I}} A_4$, $B_1 <_{\mathcal{I}} B_2 <_{\mathcal{I}} B_3$, etc... avec $\mathcal{I} = \{A_i, B_i, S_i, B'_i, A'_i, S'_i \mid i \in I\!N\}$. Enfin, les connaissances initiales de l'intrus sont restreintes à $S = \{\mathit{Init}\}$. Ce protocole utilise un opérateur $-$ sur les atomes. Il ne sert qu'à fabriquer aisément, et de manière réversible, un nouveau nonce à partir du premier. Ceci se code très facilement à l'aide d'une encryption symétrique dont la clef est connue de tous. A la différence de $N_B - 1$, le résultat n'est pas atomique, mais comme on travaille sur des termes non typés, cela n'a aucune importance. Pour simplifier, on a conservé la notation $N_B - 1$ d'origine. Pour l'intrus, l'attaque consiste à récupérer le premier message transmis par S lors de la première session, à le tronquer grâce à la règle préfixe, puis à utiliser le message ainsi calculé dans la seconde session pour berner A' sur la clef temporaire K_{BA} qu'il doit utiliser pour transmettre le secret. Cette attaque est la suivante, pour π de domaine A_1, S_1, A'_1, A'_2 :

$$
\begin{array}{rl}
\pi: & A_1 <_\pi S_1 <_\pi A'_1 <_\pi A'_2 \\
\sigma(x_{N_A}) = & N_A \\
\sigma(x'_{K_{BA}}) = & N_A
\end{array}
$$

Il nous reste à décrire la manière dont l'intrus construit les messages attendus par les principaux lors de cette exécution du protocole :

– Le premier message, $Init$, fait partie des connaissances initiales de l'intrus.
– Le message A, B, N_A attendu au pas S_1 est exactement le message reçu au pas précédent A_1.
– Le message $\{N_A, B\}^s_{K_{AS}}$ attendu au pas A'_1 est construit grâce à une règle préfixe, avec le message transmis par le pas précédent S_1 :

$$\left\{N_A, B, K_{AB}, \{K_{AB}, A\}^s_{K_{BS}}\right\}^s_{K_{AS}} \rightarrow_{prefix} \{N_A, B\}^s_{K_{AS}}$$

– Enfin, le message $\{N'_A - 1\}^s_{N_A}$ est construit simplement à partir du message $\{N'_A\}^s_{N_A}$ transmis par A'_1. En effet, l'intrus connaît N_A dès le premier pas A_1.

A la fin de cette exécution, l'intrus connaît le message $\{Secret\}^s_{N_A}$ transmis au dernier pas A'_2, et peut donc calculer $Secret$ puisqu'il connaît N_A.

2.4 Insécurité modulo un opérateur algébrique

Bien que reposant sur l'hypothèse de chiffrement parfait, le modèle de protocoles présenté jusqu'ici peut nous permettre de décrire certaines propriétés algébriques non triviales des protocoles cryptographiques. Le but de cette section est d'étendre le modèle de protocole et les notions d'intrus et d'attaque précédentes pour pouvoir non seulement envisager certaines propriétés algébriques des opérateurs d'encryption déjà présentés, mais également introduire de nouveaux opérateurs et leurs propriétés. Ceci sera utilisé dans les chapitres 4 à 6 pour la vérification de protocoles modulo les opérateurs xor et exponentiation, ainsi que modulo la commutation de clef privées/publiques (type RSA). Ces extensions sont relativement simples et suivent exactement le même schéma que le modèle de base.

2.4.1 Intérêt du modèle

Nous allons présenter un modèle de protocoles (bien formé) et un modèle d'intrus (normalisé) dont le but est de décrire toutes les attaques de protocoles concrets avec les opérateurs de ou exclusif bit à bit, d'exponentiation, et d'encryption commutative, pour des propriétés algébriques raisonnables, décrites à la Table 2.4. Ce modèle repose sur une hypothèse réaliste des protocoles cryptographiques concrets, que l'on défendra à la section 2.4.4. Cette hypothèse vise à réduire l'expressivité de notre modèle de protocoles, de manière à avoir la meilleur complexité possible pour le recherche d'attaques, en éliminant (uniquement !) les spécifications de protocoles inutilisables en pratique. Ainsi, notre modèle va nous permettre d'obtenir des algorithmes de décision NP (non déterministe polynomial) pour le problème de l'insécurité de protocoles cryptographiques pour les trois intrus décrits ci-dessus : $\mathcal{L}_{xor}$, $\mathcal{L}_{D.H.}$ et $\mathcal{L}_{E.C.}$. De plus, ces algorithmes sont optimaux car même sans aucun opérateur supplémentaire ni aucune propriété algébrique, le problème de l'insécurité de protocoles cryptographiques dans le modèle de Dolev-Yao est NP-difficile. Différents codages originaux de problèmes NP-difficiles seront présentés au chapitre 3 pour le modèle de base. Ils seront également utilisables avec les différents modèles d'intrus normalisé décrits ici, car un intrus $\mathcal{L}_{xor}$, $\mathcal{L}_{D.H.}$ ou $\mathcal{L}_{E.C.}$ utilisé sur un protocole sans aucun opérateur xor, produit, exponentielle ou encryption commutative n'est pas capable de décomposer plus de termes (du protocole) que l'intrus de Dolev-Yao (il peut au mieux créer des termes basés sur ces opérateurs totalement inutiles pour ce genre de protocoles).

En outre, le modèle de protocoles par rôles décrit ici ne permet que la spécification d'une seule session. Pour modéliser plusieurs sessions, en parallèle ou en séquence, on doit dupliquer les pas de protocoles correspondant, soit pour créer de nouveau rôles (sessions parallèles), soit pour

étendre des rôles existants (sessions en séquence). Dans les deux cas, il faut veiller à renommer les variables pour éviter tout confit. Cela crée une spécification de protocole plus importante que nécessaire, mais simplifie également la recherche d'attaque (on évite toute structure de contrôle des différentes sessions). De plus, en dupliquant les sessions on ne peut faire croître que polynomialement la taille de la spécification, ce qui n'a pas beaucoup d'influence dans un résultat de NP complétude.

2.4.2 Extension de l'algèbre des termes

Dans la section précédente, nous avons défini les termes et messages à l'aide d'opérateurs standard de paire et d'encryption symétrique et asymétrique). L'idée est maintenant d'ajouter de nouveaux opérateurs à cette grammaire de base. Dans toute la suite, nous aurons besoin des opérateurs suivants :

$\oplus$: L'opérateur *XOR*, appliqué bit à bit sur deux messages.
$\cdot$: L'opérateur de *produit* entre deux messages.
Exp : L'opérateur d'*exponentiation* d'un message par un produit.
$\{\_\}^{pc}_{\_}$: Un opérateur d'encryption publique/privée *commutatif* sur les clefs. (Rq : *pc* signifie Publique/privée avec Commutation)

De plus, on continue d'utiliser les opérateurs présentés à la section précédente, i.e. :

$\langle \_, \_ \rangle$: L'opérateur de concaténation (création d'une paire).
$\{\_\}^{s}_{\_}$: Un opérateur d'encryption symétrique (Rq : s pour Symétrique).
$\{\_\}^{p}_{\_}$: Un opérateur d'encryption publique/privée (*sans* commutation). (Rq : p signifie Publique/privée)

Le langage de termes construit sur ces nouveaux opérateurs est le suivant, et les messages sont comme précédemment le sous ensemble des termes sans variable :

$$
\begin{array}{rcl}
Terme & ::= & StdTerme \,|\, Xor \\
StdTerme & ::= & Var \,|\, Atomes \,|\, \langle Terme, Terme \rangle \,|\, \{Terme\}^{s}_{Terme} \,|\, \{Terme\}^{p}_{Terme} \\
 & & |\, Exp(Terme,\, Produit_{\mathbb{Z}}) \,|\, \{Terme\}^{pc}_{Produit_{\mathbb{N}}} \\
 & & \\
Xor & ::= & StdTerme \,|\, StdTerme \oplus Xor \\
Produit_{\mathbb{Z}} & ::= & Terme^{\mathbb{Z}} \,|\, Terme^{\mathbb{Z}} \cdot Produit_{\mathbb{Z}} \\
Produit_{\mathbb{N}} & ::= & Terme^{\mathbb{N}} \,|\, Terme^{\mathbb{N}} \cdot Produit_{\mathbb{N}}
\end{array}
$$

$\mathbb{Z}$: Ensemble des entiers relatifs
$\mathbb{N}$: Ensemble des entiers naturels

Dans cette grammaire, $t^z \in Terme^{\mathbb{Z}}$ représente un terme $t \in Terme$ élevé à une puissance entière $z \in \mathbb{Z}$, et z est appelé coefficient entier (ou coefficient produit). De même pour $Terme^{\mathbb{N}}$ avec $z \in \mathbb{N}$. L'ensemble des termes $Terme$ représente les termes que l'on utilisera dans la spécification d'un protocole. L'ensemble $Messages$ des messages est toujours le sous ensemble de $Terme$ des termes clos, i.e. sans variable. Il est important de remarquer qu'un produit $Produit_{\mathbb{Z}}$ ou $Produit_{\mathbb{N}}$ n'est pas un terme et ne peut apparaître que sous Exp ou $\{..\}^{pc}_{..}$. Pour cette raison, on généralise légèrement la définition de terme :

$$TermeGenerique \ ::= \ Terme \,|\, Produit_{\mathbb{Z}}$$

L'ensemble $TermeGenerique$ représente tous les termes en général, i.e. les termes $Terme$ que l'on utilisera dans la spécification des protocoles ainsi que tous les produits de termes $Produit_{\mathbf{Z}}$ ou $Produit_{\mathbf{N}}$.

L'opérateur $\oplus$ modélise l'utilisation de l'opérateur logique xor bit à bit sur deux messages ou plus. En conséquence, il sera toujours lu et écrit modulo l'associativité et la commutativité de $\oplus$, ce qui évite d'utiliser des parenthèses. Les autres propriétés de $\oplus$ sont l'existence d'un élément neutre, que l'on notera 0, tel que $a \oplus 0 = a$ pour tout terme a, ainsi que la nilpotence, i.e. $a \oplus a = 0$ pour tout terme a. 0 étant une connaissance générique, on supposera dans tout protocole qu'il appartient aux connaissances initiales de l'intrus. On veut pouvoir modéliser et vérifier des protocoles utilisant l'opérateur xor, car de nombreux protocoles concrets utilisent cet opérateur. En effet, même s'il possède beaucoup de défauts, l'opérateur xor peut être utilisé très rapidement sur de très grandes quantités de données à moindre coût, ce qui n'est pas le cas d'une encryption classique, même symétrique. Il est à noter que l'on pourra modéliser une utilisation directe de l'opérateur xor, mais que l'on ne pourra cependant pas modéliser en détail un opérateur d'encryption basé sur le xor comme DES. En effet, un tel système coupe le message à encrypter selon une mesure qui n'a en général rien à voir avec les atomes utilisés, ce que l'on ne peut pas modéliser ici. (par exemple, le message $\langle a, b\rangle$ sera d'abord découpé en trois morceaux $\langle c_1, c_2, c_3\rangle = \langle a, b\rangle$)

L'opérateur produit $\cdot$ modélise quant à lui la multiplication dans un groupe abélien. Par exemple, le produit $a^3 \cdot b^2 \cdot c^{-2}$ représente l'élément de ce groupe où $a^3 = a \cdot a \cdot a$, $b^2 = b \cdot b$, et $c^{-2} = c^{-1} \cdot c^{-1}$ avec c^{-1} l'inverse de c. En particulier, le protocole $A-GDH.2$ utilise un groupe abélien qui est un sous groupe d'ordre q du groupe multiplicatif $\mathbf{Z}_p^*$, pour p et q deux nombres premiers. Le but de cet opérateur est de modéliser facilement certaines propriétés algébriques de l'exponentielle et de l'encryption asymétrique. Comme pour l'opérateur xor, il est considéré modulo associativité et commutativité, ce qui nous permet d'éviter tout parenthésage, et modulo l'identité $t^1 = t$ pour éviter d'écrire un coefficient quand il vaut 1. Par exemple, on considère que les termes $d^1 \cdot c^{-2} \cdot (b^3 \cdot a^2)$ et $a^2 \cdot b^3 \cdot c^{-2} \cdot d$ représentent exactement les mêmes produits. De plus, on étend simplement $\cdot$ en une fonction binaire $\bullet$ sur les produits : si $M_1 = u_1^{z_1} \cdot .. \cdot u_n^{z_n} \in Produit_{\mathbf{Z}}$ et $M_2 = v_1^{z'_1} \cdot .. \cdot v_n^{z'_n} \in Produit_{\mathbf{Z}}$, alors $M_1 \bullet M_2 = u_1^{z_1} \cdot .. \cdot u_n^{z_n} \cdot v_1^{z'_1} \cdot .. \cdot v_n^{z'_n}$. Enfin, cet opérateur possède également un élément neutre noté 1, différent de l'élément neutre pour xor, appartenant lui aussi aux connaissances initiales des principaux et de l'intrus dans toute spécification de protocole.

Ce produit permet la définition de deux nouveaux opérateurs, nommés Exp et $\{..\}_{..}^{pc}$. Le premier représente l'exponentiation d'un terme par un produit de termes. Par exemple, $Exp(a, b^2 \cdot c^{-1})$ représente $a^{(b^2 \cdot c^{-1})} = \left(\left(a^b\right)^b\right)^{c^{-1}}$. Nous donnerons par la suite à cet opérateur les propriétés algébriques de l'exponentiation de Diffie-Hellman. Le second opérateur $\{..\}_{..}^{pc}$ est une version plus générale de l'encryption *asymétrique* décrite dans les sections précédentes. Le terme $\{M\}_{a^2 \cdot b^*}^{pc}$ décrit l'encryption successive du message M par les clefs a, a et b^*, à savoir $\left\{\left\{\{M\}_a^p\right\}_a^p\right\}_{b^*}^p$. Les propriétés du produit permettront ici de simuler les propriétés de l'opérateur d'encryption comme la commutation de clefs. A la différence de l'exponentielle, on ne définit par d'inverse propre à ce nouvel opérateur d'encryption. En effet, on utilisera ici l'inverse déjà définie sur les clefs atomiques (K^*, d'où des termes de la forme $\{M\}_{(K^*)^4}^{pc}$), alors que l'exponentielle utilisera une autre notion d'inverse sur n'importe quel terme (t^{-1}, d'où des termes de la forme $Exp(M, t^{-4})$. La différence principale entre ces deux notions d'inverse résidera dans les capacités de l'intrus : il pourra calculer $Exp(M, t^{-1})$ avec M et t, mais ne pourra calculer $\{M\}_{K^*}$ qu'avec M et K^* (au lieu de M et K). On utilise donc en fait deux notions relativement différentes du pro-

duit. Mais la position d'un opérateur produit dans un terme (i.e. sous une exponentielle ou sous une encryption) détermine sans équivoque de quel sorte de produit il s'agit[5]. Enfin, on conserve l'ancien opérateur d'encryption asymétrique $\{..\}^p_{..}$, car tous les opérateurs d'encryption publique/privée ne permettent pas cette commutation : un protocole pourrait très bien utiliser deux types d'opérateurs asymétriques, l'un avec et l'autre sans commutation.

On répartit les différents termes dans deux catégories :

Définition 2.4.2.1 *Termes standards et non standards.*

- L'ensemble des termes standards *est le sous ensemble de $Terme$ des termes sans symbole $\oplus$ en tête, i.e. $StdTerme$.*

- L'ensemble des termes non standards *est le sous ensemble de $TermeGenerique$ des termes ayant le symbole $\oplus$ ou $\cdot$ en tête, i.e. $Xor \cup Produit_{\mathbf{Z}} = TermeGenerique \setminus StdTerme$.*

L'ensemble des termes standards ou non standards est $TermeGenerique$. On peut à présent étendre la notion de sous termes à ces nouveaux opérateurs :

Définition 2.4.2.2 *Sous terme.*

Pour tout terme générique $t \in TermeGenerique$, on note $STermes(t)$ l'ensemble des sous termes de t défini récursivement par :

- $STermes(t) = \{t\}$ *si* $t \in Atomes$ *ou* $t \in Var$
- $STermes(t) = \{t\} \cup STermes(u) \cup STermes(v)$ *si* $t = \langle u, v\rangle$ *ou* $t = \{u\}^s_v$ *ou* $t = \{u\}^p_v$
- $STermes(t) = \{t\} \cup STermes(u) \cup \bigcup_i STermes(v_i)$ *si* $t = Exp(u, v_1^{z_1} \cdot .. \cdot v_n^{z_n})$ *ou* $t = \{u\}^{pc}_{v_1^{p_1}\cdot ... \cdot v_n^{p_n}}$
- $STermes(t) = \{t\} \cup \bigcup_i STermes(v_i)$ *si* $t = v_1^{z_1} \cdot .. \cdot v_n^{z_n}$ *ou* $t = v_1 \oplus .. \oplus v_n$

L'ensemble des sous termes propres de $t \in TermeGenerique$ est $STermes(t) \setminus \{t\}$.

On peut remarquer plusieurs choses dans cette définition. Par exemple, $a \oplus b$ n'est pas un sous terme de $a \oplus b \oplus c$. Ensuite, dans cette définition un produit n'est jamais sous terme d'un terme standard ou sous terme propre d'un produit. Ceci est dû au fait que les produits sont toujours liés à un opérateur Exp ou $\{..\}^{pc}$. Par exemple, $a^2 \cdot b$ n'est ni un sous terme de $Exp(M, a^2 \cdot b)$, ni un sous terme de $\{M\}^{pc}_{a^2 \cdot b}$. En revanche, un terme non standard xor (i.e. de symbole de tête xor) peut être sous terme d'un terme standard. On définit une notion de sous terme étendu pour inclure au besoin ces produits dans la notion de sous terme :

Définition 2.4.2.3 *Sous termes étendus.*

Pour tout terme générique $t \in TermeGenerique$, on note $STermes^+(t)$ l'ensemble des sous termes étendu de t, défini par :

$$STermes^+(t) ::= STermes(t) \cup \{M \mid Exp(u, M) \in STermes(t) \text{ ou } \{u\}^{pc}_M \in STermes(t)\}$$

Dans cette définition, les produits sont traités comme les xor : ils sont comptés parmi les sous termes étendus, mais jamais décomposés. Par exemple, $a^2 \cdot b$ n'est pas un sous terme de $a^2 \cdot b \cdot c^{-1}$. Dans la section précédente, nous avions défini la taille des termes par le nombre de sous termes, ce qui correspondait (modulo un coefficient linéaire) à la taille de la représentation DAG d'un terme. Cette définition doit être étendue, car nous perdrions sinon la taille des coefficients utilisés dans les produits. En particulier, deux termes ne différant que par les valeurs de leurs coefficients,

[5] Deux opérateurs produit différents n'auraient fait que surcharger inutilement la syntaxe, d'autant plus que les différences apparaîtront dans des capacités de l'intrus, et non dans les propriétés algébriques de l'opérateur produit.

comme $a^2 \cdot b^3$ et $a \cdot b^{2003}$, auraient la même taille. Nous définissons donc deux notions de taille de terme, l'une comptant le nombre de sous termes distincts et l'autre comptant les coefficients. On obtient :

Définition 2.4.2.4 *Tailles d'un ensemble de termes génériques.*

Soit E un ensemble de termes génériques. La taille DAG de E, notée $|E|_{dag}$, est le nombre de sous termes distincts de E non réduits à 0, 1 ou $Init$, et la taille des exposants de E, notée $|E|_{exp}$, est la somme des tailles des représentations binaires des coefficients des produits sous termes étendus de E, i.e. :

$$\begin{array}{rcl} |E|_{dag} & = & \#STermes(E) \\ |E|_{exp} & = & \sum_{t_1^{z_1} \cdot \ldots \cdot t_n^{z_n} \in STermes^+(E)} \sum_{i \in \{1..n\}} |z_i| \end{array}$$

avec $|n|$ le nombre de bits de la représentation binaire[6] *de n, entier naturel ou relatif.*

On remarque qu'avec ces définitions, les termes a et $a \oplus a \oplus a$ ont les mêmes tailles. Ceci sera cohérent avec la définition de normalisation de la section suivante, car ces deux termes représentent en fait le même message (a n'est pas dupliqué). Par ailleurs, on a défini la taille DAG grâce à la notion de sous termes, mais on aurait pu indifféremment utiliser la notion de sous terme étendu. En effet :

Lemme 2.4.2.5 *Pour tout terme générique t, on a $\#STermes^+(t) < 2.\#STermes(t)$.*

La preuve est triviale, puisque l'on a au plus autant de produits sous termes étendus de t que d'exponentielles ou d'encryptions commutatives sous terme de t. On peut alors définir la taille (complète) d'un terme avec :

$$\forall t \in TermeGenerique, \quad \|t\| = |t|_{dag} + |t|_{exp}$$

Dans le cas de termes sans xor ni produit, cette notion de taille coïncide avec la taille Dag de la section précédente. Par extension, ces trois notions de taille sont également définies sur des ensembles de termes, et sur des substitutions avec :

$$\|\sigma\| = \sum_{x \in Var} \|\sigma(x)\|$$

et de même pour $|\sigma|_{dag}$ et $|\sigma|_{exp}$. Pour finir de caractériser les différents éléments construisant un terme, on définit une notion de *facteurs*. Intuitivement, les facteurs d'un terme t sont ses plus grand sous termes standards.

Définition 2.4.2.6 *Facteurs d'un terme.*

Soit $t \in TermeGenerique$. On définit récursivement l'ensemble $Facteur(t)$ des facteurs de t par :

- *$Facteur(t) = \{t\}$ si t est un terme standard sans Exp ni $\{..\}^{pc}_{..}$ en tête.*
- *$Facteur(t) = \{u, v_1, .., v_n\}$ si $t = Exp(u, v_1^{z_1} \cdot \ldots \cdot v_n^{z_n}\}$ ou $t = \{u\}^{pc}_{v_1^{z_1} \cdot \ldots \cdot v_n^{z_n}}$ pour tous entiers naturels ou relatifs $\{z_i\}$.*
- *$Facteur(t) = v_1, .., v_n$ si $t = v_1^{z_1} \cdot .. \cdot v_n^{z_n}$ ou $t = v_1^{z_1} \oplus .. \oplus v_n^{z_n}$.*

[6] Le signe de $n \in \boldsymbol{Z}$ ne compte que pour un bit dans sa représentation binaire.

Les facteurs d'un terme (générique) sont toujours des termes standards, car par hypothèse les notations $v_1^{z_1} \cdot .. \cdot v_n^{z_n}$ et $v_1^{z_1} \oplus .. \oplus v_n^{z_n}$ désignent des termes v_i sans xor ni produit en tête, donc standards.

La notion de *substitution* est toujours valide (sur les termes), et simplement étendue aux termes génériques. On lui adjoint une notion de remplacement :

Définition 2.4.2.7 *Remplacement.*

Étant donnés un terme standard u et un terme générique v, le remplacement $\delta = [u \leftarrow v]$ de u par v est une fonction qui a tout terme générique t associe le terme générique $t[u \leftarrow v]$ obtenu en remplaçant dans t toutes les occurrences de u par v.

Dans cette définition, on utilise au besoin implicitement l'associativité des opérateurs $\oplus$ et $\cdot$. Ainsi, si $t = a \oplus b$ et $\delta = [b \leftarrow c \oplus d]$, alors $t\delta = a \oplus c \oplus d$. De même pour le produit $\cdot$. De plus, comme on ne remplace qu'un seul terme standard à la fois, le résultat d'un remplacement est unique et cette définition est donc correcte. La composition d'une substitution σ et d'un remplacement δ donne une nouvelle substitution $\sigma\delta$ associant à chaque $x \in Var$ la valeur $\sigma(x)\delta$. On peut remarquer que l'ordre d'application est important, puisqu'en général on a $t(\sigma\delta) \neq (t\sigma)\delta$. Par exemple, avec $t = \langle x, b\rangle$, $\sigma(x) = a$, et $\delta = [\langle a, b\rangle \leftarrow c]$, on a $t(\sigma\delta) = \langle a, b\rangle$ et $(t\sigma)\delta = c$. Les remplacements ont quelques propriétés de base :

Proposition 2.4.2.8 *Propriétés de base des remplacements.*

Soit $\delta = [u \leftarrow v]$ un remplacement, t_i des termes standards, et z_i des entiers relatifs. On a :

- $(t_1^{z_1} \cdot ... \cdot t_n^{z_n})\delta = (t_1\delta)^{z_1} \cdot ... \cdot (t_n\delta)^{z_n}$ *et* $(t_1 \oplus ... \oplus t_n)\delta = t_1\delta \oplus ... \oplus t_n\delta$
- *Si* $u \neq Exp(t_0, t_1^{z_1} \cdot .. \cdot t_n^{z_n})$, *alors* $Exp(t_0, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\delta = Exp(t_0\delta, (t_1\delta)^{z_1} \cdot .. \cdot (t_n\delta)^{z_n})$
- *Si* $u \neq \{t_0\}_{t_1 \oplus .. \oplus t_n}$, *alors* $\{t_0\}^{pc}_{t_1 \oplus .. \oplus t_n}\delta = \{t_0\delta\}_{t_1\delta \oplus .. \oplus t_n\delta}$
- *Pour tout* $s \in Terme$, $t \in TermeGenerique$, *et* $\alpha \in Atomes \cup Var$, *on a* $|t[s \leftarrow \alpha]|_{dag} \leq |t|_{dag}$.

Le premier point est une simple application de la définition de remplacement, puisque celui-ci remplace des termes et que pour tout i, $t_i^{z_i}$ n'est pas un terme mais un produit. Les second et troisième points sont également de simples applications des définitions : δ ne remplace qu'un terme et ce n'est pas u. Enfin, le dernier point s'obtient quant à lui par une simple récurrence sur la structure du terme t : c'est vrai si $s = t$, et il suffit de suivre la définition de sous terme.

2.4.3 Normalisation modulo un opérateur algébrique

Ajouter de nouveaux opérateurs à la notion de termes et aux différentes définitions qui en découlent n'aurait pas beaucoup d'intérêt si l'on ne pouvait pas les équiper avec des propriétés algébriques intéressantes. Le but de cette section est de définir une notion de normalisation sur les termes (standard ou non) modulo un ensemble de propriétés algébriques sur $\oplus$, $\cdot$, Exp et $\{..\}^{pc}_{..}$. On rappelle que les termes sont toujours écrits modulo l'associativité et la commutativité des opérateurs $\oplus$ et $\cdot$. Sur ces propriétés de base on ajoute toutes les propriétés décrites par la Table 2.4, avec t un terme standard, M_1 et M_2 deux produits (dans $Produit_{\mathbb{Z}}$ ou $Produit_{\mathbb{N}}$), k un atome dans $Clefs$, et z, z' deux entiers relatifs non nuls. On a successivement les propriétés de l'opérateur xor, i.e. la nilpotence ($t \oplus t = 0$) et l'élément neutre ($t \oplus 0 = t$), puis les propriétés du produit $\cdot$, i.e. essentiellement les propriétés de groupe abélien. Il est important de remarquer que la dernière propriété de $\cdot$ est bien valable dans $Produit_{\mathbb{N}}$ (en plus de $Produit_{\mathbb{Z}}$), puisque

	Élément neutre	**Simplification de termes**
xor	$t \oplus 0 = t$	$t \oplus t = 0$
produit	$t \cdot 1 = t$	$t^z \cdot t^{z'} = t^{z+z'}$ (avec + l'addition sur les entiers)

	Exposant unitaire	**Réduction à un élément neutre**
produit	$t^1 = t$	$t^0 = 1 \qquad 1^z = 1$

	Empilement trivial	**Réduction d'empilements multiples**
exponentielle	$Exp(t,\, 1) = t$	$Exp(Exp(t,\, M_1),\, M_2) = Exp(t,\, M_1 \bullet M_2)$
encryption	$\{t\}_1^{pc} = t$	$\left\{\{t\}_{M_1}^{pc}\right\}_{M_2}^{pc} = \{t\}_{M_1 \bullet M_2}^{pc}$

	Simplification de l'inverse
produit sous l'encryption	$\{t\}_{(k^a \cdot (k^*)^b) \bullet M_1}^{pc} = \{t\}_{k^{a-b} \bullet M_1}^{pc}$ si $a \geq b$
	$\{t\}_{(k^a \cdot (k^*)^b) \bullet M_1}^{pc} = \{t\}_{(k^*)^{b-a} \bullet M_1}^{pc}$ si $b < a$

Rappel : si $M_1 = u_1^{z_1} \cdot .. \cdot u_n^{z_n} \in Produit_{\boldsymbol{Z}}$ et $M_2 = v_1^{z'_1} \cdot .. \cdot v_n^{z'_n} \in Produit_{\boldsymbol{Z}}$, alors $M_1 \bullet M_2 = u_1^{z_1} \cdot .. \cdot u_n^{z_n} \cdot v_1^{z'_1} \cdot .. \cdot v_n^{z'_n}$

Remarque : La notation 1 représente l'atome 1 dans un terme ou l'entier 1 dans un coefficient entier.

TAB. 2.4: Propriétés algébriques des opérateurs $\oplus$, $\cdot$, Exp et $\{..\}_{..}^{pc}$

$z + z' \geq 0$ si $z \geq 0$ et $z' \geq 0$. On décrit ensuite des propriétés de l'exponentiation, puis les propriétés de l'opérateur d'encryption commutative $\{..\}^{pc}_{..}$. Ceci définit implicitement une équivalence entre termes génériques, mais pour la recherche d'attaque on aura besoin d'avoir des représentants de ces classes. D'où la définition :

Définition 2.4.3.1 *Normalisation et Forme normalisée.*

Une étape de normalisation *est l'application à un terme générique de l'une des identités de la Table 2.4, de gauche à droite.*

Étant donné un terme générique $t \in TermeGenerique$, la forme normalisée *de t, notée $\ulcorner t \urcorner$, est obtenue en itérant les étapes de normalisation sur t et ses sous termes étendus jusqu'à l'obtention d'un terme auquel ne s'applique aucune règle.*

Pour que cette définition soit utile, on doit immédiatement constater la confluence et la terminaison des étapes de normalisation. Détaillons les points importants d'une telle preuve.

Pour cela, nous allons séparer l'opérateur produit en deux opérateurs produit_exp (noté $\cdot$ comme avant) et produit_enc (noté $\odot$) selon qu'il est utilisé sous une exponentielle ou sous une encryption commutative. Nous ne faisons cette distinction que pour justifier la confluence et la terminaison de la normalisation. De plus, nous pouvons faire cette distinction car syntaxiquement, l'opérateur produit ne peut être utilisé que sous une exponentielle ou sous une encryption commutative. Il y a naturellement une bisimulation forte entre la normalisation avec un seul opérateur produit et la normalisation avec produit_exp et produit_enc. Il nous suffit donc de prouver la confluence et la normalisation de cette seconde normalisation. De plus, on remarque immédiatement qu'avec cette séparation entre produit_exp et produit_enc, les deux dernières règles de normalisation (n°12 et 13, simplification du produit sous une encryption commutative) peuvent être vues comme deux règles de $(k^a \odot (k^*)^b)$ vers k^{a-b} ou $(k^*)^{b-a}$, selon $a \geq b$ ou $a < b$, étendant ainsi les règles de normalisation de produit_enc. On constate qu'il n'existe aucune interaction entre les xor et les deux types de produits, de même qu'entre les produits sous une exponentielle et les produits sous une encryption commutative. De plus, les règles de normalisation du xor, de produit_exp, et de produit_enc, i.e. les sept premières règles du tableau plus les deux nouvelles règles précédentes, sont tout à fait classiques. On sait déjà qu'elles sont confluentes et qu'elles terminent, même modulo l'associativité et la commutativité du xor, de produit_exp, et de produit_enc. Il nous reste donc à regarder les paires critiques utilisant au moins une règle d'exponentielle ou d'encryption commutative. Commençons par l'exponentielle seule. Entre $Exp(t, 1) \rightarrow t$ et $Exp(Exp(t, M_1), M_2) \rightarrow Exp(t, M_1 \bullet M_2)$, on a $Exp(t, 1 \bullet M_2) = Exp(t, M_2)$ et $Exp(t, M_1 \bullet 1) = Exp(t, M_1)$ ce qui est naturellement confluent. En outre, les autres paires critiques sont triviales, en particulier entre l'exponentielle et produit_exp (ou xor, ou produit_enc) modulo l'associativité et la commutativité de xor, produit_exp, et produit_enc. De même avec l'opérateur d'encryption commutative. Au final, la normalisation (avec un ou deux opérateurs produit) est confluente. De plus, on constate aisément la terminaison de la normalisation en remarquant que chaque règle élimine un opérateur. En résumé :

Théorème 2.4.3.2 *Unicité de la forme normale.*

Tout terme admet une forme normalisée, et elle est unique.

Corollaire 2.4.3.3 *Pour tous termes génériques t_1 et t_2, $\ulcorner t_1 \urcorner = \ulcorner t_2 \urcorner$ ssi $t_1 = t_2$ modulo les propriétés algébriques de la Table 2.4.*

Pour illustrer la normalisation, nous donnons quelques exemples :

Exemple 2.4.3.4 *Exemples de normalisations.*
Si $\{t, u\} \subset Terme$ *et* $\{a, b, c\} \subset Clefs$*, on a :*

$$\textit{Simplification du xor :} \quad \ulcorner a \oplus b \oplus a \oplus c \urcorner = b \oplus c$$

$$\textit{Simplification du produit :} \quad \ulcorner a^2 \cdot b^1 \cdot b^{-1} \urcorner = a^2$$

$$\textit{Réduction de l'encryption :} \quad \ulcorner \left\{ \{t\}^{pc}_{a^3 \cdot (c^*)^6 \cdot (a^*)^3} \right\}^{pc}_{c^6} \urcorner = t$$

$$\ulcorner \left\{ \{t\}^{pc}_{u \cdot c \cdot b} \right\}_{c^* \cdot (b^*)^2} \urcorner = \{t\}_{u \cdot (b^*)^1}$$

$$\textit{Réduction de l'exponentielle :} \quad \ulcorner Exp(Exp(t,\, a^3 \cdot u^{-6} \cdot a^{-3}),\, u^6) \urcorner = t$$

$$\ulcorner Exp(Exp(t,\, u \cdot c),\, c^* \cdot u^{-2} \urcorner = Exp(t,\, u^{-1} \cdot c \cdot c^*)$$

On ne considère ici aucune propriété de distributivité pour les paires. En particulier, on a $\ulcorner \langle a, b \rangle \cdot \langle a^{-1}, c \rangle \urcorner \neq \langle 1, b \cdot c \rangle$, et de même avec l'opérateur xor. On dit qu'un terme (générique) t est *normalisé* quand $\ulcorner t \urcorner = t$.

Définition 2.4.3.5 *Équivalence de termes.*
Deux termes génériques t_1 *et* t_2 *sont dits* équivalents *quand* $\ulcorner t_1 \urcorner = \ulcorner t_2 \urcorner$.

Ceci correspond aux classes d'équivalence modulo les propriétés de la Table 2.4, plus l'associativité et la commutativité de $\oplus$ et $\cdot$. On définit une notion de pré terme utilisant la normalisation :

Définition 2.4.3.6 *Pré termes.*
Soient deux termes t *et* t'*, et une substitution close* θ*. On dit que* t *est un pré terme de* t' *selon* θ*, noté* $t \sqsubseteq_\theta t'$*, quand* t *et* t' *sont des termes standards,* t *n'est pas une variable, et* $\ulcorner t\theta \urcorner = t'$.

Ceci correspond à l'idée intuitive que le terme t est la partie supérieure du terme t' modulo la normalisation, la partie inférieure étant décrite par θ. Enfin, la notion de forme normale s'étend de manière évidente aux ensembles de termes et aux substitutions. De plus, elle possède quelques propriétés de base intéressantes pour la recherche d'attaque, comme la décroissance en taille et la stabilité par substitution. Formellement :

Lemme 2.4.3.7 *Propriétés de la normalisation.*
Soient t *et* t' *deux termes génériques, et* σ *une substitution quelconque. On a :*

- $\left|\ulcorner t \urcorner\right|_{dag} \leq |t|_{dag}$,
- $\left|\ulcorner t \urcorner\right|_{exp} \leq |t|_{exp}$,
- $\left\|\ulcorner t \urcorner\right\| \leq \|t\|$, *et*
- $\ulcorner t\sigma \urcorner = \ulcorner \ulcorner t \urcorner \sigma \urcorner = \ulcorner t \ulcorner \sigma \urcorner \urcorner = \ulcorner \ulcorner t \urcorner \ulcorner \sigma \urcorner \urcorner$.

Les trois premiers points sont des conséquences des notions de taille. En effet, la normalisation ne fait qu'éliminer des sous termes et sommer des coefficients. Le dernier point est en fait une conséquence de la confluence de la normalisation. En effet, il suffit de remarquer que l'on peut normaliser dans $t\sigma$ tous les sous termes obtenus grâce à la substitution σ (indépendamment des autres sous termes de $t\sigma$). On peut donc atteindre $t\ulcorner\sigma\urcorner$ à partir de $t\sigma$ grâce aux règles de

normalisation, et ces deux termes ont donc la même forme normale : $\ulcorner t\sigma \urcorner = \ulcorner t \ulcorner \sigma \urcorner \urcorner$. Le même raisonnement donne $\ulcorner t\sigma \urcorner = \ulcorner\urcorner t \urcorner \sigma \urcorner$ et $\ulcorner t\sigma \urcorner = \ulcorner\urcorner t \urcorner \ulcorner \sigma \urcorner \urcorner$ en remarquant que si l'on peut appliquer une règle de normalisation à un terme t, alors on peut appliquer la même règle (à la même position) sur $t\sigma$.

2.4.4 Protocole normalisé et bien formé.

On veut un modèle de protocoles cryptographiques où ni les principaux ni l'intrus ne pourront différencier deux termes appartenant à la même classe d'équivalence. Intuitivement, quand un principal ayant les connaissances σ exécute un pas $R \Rightarrow S$ où il reçoit le message m, il commence par vérifier si m et R correspondent, i.e. il existe une substitution σ telle que $\ulcorner m \urcorner = \ulcorner R\sigma \urcorner$ (puisqu'il ne peut différencier m de $\ulcorner m \urcorner$). Si c'est le cas, le message $\ulcorner S\sigma \urcorner$ sera envoyé à l'intrus. En conséquence, on supposera toujours que tous les messages échangés entre les principaux et l'intrus (ou par le biais de l'intrus) seront normalisés, ce qui nous donne :

Définition 2.4.4.1 *Pas de protocole normalisé.*

Un pas de protocole normalisé est une paire de termes $R, S \in Terme$ normalisés*, i.e. $\ulcorner R \urcorner = R$ et $\ulcorner S \urcorner = S$, noté $R \Rightarrow S$.*

Ceci est strictement équivalent à la définition de pas de protocole, car $\ulcorner R\sigma \urcorner = \ulcorner\urcorner R \urcorner \sigma \urcorner$, et donc le principal ne fait aucune différence entre R et $\ulcorner R \urcorner$. De même pour $\ulcorner S\sigma \urcorner = \ulcorner\urcorner S \urcorner \sigma \urcorner$. L'exécution d'un pas de protocole normalisé reste la même que pour un pas de protocole classique, et on définit la notion de *protocole normalisé* à partir de la notion de protocole en n'utilisant que des pas de protocole normalisés. A partir de maintenant, tout protocole sera au moins un protocole normalisé. Cette notion de protocole est cependant un peu trop générale pour modéliser précisément des protocoles concrets. En effet, elle permet de spécifier des protocoles cryptographiques ne pouvant pas être exécutés par des principaux concrets. En voici un exemple, à un seul pas, basé sur l'opérateur xor :

$$A \ : \ x \oplus y \Rightarrow \langle x, y \rangle$$

avec x et y deux variables. En effet, on imagine mal comment un principal réel pourrait calculer les deux termes x et y en ne connaissant que leur xor. Pourtant, ceci est permis par ce modèle de protocole. Le même genre de problème se pose pour l'opérateur produit dans une exponentielle ou une encryption commutative. Pour cette raison, nous donnons une définition un peu plus restreinte de protocole, que nous justifierons ensuite :

Définition 2.4.4.2 *Protocole bien formé. (Hypothèse réaliste)*

Un protocole bien formé *est un triplet $(\{R_\iota \Rightarrow S_\iota \mid \iota \in \mathcal{I}\}, <_\mathcal{I}, S)$ où S est un ensemble fini de messages, $<_\mathcal{I}$ est un ordre partiel sur $\mathcal{I}$, et pour tout $\iota \in \mathcal{I}$, $R_\iota \Rightarrow S_\iota$ est un pas de protocole* normalisé *tel que :*

- *pour tout $x \in Var(S_\iota)$, il existe $\iota' \leq_\mathcal{I} \iota$ tel que $x \in Var(R_\iota)$, avec $\leq_\mathcal{I}$ la clôture réflexive de $<_\mathcal{I}$, et*
- *pour tout sous terme $t_1 \oplus ... \oplus t_n$, $Exp(t_1, t_2^{z_2} \cdot ... t_n^{z_n})$ ou $\{t_1\}^{pc}_{t_2^{p_2} \cdot ... t_n^{p_n}}$ de R_ι, il existe $j \in \{1..n\}$ tel que $Var(t_i) \subseteq \bigcup_{\iota' <_\mathcal{I} \iota} Var(R_{\iota'})$ pour tout $i \in \{1..n\} \backslash \{j\}$.*

Le premier point vient de la définition de protocole : toute variable utilisée par un principal doit avoir été précédemment reçue. Le second point par contre est novateur. Il impose à tout principal décomposant un sous terme de R_ι de type xor, exponentielle ou encryption commutative, d'avoir

au plus un degré de liberté, i.e. au plus un sous terme propre inconnu. Si ce n'est pas le cas, alors on sait qu'en pratique le principal ne sera pas capable de décomposer ce sous terme, et une bonne spécification devrait utiliser une simple variable à la place. Considérons par exemple le protocole suivant à un seul principal A, d'ordre $A_1 <_{\mathcal{I}} A_2$ et de variables x, y, z :

$$\begin{array}{lrcl} A_1 : & \{x\}_a^s \oplus \{y\}_a^s & \Rightarrow & a \\ A_2 : & x & \Rightarrow & y \end{array}$$

On peut argumenter que ce protocole "mal formé" pourrait (presque) être exécuté en pratique : le principal stoque sans le décomposer le message reçu au premier pas, puis utilise x et a au second pas pour retrouver y. Cependant, cette spécification laisse penser que le principal, même s'il ne peut calculer x ou y, peut vérifier que le message reçu est bien construit avec l'opérateur xor, ce qui n'est pas le cas. L'intrus peut envoyer n'importe quel atome au premier pas de manière à connaître a (si $a = Secret$, par exemple) et stopper l'exécution sans permettre au principal d'effectuer la vérification au second pas. Pour cette raison, on préfère la spécification suivante de ce protocole :

$$\begin{array}{lcccl} A : & z & \Rightarrow & a & \\ B : & x & \Rightarrow & y & [z = \{x\}_a^s \oplus \{y\}_a^s] \end{array}$$

avec $[z = \{x\}_a^s \oplus \{y\}_a^s]$ l'unification de la connaissance z avec le terme $\{x\}_a^s \oplus \{y\}_a^s$. Nous verrons à la section 2.5.1 comment transformer ce genre de protocole en un protocole bien formé. D'autres travaux ne limitent pas les spécifications de protocole de la sorte. Par exemple, dans [34] H.Comon et V.Shmatikov obtiennent que l'insécurité des protocoles avec l'opérateur xor (uniquement) est DEXPTIME, sans l'hypothèse de protocole bien formé. Ici, cette hypothèse réaliste va nous permettre d'obtenir une procédure de décision NP pour ce même problème, mais pour des protocoles bien formés. Ce problème est NP-difficile même dans le cas de base, i.e. sans aucun opérateur $\oplus$, $\cdot$, Exp ou $\{..\}_{..}^{pc}$, donc la complexité est optimale dans notre cadre. Pour les cas plus généraux, il n'existe pas de résultat analogue.

2.4.5 Intrus étendus

Nous avons présenté à la section précédente une modélisation de protocole cryptographique prenant en compte le fait que deux termes dans une même classe d'équivalence sont indiscernables par les principaux. Nous avons besoin de la même propriété pour l'intrus. Ainsi, les règles d'intrus devront porter sur des termes normalisés, et créer des termes normalisés. De cette façon, on interdit à l'intrus toute règle de déduction de connaissance pouvant lui permettre de différencier deux termes identiques modulo les propriétés algébriques de la Table 2.4. De plus, nous répartissons les règles d'intrus en deux catégories : les règles de décomposition, dont l'idée générale est de ne pas créer de nouveaux sous termes, et les règles de composition, dont l'idée générale est de construire un terme en n'ajoutant qu'un seul nouvel opérateur. Toutes les règles d'intrus nécessaires à l'utilisation des nouveaux opérateurs $\oplus$, $\cdot$, Exp et $\{..\}_{..}^{pc}$, ainsi que les règles d'intrus plus exotiques comme l'intrus préfixe, seront des règles d'intrus normalisé formellement définies comme ceci :

Définition 2.4.5.1 *Règles d'intrus normalisé.*

Une règle d'intrus normalisé de décomposition est une règle $M \rightarrow t$, avec M, t normalisés et t sous terme de M.

Une règle d'intrus normalisé de composition est une règle $M \rightarrow t$, avec M, t normalisés et telle que tout sous terme propre de t soit un sous terme de M.

Règles de décomposition

$$L_{xd}(\ulcorner t_1 \oplus .. \oplus t_n \urcorner) : \quad t_1, .., t_n \rightarrow \ulcorner t_1 \oplus .. \oplus t_n \urcorner$$

si $\ulcorner t_1 \oplus .. \oplus t_n \urcorner$ est standard.

Règles de composition

$$L_{xc}(\ulcorner t_1 \oplus .. \oplus t_n \urcorner) : \quad t_1, .., t_n \rightarrow \ulcorner t_1 \oplus .. \oplus t_n \urcorner$$

si $\ulcorner t_1 \oplus .. \oplus t_n \urcorner$ est non standard.

Tab. 2.5: Règles de l'intrus XOR.

L'ensemble des règles d'intrus normalisé de décomposition et de composition forme l'ensemble des *règles d'intrus normalisé*. Il est important de remarquer que la plupart des règles d'intrus définies jusqu'ici sont des règles d'intrus normalisé. En particulier, les règles d'intrus de Dolev-Yao sont clairement des règles d'intrus normalisé : toute règle L_d extrait un sous terme du membre de gauche (décomposition), et toute les règle L_c crée un nouveau terme par ajout d'un opérateur (pair, encryption symétrique ou asymétrique). De la même manière, les règles de l'intrus préfixe sont des règles d'intrus normalisé de composition. On déduit de ceci une nouvelle définition d'intrus :

Définition 2.4.5.2 *Intrus normalisé.*

Un intrus normalisé (ou environnement hostile normalisant) est un intrus disposant d'un ensemble de règles d'intrus normalisé.

De la définition de règle d'intrus normalisé, on déduit immédiatement :

Proposition 2.4.5.3 *Pour tout intrus normalisé $\mathcal{L}$ et tout ensemble de termes normalisés E, on a $forge_{\mathcal{L}}(E) = \ulcorner forge_{\mathcal{L}}(E) \urcorner$.*

En conséquence, à partir de maintenant nous n'utiliserons la notation $t \in forge(E)$ que sur des termes normalisés, i.e. t normalisé et E ensemble de termes normalisés.

On a vu que $\mathcal{L}_{DY}$ et $\mathcal{L}_{Prefixe}$ sont des intrus normalisés. Mais ils ne suffisent évidement pas pour exploiter pleinement les opérateurs xor, produit, exponentielle et encryption commutative que l'on a introduit. On va donc définir plusieurs instances d'intrus permettant d'exploiter ces opérateurs. Tout d'abord, on veut utiliser l'opérateur xor. Pour cela, on définit les règles d'intrus $L_{xd} = \bigcup_t L_{xd}(t)$ et $L_{xc} = \bigcup_t L_{xc}(t)$ de la Table 2.5, avec $\{t_i\}$ termes standards.

Cette répartition entre règles de décomposition et règles de composition est dû au fait que si $\ulcorner t_1 \oplus .. \oplus t_n \urcorner$ est standard, alors il est nécessairement sous terme de t_1, .., t_n, et s'il n'est pas standard ses sous termes propres forment un sous ensemble de $\{t_1, .., t_n\}$. En effet, comme tous les termes t_i sont normalisés et standards par hypothèse, soit il existe $i \in \{1..n\}$ tel que $\ulcorner t_1 \oplus .. \oplus t_n \urcorner = t_i$, soit il existe $\{t'_j\} \subseteq \{t_i\}$ tel que $\ulcorner t_1 \oplus .. \oplus t_n \urcorner = t'_1 \oplus .. \oplus t'_p$. Ainsi, L_{xd} est un

Règles de décomposition

$$L_{DHd}(\ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n}\urcorner) : \quad t_0, ..., t_n \;\rightarrow\; \ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner$$
$$\text{si} \quad \ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner \neq Exp(...)$$

Règles de composition

$$L_{DHc}(\ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n}\urcorner) : \quad t_0, ..., t_n \;\rightarrow\; \ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner$$
$$\text{si} \quad \ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner = Exp(...)$$

TAB. 2.6: Règles de l'intrus DH

ensemble de règles d'intrus normalisé de décomposition, et L_{xc} est un ensemble de règles d'intrus normalisé de composition. L'utilité de cette répartition en deux groupes de règles apparaîtra plus clairement lors de la recherche d'attaques, au chapitre 4. Ces règles d'intrus permettent de construire l'intrus xor :

Définition 2.4.5.4 *Intrus xor.*

L'intrus xor est l'intrus disposant de l'ensemble $\mathcal{L}_{xor}$ des règles d'intrus normalisé $L_{D.Y.} \cup L_{xd} \cup L_{xc}$.

Définissons à présent les règles d'intrus utilisées pour l'opérateur Exp. Il est important de remarquer que comme l'intrus ne manipule que des termes, et comme un produit n'est pas un terme (c'est un terme générique, il n'est jamais utilisé seul), les règles que l'on va définir n'utiliseront le produit qu'à l'intérieur d'une exponentielle. C'est une hypothèse assez importante, que nécessite au moins une petite justification. Définir un intrus travaillant sur des termes génériques, i.e. pouvant calculer des produits hors d'une exponentielle, n'a réellement d'intérêt que si l'on considère des interactions entre le produit et les autres opérateurs. Typiquement, cela pourrait être une règle de normalisation de la forme $Exp(t,\, M_1) \cdot Exp(t,\, M_2) = Exp(t,\, M_1 + M_2)$, avec $+$ un opérateur d'addition sur des termes (ou les produits). Mais on ne peut pas vérifier de manière exhaustive un tel problème, car c'est au moins aussi difficile que l'unification modulo addition et produit (qui est indécidable). Limiter l'intrus aux termes nous permet d'éviter de telles interactions. Ainsi, on ne permet à l'intrus de créer un produit que pour l'utiliser dans une exponentielle. Ce sera d'ailleurs la même chose pour l'encryption commutative. On définit donc les règles d'intrus $L_{DHd} = \bigcup_t L_{DHd}(t)$ et $L_{DHc} = \bigcup_t L_{DHc}(t)$ à partir de la Table 2.6, avec $\forall i$, $t_i \in Terme$ et $z_i \in \boldsymbol{Z}^*$.

Les règles de la Table 2.6 sont réparties entre L_{DHd} et L_{DHc} sensiblement pour les mêmes raisons que dans le cas du xor. Ainsi, si $\ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner \neq Exp(...)$ alors nécessairement $\ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner = t_0$ si $t_0 \neq Exp(...)$ ou $\ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner = t_0'$ si $t_0 = Exp(t_0',\, ...)$. Cela prouve que les règles $L_{D.H.d}$ sont des règles d'intrus normalisé de décomposition. De même, si $\ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner = Exp(...)$, alors nécessairement $\ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner = Exp(t_0,\, u_1^{z_1'} \cdot .. \cdot u_p^{z_p'})$ avec $\{u_j\} \subseteq \{t_i\}$ si $t_0 \neq Exp(...)$, ou $\ulcorner Exp(t_0,\, t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner = Exp(t_0',\, u_1^{z_1'} \cdot .. \cdot u_p^{z_p'})$ avec

Règles de décomposition

$$L_{ECd}(\ulcorner\{t_0\}^{pc}_{t_1^{z_1}\cdot\ldots\cdot t_n^{z_n}}\urcorner) : \quad t_0,...,t_n \rightarrow \ulcorner\{t_0\}^{pc}_{t_1^{z_1}\cdot\ldots\cdot t_n^{z_n}}\urcorner$$

$$\text{si} \quad \ulcorner\{t_0\}^{pc}_{t_1^{z_1}\cdot\ldots\cdot t_n^{z_n}}\urcorner \neq \{..\}^{pc}_{..}$$

Règles de composition

$$L_{ECc}(\ulcorner\{t_0\}^{pc}_{t_1^{z_1}\cdot\ldots\cdot t_n^{z_n}}\urcorner) : \quad t_0,...,t_n \rightarrow \ulcorner\{t_0\}^{pc}_{t_1^{z_1}\cdot\ldots\cdot t_n^{z_n}}\urcorner$$

$$\text{si} \quad \ulcorner\{t_0\}^{pc}_{t_1^{z_1}\cdot\ldots\cdot t_n^{z_n}}\urcorner = \{..\}^{pc}_{..}$$

Tab. 2.7: Règles de l'intrus EC

$\{u_j\} \subseteq \{t_i\}$ si $t_0 = Exp(t'_0, t_{n+1}^{z_{n+1}} \cdot .. \cdot t_m^{z_m})$, ce qui prouve que les règles L_{DHc} sont des règles d'intrus normalisé de composition. Le chapitre 5 détaillera la recherche d'attaques avec ces règles. On peut alors construire l'intrus DH (pour propriétés de Diffie-Hellman) :

Définition 2.4.5.5 *Intrus de Diffie-Hellman.*
L'intrus DH est l'ensemble $\mathcal{L}_{DH}$ des règles d'intrus normalisé $\mathcal{L}_{DY} \cup L_{DHd} \cup L_{DHc}$.

Il nous reste à définir les règles d'intrus utilisées pour l'opérateur d'encryption commutative $\{..\}^{pc}_{..}$. Celles-ci sont assez semblables aux règles DH précédentes. La différence essentielle porte sur les coefficients, qui doivent à présent être positifs. Cela changera la manière dont on va résoudre les équations liées à l'intrus dans la recherche d'attaque (chapitre 6). On définit donc les règles d'intrus $L_{ECd} = \bigcup_t L_{ECd}(t)$ et $L_{ECc} = \bigcup_t L_{ECc}(t)$ à partir de la Table 2.7, avec $\forall i$, $t_i \in Terme$ et $z_i \in I\!N^*$.

La seule différence entre la normalisation avec $\{..\}^{pc}_{..}$ et la normalisation avec Exp portant sur les termes k, k^* dans les produits, les règles L_{ECd} et L_{ECc} sont des règles d'intrus normalisé respectivement de décomposition de composition pour les mêmes raisons que L_{DHd} et L_{DHc}. Nous verrons en détails les différences dans la recherche d'attaques avec ces deux opérateurs au chapitre 6. On construit l'intrus EC (pour Encryption Commutative) avec ces règles :

Définition 2.4.5.6 *Intrus pour l'encryption commutative.*
L'intrus EC est l'ensemble des règles d'intrus normalisé $\mathcal{L}_{DY} \cup L_{ECd} \cup L_{ECc}$.

On définit à présent un genre particulier de dérivations, très utiles pour limiter la taille des dérivations nécessaires à la construction d'une attaque :

Définition 2.4.5.7 *Dérivations bien formées.*
Une dérivation $D = E \rightarrow_{L_1} E, t_1 \rightarrow ... \rightarrow_{L_n} E, t_1, .., t_n$ de but $t = t_n$ est dite bien formée quand pour tout $i \in \{1..n\}$, on a :
- *L_i est une règle d'intrus normalisé,*
- *Si L_i est une règle d'intrus normalisé de composition, alors $t_i \in STermes(E, t)$, et*

– *Si L_i est une règle d'intrus normalisé de décomposition, alors $t_i \in STermes(E)$.*

Rappelons que par définition, une dérivation ne peut pas créer deux fois le même terme (même avec des règles différentes). Nous verrons aux chapitres 3 à 6 que pour tout intrus dans $\mathcal{L}_{DY}$, $\mathcal{L}_{xor}$, $\mathcal{L}_{D.H.}$ ou $\mathcal{L}_{E.C.}$, il existe une dérivation partant de E et de but t si et seulement s'il existe une dérivation bien formée partant de E et de but t. Ainsi, nous pourrons nous restreindre à utiliser dans la recherche d'attaques des dérivations structurellement plus simples.

2.4.6 Attaque normalisée

Le but de cette section est d'adapter la notion d'attaque à la notion d'intrus normalisé. En effet, dans ce type d'intrus on a délibérément restreint les règles pour ne créer que des termes normalisés (à partir de termes normalisés). Or la notion d'attaque définie en Section 2.3.5 appliquée directement utiliserait des termes $R\sigma$ et $S\sigma$ non normalisés (même si R, S et σ le sont). On définit donc :

Définition 2.4.6.1 *Attaque normalisée.*

Soit $P = (\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}\}, <_{\mathcal{I}}, S)$ un protocole bien formé, et soit $\mathcal{L}$ un intrus normalisé. Alors une attaque normalisée sur P avec $\mathcal{L}$ est une exécution (π, σ) de P, avec π de domaine $\mathcal{J}$ vu comme une bijection de $\mathcal{J}$ dans $\{1..\#\mathcal{J}\}$, telle que :

$$\forall i \in \{1..\#\mathcal{I}\}, \quad \begin{array}{rcl} \ulcorner R_i \urcorner & \in & forge_{\mathcal{L}}(S, \ulcorner S'_1\sigma \urcorner, ..., \ulcorner S'_{i-1}\sigma \urcorner) \\ Secret & \in & forge_{\mathcal{L}}(S, \ulcorner S'_1\sigma \urcorner, ..., \ulcorner S'_{\#\mathcal{I}}\sigma \urcorner) \end{array}$$

avec $R_\iota = R'_{\pi(\iota)}$ et $S_\iota = S'_{\pi(\iota)}$ pour tout $\iota \in \mathcal{J}$.

On peut se demander si cette notion d'attaque normalisée ne serait pas plus restreinte que la notion d'attaque habituelle, avec des règles d'intrus portant sur n'importe quel terme, normalisé ou non. Pour voir cela, définissons $\mathcal{L}^g_{xor}$, $\mathcal{L}^g_{D.H.}$ et $\mathcal{L}^g_{E.C.}$ les intrus (au sens habituel de la section 2.3.3) créés à partir de $\mathcal{L}_{xor}$, $\mathcal{L}_{D.H.}$ et $\mathcal{L}_{E.C.}$ en éliminant la contrainte de termes normalisés (et donc sans différencier décryptage et encryption), et complétés par toutes les règles d'équivalence de termes issues de la Table 2.4 (de gauche à droite et de droite à gauche) respectivement sur les opérateurs $\oplus$, Exp et $\cdot$, et $\{..\}^{pc}_{\cdot\cdot}$ et $\cdot$. On se rend assez facilement compte que ces nouveaux intrus $\mathcal{L}^g_{xor}$, $\mathcal{L}^g_{D.H.}$ et $\mathcal{L}^g_{E.C.}$ se comportent exactement de la même manière que $\mathcal{L}_{xor}$, $\mathcal{L}_{D.H.}$ et $\mathcal{L}_{E.C.}$, ce qui se traduit par :

Lemme 2.4.6.2 *Équivalence entre intrus et intrus normalisé.*

Pour tout ensemble de termes E, et pour tout intrus $\mathcal{L} \in \{\mathcal{L}_{xor}, \mathcal{L}_{D.H.}, \mathcal{L}_{E.C.}\}$, on a :

$$\ulcorner forge_{\mathcal{L}^g}(E) \urcorner = forge_{\mathcal{L}}(\ulcorner E \urcorner)$$

Pour le prouver il suffit de regarder les règles une par une : les intrus $\mathcal{L}^g$ ne créent que des termes appartenant aux classes d'équivalence des termes normalisés créés par les intrus normalisés correspondants. En conséquence, on obtient une équivalence entre les attaques au sens habituel (section 2.3.3) et les attaques normalisées (c.f. [24] pour les détails de cette preuve) :

Corollaire 2.4.6.3 *Équivalence entre attaque et attaque normalisée.*

Soit P un protocole bien formé. Pour tout $\mathcal{L} \in \{\mathcal{L}_{xor}, \mathcal{L}_{D.H.}, \mathcal{L}_{E.C.}\}$, P admet une attaque normalisée avec l'intrus $\mathcal{L}$ si et seulement si P admet une attaque avec l'intrus $\mathcal{L}^g$.

Nous allons définir une notion d'attaque minimale pour décrire les plus petites attaques permettant de résoudre le problème de l'insécurité de protocoles cryptographiques.

Définition 2.4.6.4 *Attaque minimale.*

Soit $P = (\{R_\iota \Rightarrow S_\iota \mid \iota \in \mathcal{I}\}, <_\mathcal{I}, S)$ *un protocole bien formé et* (π, σ) *une attaque normalisée sur* P[7]*. Alors* (π, σ) *est une attaque minimale quand* $\sum_{x \in Var} |\sigma(x), Init, 0, 1|_{dag}$ *est minimale parmi toutes les attaques possibles.*

La mesure $|t, Init, 0, 1|_{dag}$ est une taille DAG du terme t proche de $|t|_{dag}$. On compte ajoute les atomes particuliers 0, 1, et $Init$ pour des raisons techniques. En effet, cela facilitera certaines preuves des chapitres suivants, où l'on remplacera certains termes par $Init$, 0, ou 1 en garantissant une diminution de cette taille un peu spéciale. Une attaque minimale est ainsi un représentant particulier de l'ensemble des attaques sur un protocole donné (face à n'importe quel intrus). Puisque cette définition est basée sur un ordre bien fondé sur les substitutions, il est clair qu'un protocole admet toujours une attaque minimale à partir du moment où il admet au moins une attaque normalisée. En revanche, une attaque minimale n'est pas nécessairement unique.

2.4.7 Exemples

Pour illustrer les modèles de protocole bien formés, d'intrus normalisé, et d'attaque normalisée, nous allons présenter trois protocoles montrant l'utilisation typique des opérateurs $\oplus$, Exp et $\{..\}^{pc}_{..}$ respectivement. Ces trois protocoles se placent tous dans des restrictions différentes du modèle de protocoles bien formés de la section précédente, à savoir les protocoles n'utilisant que les opérateurs classiques et le xor (i.e. sans Exp ni $\{..\}^{pc}_{..}$) appelés *protocoles xor*, les protocoles n'utilisant que les opérateurs classiques et l'exponentielle (i.e. sans $\oplus$ ni $\{..\}_{..}$) appelés *protocoles* DH, et enfin les protocoles n'utilisant que les opérateurs classiques et l'encryption commutative (i.e. sans $\oplus$ ni Exp) appelés *protocoles EC*. Ces trois ensembles de protocoles ne sont que des sous ensembles des protocoles bien formés.

Protocole avec xor

Le premier exemple de protocole que l'on considère est un protocole avec xor. C'est une variante du protocole de Needham-Schroeder-Lowe [57] construite avec l'opérateur xor, à savoir le protocole originel de Needham-Schroeder avec la correction apportée par G.Lowe. L'idée est d'utiliser l'opérateur xor à la place de la concaténation (création d'une paire) à un point bien précis du protocole pour "marquer" un nonce avec le nom du principal l'ayant reçu. On appellera marquage de N par B le message $N \oplus B$. Voici tout d'abord la spécification de ce protocole tel qu'on peut la trouver dans la littérature, i.e. dans le modèle standard de la section 2.2 :

$$\begin{array}{ll} 1.\ A \to B: & \{N_A,\ A\}^p_{K_B} \\ 2.\ B \to A: & \{N_B,\ N_A \oplus B\}^p_{K_A} \\ 3.\ A \to B: & \{N_B\}^p_{K_B} \end{array}$$

L'idée de ce protocole est de permettre à A et B de s'authentifier mutuellement. Pour cela, A crée un nonce N_A qu'il envoie encrypté à B. C'est le défi (ou challenge) de A pour B. En principe, seul B pourra décrypter ce message et retrouver N_A. Une fois ce nonce connu, B crée le nonce N_B pour A, et le lui envoie avec $N_A \oplus B$ i.e. un marquage du nonce N_A par le nom

[7] Avec n'importe quel intrus.

de B. Cela permet en principe de garantir l'identité du principal ayant généré N_B. Le principal A peut alors authentifier B grâce à N_A, et prouver son identité en revoyant N_B. On cherche ici une attaque de secret sur N_B, i.e. $N_B = Secret$. Ce n'est pas tout-à-fait équivalent à une attaque d'authentification classique (i.e. l'intrus peut changer la valeur de N_B connue par A sans connaître N_B lui même), mais ce sera tout de même suffisant. Tous ces messages étant encryptés par les clefs publiques des principaux les recevant, l'intrus n'a normalement aucun moyen de calculer N_B. De fait, si l'on ne prend pas en compte les propriétés algébriques de l'opérateur xor, ce protocole a été prouvé comme étant sûr [57]. Néanmoins, si l'on prend en compte des propriétés algébriques du xor, ce protocole possède une attaque permettant à l'intrus de calculer N_B. C'est une variante de l'attaque originelle sur le protocole de Needham-Schroeder. On considère ici deux sessions parallèles de ce protocole. La première est une session d'authentification classique entre un principal A et un principal B. La seconde en revanche est une session d'authentification entre le principal A et l'intrus I jouant officiellement en son nom propre. Le but de l'intrus est d'utiliser sa session officielle avec A pour attaquer la session d'authentification entre A et B. Voici tout d'abord la spécification dans le modèle de protocole bien formé :

$$\begin{array}{lrcl}
AB_1 : & Init & \Rightarrow & \{N_A,\, A\}^p_{K_B} \\
AB_2 : & \{x_{Nb},\, N_A \oplus B\}^p_{K_A} & \Rightarrow & \{x_{Nb}\}^p_{K_B} \\
 & & & \\
BA_1 : & \{x_{Na},\, A\}^p_{K_B} & \Rightarrow & \{N_B,\, x_{Na} \oplus B\}^p_{K_A} \\
BA_2 : & \{N_B\}^p_{K_B} & \Rightarrow & End \\
 & & & \\
AI_1 : & Init & \Rightarrow & \{N'_A,\, A\}^p_{K_I} \\
AI_2 : & \{x_{Ni},\, N'_A \oplus I\}^p_{K_A} & \Rightarrow & \{x_{Ni}\}^p_{K_I}
\end{array}$$

Il n'y a pas lieu de spécifier le rôle joué par l'intrus, puisque les règles d'intrus sont strictement plus expressives que les règles de protocole. Il faut néanmoins correctement définir les connaissances initiales de l'intrus. C'est $S = \{Init, A, B, I, K_A, K_B, K_I, K_I{}^*\}$, i.e. le terme spécial $Init$ pour démarrer une session, les noms des principaux en jeu, les clefs publiques de ces principaux, et un couple de clefs privées/publiques propre à l'intrus. L'ordre sur les pas de protocole est simplement $AB_1 <_{\mathcal{I}} AB_2$, $BA_1 <_{\mathcal{I}} BA_2$, et $AI_1 <_{\mathcal{I}} AI_2$ pour $\mathcal{I} = \{AB_1, AB_2, BA_1, BA_2, AI_1, AI_2\}$. On peut à présent détailler une attaque de ce protocole. Il est important de noter que dans cette attaque, l'intrus génère de nouveaux messages grâce à l'opérateur $\oplus$, et utilise le fait que $N_A \oplus B \oplus I \oplus B = N_A \oplus I$. L'attaque est la suivante :

$$\begin{array}{rcl}
\pi & = & AI_1 <_\pi BA_1 <_\pi AI_2 \\
\sigma(x_{Na}) & = & N'_A \oplus B \oplus I \\
\sigma(x_{Ni}) & = & N_B
\end{array}$$

On peut détailler cette attaque en en donnant la trace :

$$\begin{array}{lrcl}
AI_1 : & Init & \Rightarrow & \{N'_A,\, A\}^p_{K_I} \\
BA_1 : & \{N'_A \oplus B \oplus I,\, A\}^p_{K_B} & \Rightarrow & \{N_B,\, N'_A \oplus B \oplus I \oplus B\}^p_{K_A} = \{N_B,\, N'_A \oplus I\}^p_{K_A} \\
AI_2 : & \{N_B,\, N'_A \oplus I\}^p_{K_A} & \Rightarrow & \{N_B\}^p_{K_I}
\end{array}$$

On se rend bien compte ici que l'intrus a pu déjouer le marquage du nonce N'_A par B en marquant lui-même auparavant le nonce avec $B \oplus I$. Ainsi, les deux B s'annulent et le résultat est un nonce marqué par I uniquement. Il ne reste plus qu'à vérifier que l'intrus xor est bien capable de construire les messages reçus par les principaux lors de cette attaque. En effet :

- Pas AI_1 : l'intrus connaît $Init$ dès le départ,
- Pas BA_1 : l'intrus connaît K_B, A, B, et I dès le départ, et il obtient N'_A en décomposant $\{N'_A, A\}^p_{K_I}$.
- Pas AI_2 : l'intrus connaît $\{N_B, N'_A \oplus I\}^p_{K_A}$ grâce au pas BA_1. Il obtient alors $N_B = Secret$ en décomposant $\{N_B\}^p_{K_I}$, puisqu'il connaît ${K_I}^*$.

Ainsi, cette exécution de protocole est bien une attaque.

Protocole avec exponentielle

Pour illustrer l'utilisation des opérateurs produit et exponentielle, nous présentons un protocole DH très utilisé sur Internet, à savoir le protocole point à point [88] (Station to Station protocol). Ce protocole est basé sur le protocole d'échange de clefs de Diffie-Hellman. L'idée est d'éviter la vulnérabilité évidente de ce protocole à l'attaque de l'intercepteur en obligeant les deux participants (Alice et Bernard) à signer leurs messages respectifs. Pour cela, on suppose que ces deux principaux possèdent chacun un couple clef privée-clef publique, et que les clefs publiques ont déjà été partagées, par exemple par l'intermédiaire d'un tiers de confiance. Voici le détail de ce protocole, spécifié dans notre modèle à partir de [88] :

$$
\begin{array}{rll}
Alice_1 : & Init & \Rightarrow Exp(g,\, Na) \\
Bernard_1 : & x_{gNa} & \Rightarrow Exp(g,\, Nb),\, \left\{\{x_{gNa},\, Exp(g,\, Nb)\}^p_{Kb^*}\right\}^s_{Exp(x_{gNa},\, Nb)} \\
Alice_2 : & & x_{gNb},\, \left\{\{Exp(g,\, Na),\, x_{gNb}\}^p_{Kb^*}\right\}^s_{Exp(x_{gNb},\, Na)} \\
 & & \Rightarrow \left\{\{Exp(g,\, Na),\, x_{gNb}\}^p_{Ka^*}\right\}^s_{Exp(x_{gNb},\, Na)} \\
Bernard_2 : & \left\{\{x_{gNa},\, Exp(g,\, Nb)\}^p_{Ka^*}\right\}^s_{Exp(x_{gNb},\, Na)} & \Rightarrow Fin
\end{array}
$$

On constate aisément que ce protocole est bien formé. Le protocole de Diffie-Hellman sert à générer la clef secrète partagée. Du point de vue d'*Alice*, c'est $Exp(x_{gNb},\, Na)$, et du point de vue de Bernard c'est $Exp(x_{gNa},\, Nb)$. Le protocole originel est étendu avec l'échange des clefs publiques $Exp(g,\, Na)$ et $Exp(g,\, Nb)$ pour certifier qu'il n'y a pas eu de substitution de clef lors de l'exécution du protocole, signé avec les clefs publiques d'*Alice* et de *Bernard* pour garantir l'identité des participants, et crypté par la clef secrète partagée. A notre connaissance, il n'y a pas d'attaque (pour le moment) sur ce protocole. Il sera très intéressant, quand des outils concrets comme Atsé (c.f. section 8) pourront utiliser les règles d'intrus DH présentées ici, de vérifier ce protocole (ainsi que des versions plus ou moins simplifiées).

Protocole avec encryption commutative

Il nous reste à donner un exemple de protocole EC exploitant la commutation de clef. Ce sera le protocole bien connu RSA. En effet, ce protocole exploite la commutation de clefs pour transmettre un secret. Dans le modèle Alice-Bob, ce protocole s'écrit :

$$
\begin{array}{ll}
A \rightarrow B : & \{Secret\}^{pc}_{K_A} \\
B \rightarrow A : & \left\{\{Secret\}^{pc}_{K_A}\right\}^{pc}_{K_B} \\
A \rightarrow B : & \{Secret\}^{pc}_{K_B}
\end{array}
$$

L'idée de ce protocole est la suivante : Le principal A envoie le secret à B encrypté par la clef publique de A. Ce message $\{Secret\}^p_{K_A}$ ne peut être décomposé par personne d'autre que A, car lui seul possède la clef privée ${K_A}^*$. B reçoit ce message, et le renvoie encrypté avec sa propre clef privée. Utilisant la commutation de l'encryption, ce message peut aussi être écrit

$\left\{\{Secret\}^{p}_{K_B}\right\}^{p}_{K_A}$. Ainsi, le principal A peut calculer $\{Secret\}^{p}_{K_B}$ et le renvoyer à B. Seul B peut décrypter ce message, et il récupère le secret. Ce protocole suppose l'utilisation d'un module RSA commun n. La clef publique de A est alors (n, K_A) et celle de B est (n, K_B). Le terme $Secret$ est un nonce, i.e. un entier positif. Ainsi, le terme $\{Secret\}^{pc}_{K_A}$ représente $Secret^{K_A} \bmod n$, ce qui donne la commutation de l'encryption grâce aux propriétés algébriques de l'exponentiation. Cela se traduit au dernier pas du protocole par le fait que A calcule $\{\{\{Secret\}^{pc}_{K_A}\}^{pc}_{K_B}\}^{pc}_{K_A{}^*} = \{Secret\}^{pc}_{K_A}$. En conséquence, le protocole lui-même utilise la commutation de l'encryption. Ce protocole RSA est malheureusement trivialement vulnérable à l'attaque de l'intercepteur, puisque B n'est jamais authentifié : l'intrus peut se contenter d'imiter B en utilisant son propre couple de clefs publique/privée K_I, $K_I{}^*$ à la place de K_B et $K_B{}^*$.

2.5 Extensions directes du modèle par rôles.

Dans certains cas, le modèle de protocole que l'on a défini peut sembler assez restrictif, même si l'on ne se limite pas aux protocoles bien formés. Par exemple, on n'autorise les principaux qu'à utiliser du filtrage sur les messages reçus, alors qu'ils devraient pouvoir également filtrer leurs connaissances pour décomposer une ancienne connaissance avec une clef nouvellement reçue (c.f. sous section 2.5.1) En outre, on spécifie des protocoles communiquant sur un seul et unique canal public, dont les propriétés sont fixées par avance, alors que certains protocoles concrets exploitent différents canaux plus ou moins accessibles par l'intrus. Par exemple, sur certains canaux l'intrus peut lire les messages transmis mais ne peut pas les modifier, ou bien il peut écrire sur le canal mais ne peut pas intercepter de message, etc... (c.f. sous section 2.5.2) Le but de cette section est de donner une intuition suffisamment précise des différents codages à mettre en oeuvre pour pouvoir exprimer ces différentes extensions dans notre modèle de protocole. La taille du protocole traduit devra être bornée par un polynôme en la taille du protocole d'origine, pour que les résultats de complexité des chapitres suivants soient valides dans ces extensions.

2.5.1 Filtrage de connaissances

On aimerait pouvoir exprimer dans notre modèle de protocoles des agents capables de décomposer des connaissances passées quand la clef nécessaire vient d'être reçue. En particulier, il ne nous est pas possible de traduire *directement* le protocole suivant, où B ne connaît pas K au départ :

$$\begin{array}{ll} A \to B : & \{M\}^{s}_{K} \\ B \to A : & B \\ A \to B : & K \\ B \to A : & M \end{array}$$

En effet, le message $\{M\}^{s}_{K}$ est nécessairement vu comme une variable x au premier pas de B, et par la suite le modèle par rôles ne nous permet pas de filtrer directement x si aucun message contenant x n'est à nouveau reçu. D'une manière plus générale, il faudrait permettre aux principaux d'effectuer un test d'égalité à chaque pas de protocole. Ainsi, le principal B pourrait unifier x avec $\{y\}_{x_K}$ après avoir reçu x_K (supposé contenir la clef K). Pour cela, considérons la spécification suivante de ce protocole, avec les connaissances initiales de l'intrus $S = \{Init\}$ et l'ordre habituel rôle par rôle :

Principal A			Principal B			
$Init$	$\Rightarrow$	$\{M\}^s_K$	x	$\Rightarrow$	B	
B	$\Rightarrow$	K	x_K	$\Rightarrow$	y	$\left[x = \{y\}^s_{x_K}\right]$
M	$\Rightarrow$	Fin				

Dans cette modélisation, le second pas de protocole de B doit être interprété de la manière suivante : B reçoit le message x_K, puis unifie sa connaissance x avec le terme $\{y\}^s_{x_K}$, et obtient la connaissance y qu'il peut donc envoyer à A. Ceci revient à dire que le principal B décompose sa connaissance x avec la clef x_K, ce qui lui donne y qu'il envoie ensuite à A. Appelons *protocoles avec contrainte* les protocoles construits avec ce genre de contrainte sur chaque pas. Il en découle des notions d'exécution et d'attaque (avec contrainte) de la même manière que pour les protocoles habituels. La seule différence est que pour tout protocole avec contrainte P, pour toute exécution (π, σ) de ce protocole, et pour toute contrainte $[\alpha = \beta]$ présente dans un pas de protocole du domaine de π, on doit avoir $\sigma(\alpha) = \sigma(\beta)$. Cette notion de protocole avec contrainte semble nettement plus générale que le modèle de protocole des sections précédentes. Pourtant, on peut exploiter les capacités de l'intrus pour coder (linéairement) un protocole avec contrainte dans un protocole sans contrainte. Pour cela, il suffit de générer des clefs T_i pour chaque contrainte $[a = b]$ du protocole de départ, et de remplacer chaque règle $R \Rightarrow S\ [\alpha = \beta]$ par deux règles $R \Rightarrow \{\alpha\}^s_{T_i}$ et $\{\beta\}^s_{T_i} \Rightarrow S$. En effet, comme toutes les clefs T_i sont inconnues de l'intrus et comme chaque clef T_i n'est utilisée que pour ces deux nouvelles règles, on garantit (si l'encryption symétrique n'a aucune propriété algébrique) d'une part que le terme $\{\beta\}^s_{T_i}$ reçu sera unifié avec le terme $\{\alpha\}^s_{T_i}$ émis au pas précédent, et d'autre part que la connaissance $\{\beta\}^s_{T_i}$ ne sera pas plus utile à l'intrus qu'un nouvel atome N_i (n'apparaissant pas dans le protocole). On obtient le codage suivant pour notre protocole, avec une nouvelle clef symétrique T :

Principal A			Principal B		
$Init$	$\Rightarrow$	$\{M\}^s_K$	x	$\Rightarrow$	B
B	$\Rightarrow$	K	x_K	$\Rightarrow$	$\{x\}^s_T$
M	$\Rightarrow$	Fin	$\left\{\{y\}^s_{x_K}\right\}^s_T$	$\Rightarrow$	y

Cependant, pour que ce codage soit valide il faut faire attention à obtenir un protocole satisfaisant la définition de protocole dans notre modèle. En particulier, il faut que toute variable émise ait été reçue précédemment. Sur notre protocole, c'est le cas mais en inversant x et $\{y\}^s_{x_K}$ on obtiendrait un protocole invalide. Pour garantir la validité de la traduction, il suffit d'étendre aux contraintes la condition sur les variables émises : pour tout protocole avec contrainte $P = (\{R_\iota \Rightarrow S_\iota\,[a_\iota = b_\iota] \mid \iota \in \mathcal{I}\}, <_\mathcal{I}, S)$, et pour tout pas $R_\iota \Rightarrow S_\iota\,[a_\iota = b_\iota]$ de P, on doit avoir $Var(a_\iota) \subseteq \bigcup_{\iota' \leq_\mathcal{I} \iota} Var(R_{\iota'})$. C'est un codage possible parmi beaucoup d'autres, mais il permet d'ajouter assez simplement des contraintes à certains pas de protocole. De plus, ce codage est linéaire, puisque dans le pire des cas on crée un nombre linéaire de nouveaux pas de protocole utilisant des termes déjà présents dans la spécification d'origine plus un niveau d'encryption avec une clef atomique.

Remarque 2.5.1.1 *Ce codage est adapté à une seule sessions, comme décrite par la définition de protocole. Cependant, on peut aisément coder plusieurs sessions en utilisant pour chacune des clefs T_i différentes.*

En résumé, on a construit pas à pas la définition suivante de protocole avec contrainte :

Définition 2.5.1.2 *Protocole avec contrainte.*

Un protocole avec contrainte est un triplet $(\{R_\iota \Rightarrow S_\iota\,[\alpha_\iota = \beta_\iota] \mid \iota \in \mathcal{I}\}, <_\mathcal{I}, S)$ *où* S *est un ensemble fini de messages,* $<_\mathcal{I}$ *est un ordre partiel sur l'ensemble fini* $\mathcal{I}$*, et pour tout* $\iota \in \mathcal{I}$ *on a :*

$$(C_2) \ : \quad \textit{Pour tout } \iota \in \mathcal{I}, \textit{ on a } Var(S_\iota, \alpha_\iota) \subseteq \bigcup_{\iota' \leq_\mathcal{I} \iota} Var(R_{\iota'}).$$

De plus, la construction précédente justifie la proposition suivante :

Proposition 2.5.1.3 *Équivalence entre protocole avec contrainte et protocole (habituel).*

Pour tout protocole avec contrainte P*, il existe un protocole (sans contrainte)* P' *tel que* P *admet une attaque ssi* P' *admet une attaque, et* $\|P'\| \leq 2.\|P\|$.

Enfin, on se rend compte assez facilement que l'on peut étendre la définition et la propriété ci-dessus aux protocoles bien formés, c'est à dire étendre la définition de protocole bien formé aux protocoles avec contrainte de manière à ce que la construction précédente donne un protocole bien formé. En effet, il suffit d'imposer à chaque β_ι, $\iota \in \mathcal{I}$, la même contrainte que sur les R_ι d'un protocole bien formé. De cette manière, les (nouveaux) messages reçus $\{\beta_\iota\}^{pc}_{T_\iota}$ vérifieront bien la contrainte de protocole bien formé. Comme précédemment, cette construction préserve les propriétés de sécurité ou d'insécurité de protocole, et admet naturellement la même borne sur la taille du protocole généré.

2.5.2 Communication multicanal

En général, les protocoles basés sur Internet n'utilisent qu'un seul médium de communication. Cependant, certains protocoles travaillant sur plusieurs réseaux simultanément nécessitent l'utilisation de différents canaux de communication sur lesquels l'intrus peut agir de manière différente. Il existe dans la littérature de nombreuses définitions de canaux avec des propriétés très variées. Nous n'en utiliserons ici que quelques unes. En particulier, nous ne décrirons pas les canaux utilisant des files d'attente de messages sur lesquelles l'intrus peut inverser (sans effacer) plusieurs messages. Le modèle de protocole présenté ici est loin de permettre cela, et le codage nécessaire serait très loin d'être intuitif. En revanche, nous pouvons restreindre les capacités d'interception, lecture ou écriture des messages de l'intrus pour certains pas de protocole. De plus, le fait que toutes les variables d'un protocole soient globales peut nous permettre de modéliser des principaux partageant des connaissances, et donc de modéliser des canaux sûrs ou presque sûrs. Chaque type de canal de communication ayant ses propres propriétés, nous allons simplement donner quelques codages des restrictions les plus courantes que l'on peut appliquer à l'intrus. Considérons par exemple le protocole suivant, pour un canal publique "c" :

$$\begin{array}{ll} A \to B : & N_A \text{ sur le canal public} \\ B \to C : & N_A \text{ sur le canal c} \\ C \to A : & N_A \text{ sur le canal public} \end{array}$$

Ce protocole très simple se traduit normalement dans notre modèle de protocole de la manière suivante, sans tenir compte des canaux, et en éliminant le dernier pas inutile $N_A \Rightarrow Fin$:

$$\begin{array}{lrcl} A : & Init & \Rightarrow & N_A \\ B : & x & \Rightarrow & x \\ C : & y & \Rightarrow & y \end{array}$$

Cependant, si l'on considère que le canal c permet à l'intrus de lire et intercepter les messages, mais pas de les modifier ni d'envoyer de message sur ce c, il nous suffit (à la manière de la section précédente) de créer une nouvelle clef T, propre à ce pas de protocole, et de spécifier le protocole ainsi :

$$\begin{array}{lrcl} A: & Init & \Rightarrow & N_A \\ B: & x & \Rightarrow & x,\, \{x\}_T^s \\ C: & y,\, \{y\}_T^s & \Rightarrow & y \end{array}$$

Ainsi, l'intrus connaîtra x mais sera tenu de l'envoyer à C tel quel à ce pas de protocole. Si l'on veut cependant permettre à l'intrus de permuter et réutiliser des messages sur ce canal, on peut par exemple utiliser la même clef T_c, propre au canal c, pour tous les messages transmis sur c. A l'inverse, si l'on veut permettre à l'intrus d'écrire sur le canal c sans lui permettre de lire les messages qui y sont transmis par les principaux, on peut utiliser un couple clef public/privée T, T^* en ne mettant que T dans les connaissances de l'intrus :

$$\begin{array}{lrcl} A: & Init & \Rightarrow & N_A \\ B: & x & \Rightarrow & \{x\}_T^p \\ C: & \{y\}_T^p & \Rightarrow & y \end{array}$$

Ainsi, l'intrus peut générer un message $\{..\}_T^p$ et peut réutiliser des messages déjà transmis (si on n'utilise pas un couple de clefs différent pour chaque pas), mais il ne pourra pas décrypter $\{x\}_T^p$. Tout ceci n'est évidement valable que si l'on utilise un opérateur d'encryption sans aucune propriété algébrique. Dans le cas contraire, un tel codage est beaucoup plus problématique. Enfin, on peut modéliser simplement un canal c totalement inaccessible à l'intrus. Pour cela, il nous suffit d'effectuer, dès la spécification du protocole, l'unification entre le message transmis et le message devant être reçu. Dans notre exemple, cela revient à remplacer toutes les occurrences de y par x, puisque l'intrus n'a aucun moyen de changer le message, et d'enlever l'envoi et la réception de ce message, puisque du point de vue de l'intrus il n'a même pas été transmis. On obtient :

$$\begin{array}{lrcl} A: & Init & \Rightarrow & N_A \\ B: & x & \Rightarrow & ... \\ C: & ... & \Rightarrow & x \end{array}$$

avec n'importe quel message ... connu de tous, et l'ordre $B < C$ puisque C ne peut transmettre qu'après l'action de B. Ainsi, le principal C peut utiliser la variable x même si ce n'est pas lui qui l'a reçue. En outre, l'ordre partiel permettrait à C d'exécuter un autre pas avant celui-ci, dans le cas d'un rôle plus important. Par exemple, on peut avoir $A < C_1$ pour la transmission sur le canal c et $C_0 < C_1 < C_2$ pour définir le rôle de C.

2.6 Conclusion

Dans ce chapitre, nous avons présenté toutes les notions de base dont nous aurons besoin dans les chapitres suivants. Il s'agit essentiellement des définitions des modèles de protocole Alice-Bob et par rôle, avec et sans opérateurs algébriques. Ceci nous a permis de définir les notions d'exécution de protocole et d'attaque, elle même basée sur la notion de règle d'intrus décrivant tous les messages calculable par un intrus donné. Dans le cas des protocoles avec opérateur algébrique, nous avons défini une notion de normalisation de message nous permettant de prendre en compte les propriétés des opérateurs algébriques considérés. Ceci nous a amené à

une notion un peu particulière de protocole bien formé, c'est à dire une classe de spécifications de protocoles cryptographiques pouvant être implémentés en réalité. Ceci nous permet d'éviter des pas de protocole absurdes en pratique comme $x \oplus y \Rightarrow x$ avec y inconnu.

3

Insécurité dans le modèle de base

Sommaire

Nous avons vu au chapitre 1 qu'il existe de nombreuses méthodes formelles permettant de trouver des attaques dans des protocoles cryptographiques. Il s'agit entre autres de méthodes d'analyse finie basées sur le model-checking (p.ex [8, 29, 58, 87]), de méthodes de programmation logique ([63]) ou de réécriture ([39, 54]), de méthodes à base d'automates d'arbres ([32, 50]) ou des combinaisons de ces techniques. D'autres techniques ont surtout pour but de prouver la sécurité de protocoles, par exemple par induction avec un assistant de preuve interactif ([14, 76]).

Même si le problème général de décision de la sécurité des protocoles cryptographiques est indécidable (c.f. [47]), et même dans le cas où les tailles des messages transmis sont bornées ([45]), il est intéressant d'étudier différentes classes décidables de protocoles cryptographiques, et leur complexité. En particulier, le fait que des outils concrets de vérification de protocoles donnent des résultats intéressants montre qu'il existe probablement de telles classes représentant de nombreux protocoles réels. Plusieurs travaux vont dans cette direction. Outre la procédure de décision polynomiale de Dolev-Yao sur les protocoles ping-pong ([43]), et une procédure DEXPTIME de J. Mitchell *et al.* [45] pour le cas sessions non bornées, messages bornés et sans nonces, on trouve également des procédures de décision pour un nombre fini de sessions (et messages non bornés) proposées par A. Huima ([53]) et R. Amadio *et al.* ([4, 6]). Dans [6], les clefs sont restreintes à des atomes et ne peuvent pas être des messages quelconques.

Dans ce chapitre, nous allons nous concentrer sur la complexité du problème de l'insécurité de protocole face à l'intrus de Dolev-Yao, dans le modèle par rôles (et donc à nombre de sessions borné). Nous allons tout d'abord examiner la borne inférieure de ce problème. Nous montrerons notamment qu'il est NP-difficile même pour des protocoles très restreints. Ensuite, nous décrirons les bornes sur les tailles de certaines attaques (les attaques minimales), et nous en déduirons un algorithme NP pour le problème de l'insécurité face à l'intrus de Dolev-Yao (sans restriction). Enfin, nous verrons comment ce résultat peut être aisément étendu pour permettre l'utilisation du cryptage involutif et des points de choix. Dans tout ce chapitre, nous n'utiliserons jamais les opérateur particuliers xor, produit, exponentielle et encryption commutative. Tous les protocoles seront toujours supposés spécifiés sans ces opérateurs, et en conséquence tous les termes et tous les messages considérés seront toujours normalisés. Nous n'aurons donc pas besoin de normaliser les messages dans ce chapitre.

3.1 Borne inférieure sur la complexité du problème

Coder un problème NP-complet dans le problème de l'insécurité de protocole n'est pas fondamentalement difficile, mais il est plus subtil et assez intéressant de tenter de restreindre le modèle de protocole lors de ce codage. Ainsi, on peut se rendre compte que les sources de NP-complétude sont multiples dans ce problème. En particulier, nous allons voir que le problème de l'insécurité de protocoles est NP-difficile pour les protocoles avec paire mais sans clef composée ni variable en position clef, ainsi que pour les protocoles sans paire, sans clef composée, et avec un seul principal dont les pas de protocoles sont totalement ordonnés (i.e. le choix d'un ordre d'exécution n'intervient pas).

3.1.1 Protocoles sans clef composée ni variable en position clef

Nous allons effectuer une réduction à partir du problème 3-SAT. Cette réduction est proche de celle présentée par [4] pour leur modèle. La différence principale est qu'ici on n'a besoin d'aucun branchement conditionnel dans la spécification du protocole. Les variables propositionnelles[8] sont $\{x_1, .., x_p\} = Var$, et une instance de 3-SAT est $f(\overrightarrow{x}) = \bigwedge_{i\in\{1,..,n\}}(x_{i,1}^{\epsilon_{i,1}} \vee x_{i,2}^{\epsilon_{i,2}} \vee x_{i,3}^{\epsilon_{i,3}})$ où $\epsilon_{i,j} \in \{-1, 1\}$, x^1 représente x, x^{-1} représente $\neg x$, chaque $x_{i,j}$ représente un x_k, $k \in \{1, .., p\}$, et $\overrightarrow{x} = x_1, ..., x_p$. On pose :

- Pour tous $i \in \{1, .., n\}$ et $j \in \{1, 2, 3\}$, $g(1, x_{i,j}) = x_{i,j}$ et $g(-1, x_{i,j}) = \{x_{i,j}\}_K^s$
- Pour tout $i \in \{1, .., n\}$, $f_i(\overrightarrow{x}) = \langle g(\epsilon_{i,1}, x_{i,1}),\ g(\epsilon_{i,2}, x_{i,2}),\ g(\epsilon_{i,3}, x_{i,3})\rangle$

avec la notation $\langle a,\ b,\ ..,\ z\rangle = \langle a,\ \langle b,\ \langle .., z\rangle\rangle\rangle$. L'idée ici est d'utiliser les capacités de l'intrus pour générer un premier message $\langle x_1, ..., x_p\rangle$ représentant une solution possible du problème 3-SAT donné. A partir de ce message initial, le principal A va créer un terme représentant f appliquée à cette solution. On utilise alors les principaux B à D' pour vérifier que la solution proposée par l'intrus est bien une solution du problème 3-SAT donné, i.e. si on peut atteindre l'atome $True$. Le rôle de l'intrus sera ici de choisir l'ordonnancement des pas de protocole, i.e. de prouver que sa solution est valide. Si c'est le cas, le principal E donne le secret $Secret$ à l'intrus, et le protocole a une attaque. Si ce n'est pas possible, le problème 3-SAT donné n'a pas de solution. La description du protocole se trouve dans la Table 3.1.

Pour simplifier la description de ce protocole, on considère que toutes les variables x, y, z ... apparaissant dans la description d'un pas (U, j) sont en fait indexées par (U, j) (pour éviter de

[8] Les variables propositionnelles sont codées directement en variables de protocole.

Principal A :	(A,i),	$\langle x_1,..,x_p\rangle \Rightarrow \{\langle f_1(\overrightarrow{x}),...f_n(\overrightarrow{x}),\ fin\rangle\}^s_P$	
Principal B :	(B,i),	$\{\langle\langle\top,\ x,\ y\rangle,\ z\rangle\}^s_P \Rightarrow \{z\}^s_P$	pour $i \in \{1,..,n\}$
Principal B' :	(B',i),	$\{\langle\langle\{\bot\}^s_K,\ x,\ y\rangle,\ z\rangle\}^s_P \Rightarrow \{z\}^s_P$	pour $i \in \{1,..,n\}$
Principal C :	(C,i),	$\{\langle\langle x,\ \top,\ y\rangle,\ z\rangle\}^s_P \Rightarrow \{z\}^s_P$	pour $i \in \{1,..,n\}$
Principal C' :	(C',i),	$\{\langle\langle x,\ \{\bot\}^s_K,\ y\rangle,\ z\rangle\}^s_P \Rightarrow \{z\}^s_P$	pour $i \in \{1,..,n\}$
Principal D :	(D,i),	$\{\langle\langle x,\ y,\ \top\rangle,\ z\rangle\}^s_P \Rightarrow \{z\}^s_P$	pour $i \in \{1,..,n\}$
Principal D' :	(D',i),	$\{\langle\langle x,\ y,\ \{\bot\}^s_K\rangle,\ z\rangle\}^s_P \Rightarrow \{z\}^s_P$	pour $i \in \{1,..,n\}$
Principal E :	$(E,1)$,	$\{fin\}^s_P \Rightarrow Secret$	

TAB. 3.1: Codage de 3-SAT n°1

surcharger les indices). De plus, le nombre de pas pour chaque principal B à D' doit être égal au nombre de conjonctions dans l'instance de 3-SAT considérée. Les clefs K et P ne sont connues que des principaux.

L'ordre sur ces pas de protocole est vide (tous les ordonnancements sont permis), et les connaissance initiales de l'intrus sont $S_0 = \{\top, \bot\}$. Il existe une attaque sur ce protocole ssi le message transmis par A peut être réduit à $\{fin\}^s_P$ avec les pas $(B,1)$ à (D',n), i.e. pour tout $i \in \{1,..,n\}$ il existe $j \in \{1,2,3\}$ tel que $g(\epsilon_{i,j},\ x_{i,j}) \in \{\top,\ \{\bot\}^s_K\}$. Mais cela signifie que l'intrus A a produit un terme représentant une solution de 3-SAT, puisque $g(\epsilon_{i,j},\ x_{i,j})$ est $x^{\epsilon_{i,j}}_{i,j}$ et que $\{\bot\}^s_K$ est interprété comme $\top$. Ainsi, ce protocole admet une attaque ssi le problème 3-SAT correspondant admet une solution. En conséquence, le problème de l'insécurité de protocoles avec paires mais sans clefs composées ni variable en position clef est NP difficile.

3.1.2 Protocoles sans paire, sans clef composée, et avec un ordre d'exécution fixé (déterministe)

La seconde réduction de 3-SAT que nous présentons ici permet d'évaluer la complexité du problème de l'insécurité dans une classe de protocoles un peu différente. En effet, au lieu d'exploiter l'ordonnancement des pas de protocole comme on l'a fait ci dessus, on va donner la possibilité à l'intrus de "passer" certains pas de protocole. Cela se fera au prix de l'utilisation de variables en position clef. Ainsi, le non déterminisme lié à l'intrus (en dehors de l'ordonnancement des pas de protocole) est suffisant pour avoir un problème NP difficile.

Soit $f(\overrightarrow{x}) = \bigwedge_{i\in\{1,..,n\}}(x^{\epsilon_{i,1}}_{i,1} \vee x^{\epsilon_{i,2}}_{i,2} \vee x^{\epsilon_{i,3}}_{i,3})$ une instance de 3-SAT utilisant les notations précédentes, i.e. $\epsilon_{i,j} \in \{-1,1\}$, x^1 représente x, x^{-1} représente $\neg x$, chaque $x_{i,j}$ représente un x_k avec $k \in \{1,..,p\}$, et $\overrightarrow{x} = x_1,..,x_p$. Dans toute la suite, les variables x et y devront être lues comme indexées par le pas de protocole où elles apparaissent (pour éviter tout effet de bord d'un pas sur un autre sans surcharger la notation). A chaque x_k, $k \in \{1,..,p\}$, on associe un

atome V_k. Les connaissances initiales de l'intrus contiennent :

- $\{\{P\}^s_\bot\}^s_K$, $\{\{P\}^s_\top\}^s_K$, et P. Ceci permettra à l'intrus de fixer les valeurs de $\overrightarrow{x}$ avec $\{P\}^s_\bot$ et $\{P\}^s_\top$.
- $\{\{K\}^s_\bot\}_{V_i}$ et $\{\{K\}^s_\top\}_{V_i}$ pour tout $i \in \{1,..,p\}$. Ce sont des valeurs "bidon" qui permettront à l'intrus de passer certains pas de protocole sans en obtenir aucune connaissance utile.

On n'utilise qu'un seul principal honnête A, possédant $p + 3n + 1$ pas de protocole totalement ordonnés : $(A,i) < (A,i+1)$ pour tout $i \in \{1,..,2p\}$. Ces pas de protocole sont les suivants :

- (A,k) : $\{x\}^s_K \Rightarrow \{x\}^s_{V_k}$ pour tout $k \in \{1,..,p\}$.

  Initialement, l'intrus a la possibilité de sélectionner les valeurs qu'il veut attribuer à chaque x_k dans $\overrightarrow{x}$. Comme il existe un et un seul pas pour chaque atome V_k, ceci attribue une et une seule valeur à chaque x_k : l'instanciation de $\overrightarrow{x}$ est complète et non redondante. De plus, comme l'intrus ne connaît pas K, ces valeurs sont nécessairement $\{P\}^s_\bot$ ou $\{P\}^s_\top$.

- Pour chaque indice de conjonction $i \in \{1,..,n\}$, et pour chaque $k \in \{1,..,p\}$ tel que la variable x_k apparaît dans la conjonction $D_i = x_{i,1}^{\epsilon_{i,1}} \vee x_{i,2}^{\epsilon_{i,2}} \vee x_{i,3}^{\epsilon_{i,3}}$, on définit le pas (A,i,k) suivant :
  - (A,i,k) : $\{\{y\}^s_\top\}^s_{V_k} \Rightarrow \{Secret_i\}^s_y$ si x_k apparaît positivement dans D_i, ou
  - (A,i,k) : $\{\{y\}^s_\bot\}^s_{V_k} \Rightarrow \{Secret_i\}^s_y$ si x_k apparaît négativement dans D_i.

  Le but de l'intrus est de connaître tous les secrets $Secret_i$, car cela prouverait que toutes les conjonctions D_j sont évaluées à $\top$ avec la solution $\overrightarrow{x}$ qu'il a choisi. Pour cela, il doit décomposer $\{Secret_i\}^s_y$ et donc utiliser pour y une valeur qu'il connaît, pour chaque j. Cependant, l'intrus n'a que deux actions possibles : soit il envoie à A le message $\{\{K\}^s_\top\}^s_{V_k}$ ou $\{\{K\}^s_\bot\}^s_{V_k}$, mais recevra en réponse $\{Secret_i\}^s_K$ qu'il n'a aucun moyen de décrypter (d'où un pas de protocole inutile pour l'intrus), soit il envoie au principal A le message $\{\{P\}^s_\top\}^s_{V_k}$ ou $\{\{P\}^s_\bot\}^s_{V_k}$ (s'il le peut) représentant la valeur correcte assignée à x_k par les p premiers pas de protocole, et recevra en échange $\{Secret_i\}^s_P$, décomposable puisque P est connu, et prouvant que la conjonction D_i est évaluée à $\top$. De plus, à partir du moment où une conjonction D_i est évaluée à $\top$, il existe un pas de protocole permettant à l'intrus d'obtenir $\{Secret_i\}^s_P$ (à condition d'envoyer le bon message).

- Le dernier pas de protocole permet de vérifier que l'intrus possède tous les atomes $Secret_i$, i.e. que toutes les conjonctions D_i peuvent être évaluées à vrai. Si c'est le cas, l'intrus obtient $Secret$ et le protocole admet une attaque :

  (A, fin) : $Secret_1$, ..., $Secret_n \Rightarrow Secret$

  Ainsi, l'intrus peut obtenir $Secret$ si et seulement si chaque conjonction D_i est évaluée à vrai, et ce protocole admet une attaque si et seulement si l'instance de 3-SAT correspondante admet une solution.

De cette manière, on a montré que le problème de l'insécurité de protocoles cryptographiques sans paire, sans clef composée, et sans ordonnancement de pas de protocole est tout de même NP difficile. Il est intéressant de remarquer que ce protocole se rapproche beaucoup des protocoles Ping-Pong (c.f. chapitre 6), dont l'insécurité est polynomiale. En effet, l'unique différence est l'utilisation de variables en position clefs (même si celles-ci ne peuvent avoir que des valeurs atomiques dans ce codage).

3.2 Principe de la vérification

Pour pouvoir décider le problème de l'insécurité des protocoles cryptographiques face à l'intrus de Dolev-Yao en temps NP, nous allons calculer (en fonction de la taille d'un protocole donné) les tailles maximales des attaques que l'on doit explorer pour décider de ce problème. En particulier, nous allons calculer des bornes sur la substitution et sur les dérivations nécessaires à la création de l'une des plus petites attaques (s'il en existe au moins une) du protocole donné.

Pour cela, on rappelle qu'une attaque (π, σ) est minimale quand $\sum_{x \in Var} |\sigma(x), Init|_{dag}$ est minimale, et que tout protocole (bien formé) admettant au moins une attaque admet également une attaque minimale. Nous allons borner ces attaques minimales.

3.2.1 Taille de termes.

Nous utiliserons essentiellement dans les preuves la définition de taille DAG et les propriétés correspondantes données au chapitre précédent. Néanmoins, nous avons besoin d'un lemme supplémentaire, pour pouvoir plus tard borner les messages reçus et envoyés par l'intrus ($R\sigma$) en fonction de la spécification du protocole (R) et de la substitution utilisée (σ).

Lemme 3.2.1.1 *Taille d'un terme substitué.*

Soient E un ensemble de termes, x une variable, et t un message. Alors $|E[x \leftarrow t]|_{dag} \leq |E, t|_{dag}$.

Preuve. Soient E, x, et t définis comme ci-dessus. On rappelle que $|t, t|_{dag} = |t|_{dag}$. L'idée est de définir une fonction f de $STermes(E[x \leftarrow t])$ dans $STermes(E, t)$ et de montrer que f est une injection. Pour tout u sous terme de $E[x \leftarrow t]$, on pose :

- $f(u) = u$ si u est un sous terme de t, et
- $f(u) = v$ si $u = v[x \leftarrow t]$ avec v sous terme de E (même si v ne contient pas la variable x). Quand plusieurs termes v sont possibles, on en choisit un arbitrairement.

Montrons à présent que f est une bijection. Soient u et v sous termes de $E[x \leftarrow t]$ tels que $u \neq v$. On a :

- Si u et v sont des sous termes de t, alors $f(u) = u \neq v = f(v)$.
- Si u est sous terme de t mais $v = v'[x \leftarrow t]$ avec v' sous terme de E, alors $u \neq v'$ car $u[x \leftarrow t] = u \neq v = v'[x \leftarrow t]$, et donc $f(u) \neq f(v)$.
- Si $u = u'[x \leftarrow t]$ et $v = v'[x \leftarrow t]$ avec u' et v' sous termes de E, alors $u' \neq v'$ puisque $u'[x \leftarrow t] \neq v'[x \leftarrow t]$, et donc $f(u) \neq f(v)$.

Ainsi, f est bien une injection et en conséquence, $\#STermes(E[x \leftarrow t]) \leq \#STermes(E, t)$ ce qui prouve le lemme. □

On peut alors étendre ce lemme à n'importe quelle substitution close, en l'appliquant successivement pour chaque variable de la substitution :

Corollaire 3.2.1.2 *Taille d'un terme substitué.*

Soient E un ensemble de termes, et γ une substitution close avec $Var = \{x_1, .., x_n\}$. Alors :

$$|E\gamma|_{dag} \leq |E, \gamma(x_1), .., \gamma(x_n)|_{dag}$$

3.2.2 Borner pour pouvoir énumérer

Pour pouvoir donner un algorithme NP pour l'insécurité de protocoles, nous devons :

- D'une part borner polynomialement la taille de la substitution que l'algorithme doit choisir. Cette substitution plus un ordre d'exécution devra former une attaque.
- D'autre part être capable de prouver en temps polynomial que la substitution et l'ordre d'exécution choisis forment réellement une attaque, c'est à dire construire toutes les dérivations suivies par l'intrus pour forger les messages réclamés par les principaux.

Nous allons faire cela en deux temps. La section suivante va borner les tailles des dérivations nécessaires pour construire un terme (clos) donné, c'est-à-dire un message attendu par un principal instancié par une substitution close. Puis nous bornerons la taille des substitutions nécessaires à la création d'une attaque minimale. Enfin, ces deux résultats ensembles nous permettrons de détailler un algorithme NP de décision du problème de l'insécurité face à l'intrus de Dolev-Yao dans notre modèle de protocole.

3.3 Borner les dérivations

Nous allons utiliser dans cette section la définition de dérivation bien formée pour borner les dérivations nécessaires à la création d'une attaque, en montrant que toute dérivation bien formée est de taille bornée par son but et son origine. Tout d'abord, montrons que l'on peut utiliser cette définition :

Définition 3.3.0.1 *Dérivation minimale.*
Soient E un ensemble de messages et t un message tels que $t \in forge_{DY}(E)$. On note $Deriv_t(E)$ l'une des dérivations partant de E, de but t, avec l'intrus de Dolev-Yao, minimale en longueur parmi toutes ces dérivations.

Nous allons borner ces dérivations minimales, en montrant tout d'abord qu'elles n'utilisent que des messages sous termes de E ou de t (dérivations bien formées).

Lemme 3.3.0.2 *Décomposition dans une dérivation minimale.*
Soient E un ensemble de messages, t et t' deux messages, et $L \in L_d(t') \cap Deriv_t(E)$. Alors t' est un sous terme de E.

Preuve. Posons $D = Deriv_t(E) = E \rightarrow_{L_1} ... \rightarrow_{L_n} E, t_1, .., t_n$, avec $t = t_n$. On va simplement réaliser une récurrence croissante sur $i \geq 1$. On veut démontrer que (H_i) pour tout $i \in 1..n$, si $\exists t'$ t.q. $L_i \in L_d(t')$ alors t' est un sous terme de E. Pour $i = 1$, on a nécessairement $t' \in E$ et donc H_1 est vraie. Pour $i > 1$, supposons que H_j soit vrai pour tout $j < i$. On a alors seulement deux cas :

- Soit $t' \in E$, et donc H_i est vrai puisque $E \subset STermes(E)$.
- Soit il existe une règle d'intrus L_j, $j < i$, générant le terme t' dans la dérivation D. Par minimalité de D, on a $L_c(t') \cap D = \emptyset$ (sinon, l'une des deux règles $L_c(t')$ ou $L_d(t')$ serait inutile). Ainsi, L_j est nécessairement une décomposition, avec $L_j \in L_d(t'')$ et t' sous terme de t''. Or par hypothèse de récurrence, on a t'' sous terme de E, ce qui prouve H_i.

Au final, on a bien H_i vrai pour tout $i \in 1, .., n$, ce qui prouve le lemme. □

Lemme 3.3.0.3 *Composition dans une dérivation minimale.*
Soient E un ensemble de messages, t un message, et $L \in L_c(t') \cap Deriv_t(E)$. Alors t' est un sous terme de E ou de t.

PREUVE. Soit $D = Deriv_t(E)$. Par minimalité de D, et comme dans le lemme précédent, on a $L_d(t') \cap D = \emptyset$. Nous avons cette fois trois cas à considérer :

- Soit $t' \in E, t$ et le lemme est alors prouvé. Sinon, il existe au moins un règle dans D utilisant le terme t'(sinon, L est inutile et la dérivation n'est pas minimale), ce qui donne les deux cas suivants :
- Soit il existe un message a tel que $L_d(\{a\}^p_{t'*}) \cap D \neq \emptyset$ ou $L_d(\{a\}^s_{t'}) \cap D \neq \emptyset$. Dans ce cas, $\{a\}^p_{t'*}$ ou $\{a\}^s_{t'}$ est nécessairement un sous terme de E grâce au lemme précédent, et donc t' aussi.
- Soit il existe un message t_1 tel que $L_c(t_1) \cap D \neq \emptyset$ et t_0 sous terme de t_1, avec $t_0 = t'$. Il nous suffit alors d'itérer ce raisonnement sur t_1 (comme pour le lemme précédent), pour trouver un terme t_n ayant t_0 comme sous terme et vérifiant l'un des deux points précédents, i.e. t_n sous terme de E ou t, ce qui prouve le lemme.

□

Ces deux lemmes nous permettent de prouver l'une des propositions centrales de cette section, à savoir qu'il existe toujours des dérivations bien formées.

Proposition 3.3.0.4 *L'intrus de Dolev-Yao est bien formé.*
Soient un ensemble de messages E et un message t tels que $t \in forge_{DY}(E)$. Alors $Deriv_t(E)$ est une dérivation bien formée partant de E et de but t.

PREUVE. Soient E et t vérifiant ces hypothèses. Tout d'abord, on sait déjà que toute règle de l'intrus de Dolev-Yao est soit une règle d'intrus normalisé de composition, soit une règle d'intrus normalisé de décomposition (il suffit de regarder la structure de ces règles). Ainsi, le premier point de la définition de dérivation bien formée (i.e. les règles sont des règles d'intrus normalisé) est toujours vérifié. De plus, les deux lemmes précédents prouvent directement[9] les deux derniers points de la définition de dérivation bien formée, et la proposition est donc vérifiée. □

Ceci nous permet, à partir de maintenant, de ne plus considérer que des dérivations bien formées minimales pour la recherche d'attaques, et en particulier $Deriv_t(E)$. Ces dérivations ont une structure suffisamment régulière pour pouvoir être bornée, à la fois sur leur longueur et sur la taille des messages intermédiaires :

Proposition 3.3.0.5 *Borne sur les dérivations bien formées.*
Soient E un ensemble de messages et t un message, et soit $D = E \rightarrow_{L_1} ... \rightarrow_{L_n} E, t_1, .., t_n$ une dérivation bien formée partant de E et de but $t = t_n$ pour l'intrus de Dolev-Yao. Alors $n \leq |E, t|_{dag}$ et pour tout $i \in \{1, .., n\}$, $|t_i|_{dag} \leq |E, t|_{dag}$.

PREUVE. Tout d'abord, par définition de dérivation bien formée, on sait que pour tout $i \in \{1, .., n\}$, t_i est un sous terme de E ou t. Ainsi, on a nécessairement $|t_i|_{dag} \leq |E, t|_{dag}$. Ensuite, comme par définition une dérivation ne peut pas créer deux fois le même message, tous les t_i sont différents, et donc $n \leq |\{t_i\}|_{dag} \leq |E, t|_{dag}$ puisque $t_i \in STermes(E, t)$ pour tout i. □

[9]Remarque : pour les décomposition $L_d(t'_i)$, le terme généré t_i est sous terme propre de t'_i

Une autre propriété de certaines dérivations utile pour la recherche d'attaques est décrite par la proposition suivante : à l'aide de quelques conditions, il est possible de choisir des dérivations ne décomposant jamais un message donné.

Proposition 3.3.0.6 *Dérivations ne décomposant pas un message donné.*

Soient E un ensemble de messages et t, u deux messages, tels que $t \in forge_{DY}(E)$, $u \in forge_{DY}(E)$, et tels que la dernière règle d'intrus de $Deriv_u(E)$ soit une règle de composition. Alors il existe une dérivation D de but t, partant de E, telle que $L_d(u) \cap D = \emptyset$.

Preuve. Soient E, t, et u vérifiant ces hypothèses. Posons $D_1 = E \rightarrow_{L_1} ... \rightarrow_{L_p} E, t_1, .., t_p$ la dérivation $Deriv_u(E)$ sans sa dernière règle (i.e. $Deriv_u(E)$ est D_2 suivi par une règle dans $L_c(u)$). On suppose que $t \notin \{t_1, .., t_p\}$, car sinon on obtient simplement la dérivation D en tronquant la dérivation D_1. Notons de plus $D_2 = E \rightarrow_{L'_1} ... \rightarrow_{L'_n} E, t'_1, .., t'_n$ la dérivation $Deriv_t(E)$. L'idée est de concaténer ces deux dérivations pour en construire une nouvelle ne décomposant pas u (puisque tous ses sous termes propres maximaux[10] sont dans $E, t_1, .., t_p$) et de but t. Soit f une fonction injective croissante de $\{1, .., n'\}$ dans $\{1, .., n\}$, pour $n' \leq n$, telle que pour tout $i \in \{1, .., n\}$, $t'_i \notin \{t_1, .., t_p\}$ ssi i est dans l'image de f. Ceci nous permet d'éliminer de la dérivation D_2 toutes les règles devenues inutiles quand $\{t'_1, .., t'_p\}$ est ajouté à E. On obtient la dérivation suivante :

$$D = E \rightarrow_{L_1} ... \rightarrow_{L_p} E, t_1, .., t_p \rightarrow_{L'_{f(1)}} ... \rightarrow_{L'_{f(n')}} E, t_1, .., t_p, t'_{f(1)}, .., t'_{f(n')}$$

Cette suite d'applications de règles d'intrus est bien une dérivation, puisque l'on a retiré toutes les règles de D_2 devenues inutiles. De plus, comme $t_p = u$, elle ne contient aucune règle générant u, et donc aucune règle de $L_d(u)$. Enfin, $t'_{f(n')} = t$ puisque t n'est pas dans $\{t_1, .., t_p\}$, et la proposition est prouvée. □

3.4 Borner les substitutions des attaques minimales

Dans le but de borner les tailles des substitutions utilisées dans les attaques minimales, nous allons montrer que toute substitution de ce genre est construite à l'aide de sous termes de la spécification du protocole uniquement. Pour cela, nous allons tout d'abord montrer que tout pré terme d'une valeur d'une telle substitution est un sous terme de la spécification du protocole (caractérisation des pré termes), puis nous montrerons que cela suffit pour borner les substitutions.

Pour toute la suite, nous supposerons fixé un protocole $P = (\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}\}, <_{\mathcal{I}}, S_0)$ dont il existe au moins une attaque. On suppose sans perte de généralité que les connaissances initiales de l'intrus S_0 ne sont pas vides. On note $\mathcal{SP}$ l'ensemble des sous termes de P, i.e. $\bigcup_{\iota \in \mathcal{I}} STermes(R_\iota) \cup STermes(S_\iota)$. De plus, dans toute attaque (π, σ) sur P, on verra toujours π comme une bijection de $\mathcal{J} \subset \mathcal{I}$ sur $\{1, .., \#\mathcal{I}\}$, et en conséquence on notera $R_i \Rightarrow S_i$ le pas de protocole $R_{\pi^{-1}(i)} \Rightarrow S_{\pi^{-1}(i)}$ pour $i \in \{1, .., \#\pi\}$, avec $\#\pi = \#\mathcal{J}$, à partir du moment où une attaque (π, σ) sera définie. Le taille du protocole P étant le nombre de sous termes des messages le constituant, on a $|P|_{dag} = |\mathcal{SP}|_{dag}$ $(= \#\mathcal{SP})$. Pour que la notion de taille de protocole soit cohérente avec la taille DAG des termes qui le composent, on suppose que $\#\mathcal{I} \leq |P|_{dag}$. De toute façon, une représentation concrète (classique) de P doit au moins décrire $\mathcal{I}$.

[10] Maximaux pour la relation sous terme.

3.4.1 Caractériser les pré-termes des substitutions

Nous allons tout d'abord introduire un lemme technique pour préparer la propriété clef de cette section. Ce lemme stipule qu'à partir du moment où l'intrus utilise une valeur de substitution (d'une attaque) dans un message envoyé à un principal, il est capable de la construire à partir de ses connaissances.

Lemme 3.4.1.1 *Les valeurs des substitutions sont connues de l'intrus.*
Soient (π, σ) une attaque sur P, $x \in Var$, et $N \in \{1, .., \#\pi\}$ tels que $\sigma(x) \in STermes(R_N\sigma)$ mais $\sigma(x) \notin STermes(E)$ pour $E = S_0, S_1\sigma, .., S_{N-1}\sigma$. Alors $\sigma(x) \in forge_{DY}(E)$.

Preuve. Soit $Deriv_{R_N\sigma}(E) = E \rightarrow_{L_1} E, t_1 \rightarrow_{L_2} ... \rightarrow_{L_n} E, t_1, .., t_n$. Posons $i \in \{1, .., n\}$ le plus petit indice tel que $\sigma(x)$ soit un sous terme de t_i. Cet indice existe car $\sigma(x)$ est au moins un sous terme de $t_n = R_N\sigma$. On n'a que deux possibilités :

- Soit $\sigma(x) = t_i$, et donc $\sigma(x) \in forge_{DY}(E)$ ce qui prouve le lemme,
- Soit $\sigma(x)$ est un sous terme propre de t_i. Mais dans ce cas, on sait que L_i est une règle d'intrus normalisé de composition ou de décomposition (puisque l'intrus Dolev-Yao est un intrus normalisé), et dans les deux cas les sous termes propres de t_i sont sous termes de $E, t_1, .., t_{i-1}$. Cela contredit la minimalité de i puisque $\sigma(x)$ n'est pas un sous terme de E.

□

On peut à présent prouver le lemme clef de cette section, à savoir que toute valeur d'une substitution dans une attaque minimale possède un pré terme (non trivial) qui soit sous terme de la spécification du protocole. Ceci est la caractérisation désirée des pré termes des substitutions.

Lemme 3.4.1.2 *Caractérisation des pré termes de substitutions.*
Pour toute attaque minimale (π, σ) de P, et pour toute variable $x \in Var$, il existe $t \sqsubseteq_\sigma \sigma(x)$ tel que $t \in \mathcal{SP}$.

Preuve. Nous allons effectuer une preuve par contradiction. Supposons qu'il existe $x \in Var$ telle que :

$$(*) \ : \quad \text{Pour tout terme } t \text{ tel que } t \sqsubseteq_\sigma \sigma(x),\ t \notin \mathcal{SP}$$

Comme tous les atomes sont dans $\mathcal{SP}$, on a nécessairement $\sigma(x) \notin Atomes$. Supposons que $\sigma(x)$ soit un sous terme de $S_i\sigma$, pour $i \in \{1, .., \#\pi\}$. Alors nécessairement, il existe une variable $y \in Var$ dans S_i telle que $\sigma(x)$ soit un sous terme de $\sigma(y)$ (avec peut-être $x = y$). Par exemple, $\sigma(x)$ n'est pas sous terme de S_i car $\sigma(x) \notin \mathcal{SP}$. Or la définition de protocole cryptographique, dans le modèle par rôles, nous assure qu'il existe $j < i$ tel que y soit une variable de R_j. En conséquence, il existe toujours au moins un indice i tel que $\sigma(x)$ soit sous terme de $R_i\sigma$ (puisque $\sigma(x)$ n'est pas sous terme de $S_0 \subseteq \mathcal{SP}$). On définit N_x comme étant le plus petit indice tel que $\sigma(x) \in STermes(R_{N_x}\sigma)$. En résumé, $\sigma(x)$ est un sous terme de $R_{N_x}\sigma$ mais n'est pas un sous terme de $S_0, S_1\sigma, .., S_{N_x-1}\sigma$. On peut donc utiliser le lemme 3.4.1.1 sur $\sigma(x)$, ce qui nous donne :

$$\sigma(x) \ \in \ forge(S_0, S_1\sigma, .., S_{N_x-1}\sigma)$$

Posons $\delta = [\sigma(x) \leftarrow Init]$ le remplacement de $\sigma(x)$ par n'importe quel atome $Init \in S_0$. Comme (π, σ) est une attaque, on a pour tout j :

$$R_j\sigma \ \in \ forge_{DY}(S_0,\ S_1\sigma,\ ..,\ S_{j-1}\sigma)$$

et l'on veut montrer que ces relations sont toujours vraies quand on applique δ à chaque terme. Pour cela, posons $\sigma' = \sigma\delta$. On distingue deux cas :

– Si $j < N_x$, alors en vertu de la minimalité de N_x, $\sigma(x)$ n'est pas un sous terme de $R_j\sigma$ ou de $S_0, S_1\sigma, .., S_{j-1}\sigma$. Ces termes sont donc invariants par application de δ, et donc :

$$R_j\sigma' \in forge_{DY}(S_0, S_1\sigma', .., S_{j-1}\sigma') \text{ pour tout } j < N_x$$

– Sinon, $j \geq N_x$, et la dernière règle d'intrus de la dérivation $Deriv_{\sigma(x)}(S_0, S_1\sigma, .., S_{j-1}\sigma)$ est nécessairement une règle de composition, car sinon le lemme 3.3.0.2 contredirait la minimalité de N_x en donnant $\sigma(x)$ sous terme de S_0, $S_1\sigma$, .., $S_{j-1}\sigma$. Ainsi, on peut utiliser la proposition 3.3.0.6 avec $t = R_j\sigma$ et $u = \sigma(x)$. On obtient alors une dérivation D partant de S_0, $S_1\sigma$, .., $S_{j-1}\sigma$, de but $R_j\sigma$, qui ne décompose jamais $\sigma(x)$. On peut donc remplacer dans toute cette substitution toutes les occurrences de $\sigma(x)$ par $Init$ tout en conservant la validité des applications de règles. Ainsi, on obtient :

$$(R_j\sigma)\delta \in forge_{DY}(S_0, (S_1\sigma)\delta, .., (S_{j-1}\sigma)\delta)$$

De plus, comme aucun sous terme de $R_j, S_0, .., S_{j-1}$ n'est un pré terme de $\sigma(x)$, on obtient :

$$R_j\sigma' \in forge_{DY}(S_0, S_1\sigma', .., S_{j-1}\sigma') \text{ pour tout } j \geq N_x$$

On obtient donc que (π, σ') est une attaque sur P (au même titre que (π, σ)). Cependant, σ' est obtenu à partir de σ en remplaçant $\sigma(x)$ non atomique par $Init$ atomique. En conséquence, on a une contradiction avec la minimalité de l'attaque (π, σ), et le lemme est prouvé. □

3.4.2 Borner les substitutions d'attaques minimales

Nous allons maintenant utiliser le lemme clef de caractérisation des pré termes des substitutions (des attaques minimales) pour borner entièrement les tailles des valeurs de ces substitutions. L'idée est d'itérer ce lemme sur les valeurs d'une substitution de manière à la décomposer en sous termes de la spécification du protocole.

Théorème 3.4.2.1 *Borne sur les valeurs des substitutions.*
Soit (π, σ) une attaque minimale sur P. Alors pour tout $x \in Var$, on a $|\sigma(x)|_{dag} \leq |P|_{dag}$.

Preuve. On rappelle que pour tout ensemble de variables U, $U\sigma$ (ou $\sigma(U)$) représente l'ensemble $\{\sigma(y) \,|\, y \in U\}$. Choisissons une variable $x \in Var$ en particulier, dont on va borner la valeur. Pour cela, nous allons construire par induction une séquence d'ensembles $E_p \subseteq \mathcal{SP}$, et une séquence d'ensembles de variables $V_p \subseteq Var$ tendant vers $\emptyset$, telles que $|\sigma(x)|_{dag} \leq |E_p, V_p\sigma|_{dag}$ pour tout indice p. Ces deux séquences sont construites de la manière suivante :

– On pose $(E_0, V_0) = (\emptyset, \{x\})$. C'est le point de départ de ces deux séquences. On a naturellement $|\sigma(x)|_{dag} \leq |E_0, V_0\sigma|_{dag}$, et $E_0 \subseteq \mathcal{SP}$.

– Supposons à présent que nous ayons construit (E_p, V_p) vérifiant les deux propriétés :

$$|\sigma(x)|_{dag} \leq |E_p, V_p\sigma|_{dag} \text{ et } E_p \subseteq \mathcal{SP}$$

Nous définissons E_{p+1} et V_{p+1} de la manière suivante, si $V_p \neq \emptyset$:

– Choisissons $x' \in V_p$. Il existe un terme t tel que $t \sqsubseteq_\sigma \sigma(x)$ et $t \in \mathcal{SP}$, grâce au lemme 3.4.1.2.

– On pose alors $V_{p+1} = Var(t) \cup V_p \backslash \{x'\}$ et $E_{p+1} = E_p \cup \{t\}$.

Comme $t \in \mathcal{SP}$, on a bien $E_{p+1} \subseteq \mathcal{SP}$. De plus, en appliquant le corollaire 3.2.1.2 avec la substitution $\delta = [y \leftarrow \sigma(y) \mid y \in Var(t)]$ à E_p, $V_p\sigma$, on obtient :

$$|E_p, V_p|_{dag} \leq |E_p\delta, V_p\sigma|_{dag} = |E_p\delta, (V_p\sigma)\delta|_{dag} \leq |E_p, V_p\sigma, Var(t)\delta|_{dag}$$

d'où :

$$|\sigma(x)|_{dag} \leq |E_p, V_p|_{dag} \leq |E_p, t, (Var(t) \cup V_p \backslash \{x\})\sigma, |_{dag} = |E_{p+1}, V_{p+1}\sigma|_{dag}$$

Ainsi, on a bien construit E_{p+1} et V_{p+1} tels que $|\sigma(x)|_{dag} \leq |E_{p+1}, V_{p+1}\sigma|_{dag}$ et $E_{p+1} \subseteq \mathcal{SP}$.

De plus, cette construction termine. En effet, $\sum_{y \in V_p} |\sigma(y)|_{dag}$ décroît strictement à chaque pas, puisque l'on remplace $\sigma(x)$ par un sous ensemble strict de ses sous termes (t n'est pas réduit à une variable par définition). A la fin, on a donc $V_p = \emptyset$ et $|\sigma(x)|_{dag} \leq |E_p|_{dag}$ avec $E_p \subseteq \mathcal{SP}$, d'où $|\sigma(x)|_{dag} \leq |P|_{dag}$ et le théorème est prouvé. □

La conséquence du théorème 3.4.2.1 est que tout message envoyé ou reçu par un principal lors d'une attaque minimale est borné (en taille DAG) par un polynôme en la taille du protocole. Ceci est crucial pour la recherche d'attaques, car cela permet de ne rechercher que des attaques bornées *à priori* par un polynôme en la taille du protocole. En effet, on a :

Corollaire 3.4.2.2 *Borne sur les messages d'une attaque minimale.*
Soit (π, σ) une attaque minimale sur P. Alors pour tout $i \in \{1, .., \#\pi\}$, on a :

$$|R_i\sigma, S_0\sigma, .., S_{i-1}\sigma|_{dag} \leq |P|_{dag} \text{ et } |Secret, S_0\sigma, .., S_{\#\pi}\sigma|_{dag} \leq |P|_{dag}$$

Preuve. Il nous suffit de modifier très légèrement la preuve du théorème 3.4.2.1, en partant de $E_0 = \{R_i, S_0, .., S_{i-1}\}$ et $V_0 = Var$. En effet, la preuve du théorème avec cet état initial nous donne $|R_i, S_0, .., S_{i-1}, \sigma(Var)|_{dag} \leq |P|_{dag}$ pour tout i, et comme $|R_i\sigma, S_0\sigma, ...S_{i-1}\sigma|_{dag} \leq |R_i, S_0, .., S_{i-1}, \sigma(Var)|_{dag}$, on obtient le résultat pour tout $i \in \{1, .., \#\pi\}$. De même pour $Secret$. □

3.5 Algorithme NP pour le problème de l'insécurité

Nous avons maintenant à notre disposition deux types de bornes : d'une part, tous les messages transmis lors d'une attaque minimale sont bornés par la taille de la spécification de protocole, et d'autre part toute dérivation servant à créer un message $R_i\sigma$ à partir de $S_0\sigma, .., S_{i-1}\sigma$ est bornée par la taille de $R_i\sigma, S_0\sigma, .., S_{i-1}\sigma$ et donc par la taille de la spécification de protocole. De plus, tout protocole admettant une attaque en admet une minimale. Il nous suffit donc de décrire un algorithme NP permettant de trouver une attaque minimale, à partir du moment où il en existe au moins une. Ceci nous permettra donc de certifier que le protocole est sûr si aucune attaque n'est trouvée. En vérifiant l'exactitude de l'attaque choisie, on aura un algorithme juste et exhaustif. Le détail de l'algorithme est donné dans la Figure 3.1. Pour simplifier les notations, on suppose que :

$$\boxed{R_{\#\pi+1} = Secret\ .}$$

On sait déjà que si le protocole P admet une attaque, alors il est possible de choisir, dans cet algorithme, l'ordre d'exécution et la substitution d'une attaque minimale de P. De plus, si $n = |P|_{dag}$:

1. Choisir un ordre d'exécution π sur P.
2. Choisir une substitution σ telle que pour tout $x \in Var$, $|\sigma(x)|_{dag} \leq |P|_{dag}$
4. Pour chaque $i \in \{1, .., \#\pi + 1\}$, tester si $R_i\sigma \in forge_{DY}(S_0\sigma, .., S_{i-1}\sigma)$.
5. Si toutes ces vérifications sont réussies, alors répondre VRAI.

FIG. 3.1: Procédure de décision NP pour l'insécurité face à l'intrus de Dolev-Yao.

- L'ordre d'exécution π peut être choisi en temps polynomial, car ce n'est qu'une fonction injective partielle de $\mathcal{J} \subset \mathcal{I}$ dans $\{1, .., \#\mathcal{J}\}$ avec $\#\mathcal{I} \leq n$.
- Une représentation DAG de la substitution σ peut être choisie en temps polynomial en n, puisque pour tout $x \in Var$, $|\sigma(x)|_{dag} \leq n$.
- Pour tout $i \in \{1, .., \#\pi + 1\}$, posons $\mathcal{F}_i = forge_{DY}(E_i) \cap STermes(E_i, R_i\sigma)$ avec $E_i = S_0\sigma, .., S_{i-1}\sigma$. On sait déjà que pour tout i, $R_i\sigma \in forge_{DY}(E_i)$ ssi $R_i\sigma \in \mathcal{F}_i$, grâce à la proposition 3.3.0.4. On doit donc construire les ensembles $\mathcal{F}_i$ en temps polynomial en n. On pourra alors tester si $R_i\sigma \in \mathcal{F}_i$ pour chaque i, ce qui se fait en temps polynomial en n puisque $|\mathcal{F}_i|_{dag}$ est polynomial en n.

Il nous reste donc à vérifier que l'on peut construire les ensembles $\mathcal{F}_i$ en temps polynomial en n. Comme $\#\pi + 1 \leq n$, il suffit de pouvoir construire chaque ensemble en temps polynomial en n. De plus, on sait que la taille de $R_i\sigma, S_0\sigma, .., S_{i-1}\sigma$ est bornée polynomialement par n. Il nous suffit donc d'avoir la proposition suivante :

Proposition 3.5.0.3 *Construction de $Forge(E) \cap STermes(E, t)$.*

On peut construire $Forge(E) \cap STermes(E, t)$ en temps polynomial en $|E, t|_{dag}$, avec E un ensemble de messages et t un message.

PREUVE. Posons $F = Forge(E) \cap STermes(E, t)$, et $n = |E, t|_{dag}$. On remarque tout d'abord que F est nécessairement de taille inférieure à n. De plus, pour tout message $u \in F$, il existe une dérivation partant de E, de but u, et dont tous les termes intermédiaires sont sous termes de u ou de E, donc dans F. Ainsi, F n'est rien d'autre que la clôture de E par les règles d'intrus construisant des sous termes de E ou t, c'est à dire la clôture de E par les règles $E' \rightarrow_{DY} E', u$ avec u sous terme de E ou t. A chaque pas, on peut trouver en temps polynomial en n un terme u de ce type à ajouter, puisque chaque test $v \rightarrow_{DY} u$ prend un temps au pire linéaire en n. Et comme on ne peut effectuer qu'au plus n pas, on obtient un algorithme construisant F en temps polynomial en n. □

Plus généralement, nous venons en fait de démontrer la propriété suivante :

Corollaire 3.5.0.4 *Le problème $DERIVE$ est polynomial face à l'intrus DY.*

Soit $DERIVE = \{ (t, E) \mid t \in forge_{DY}(E) \}$, avec E ensemble de messages et t un message. Alors l'appartenance à $DERIVE$ est décidable en temps polynomial en $|E, t|_{dag}$.

En effet, il suffit de tester si $t \in Forge(E) \cap STermes(E,t)$. Ainsi, l'algorithme de la Figure 3.1 est bien dans NP. Finalement, s'il n'existe aucun moyen pour cet algorithme de répondre VRAI, alors il n'existe aucune attaque minimale au protocole, et donc celui-ci est sûr. On a donc bien un algorithme de décision NP pour le problème de l'insécurité de protocoles cryptographiques face à l'intrus de Dolev-Yao, d'où le théorème suivant :

Théorème 3.5.0.5 *Le problème de l'insécurité de protocoles, face à l'intrus de Dolev-Yao, est NP-complet.*

3.6 Extensions directes

Ce résultat de complexité peut être étendu de nombreuses manières. Alors que les chapitres suivants vont donner des résultats de complexité équivalents pour des intrus bien plus complexes que celui de Dolev-Yao (xor, exponentielle, etc ...), nous allons voir dans cette section comment réutiliser directement les résultats précédents pour obtenir des algorithmes NP pour l'insécurité dans deux cas relativement proches. Le premier sera celui de l'intrus involutif. Le second présentera une manière d'introduire des points de choix conditionnels (structure CASE ou IF) dans la spécification de protocole.

3.6.1 Cryptage involutif

Nous allons utiliser ici l'intrus involutif décrit à la Table 2.2. Pour mémoire, il s'agit d'un intrus de Dolev-Yao étendu à l'aide des règles de déduction :

$$L_s(\{\{a\}_b^s\}_b^s): \quad \{\{a\}_b^s\}_b^s \to a \qquad L_r(\{\{a\}_b^s\}_b^s): \quad a \to \{\{a\}_b^s\}_b^s$$

$$L_s(\{\{a\}_K^p\}_{K^*}^p): \quad \{\{a\}_K^p\}_{K^*}^p \to a \qquad L_r(\{\{a\}_K^p\}_{K^*}^p): \quad a \to \{\{a\}_K^p\}_{K^*}^p$$

Cet intrus est intéressant, car ses règles de déduction correspondent à des calculs simples en pratiques, et il permet de fabriquer de nouvelles attaques par rapport à l'intrus de Dolev-Yao. En particulier, nous avons vu à la section 2.3.6 un exemple de protocole sûr face à l'intrus de Dolev-Yao mais attaquable face à l'intrus involutif.

Le principal problème de cet intrus par rapport à l'intrus de Dolev-Yao est que ces règles ne sont pas des règles d'intrus normalisé. Néanmoins, nous allons tout de même pouvoir prouver que (comme Dolev-Yao) cet intrus est bien formé, i.e. les dérivations bien formées sont suffisantes. Une fois ceci démontré, toutes les preuves des sections précédentes fonctionneront de la même manière pour cet intrus. Pour cela, nous avons le lemme suivant :

Lemme 3.6.1.1 *Dérivations guidées.*

Soient E un ensemble de messages et t un message tels que $t \in forge_{Inv}(E)$. Alors il existe une dérivation $D = E \to_{L_1} .. \to_{L_n} E, t_1, .., t_n$ avec $t = t_n$, vérifiant les conditions suivantes pour tout i :

$$(*): \begin{cases} Si\ L_i \in L_s(\{\{a\}_b\}_{b^*}), & alors\ \forall j < i,\ L_j \notin L_c(\{\{a\}_b\}_{b^*})\ et\ L_j \notin L_c(\{a\}_b) \\ Si\ L_i \in L_r(\{\{a\}_b\}_{b^*}), & alors\ \forall j > i,\ L_j \notin L_d(\{\{a\}_b\}_{b^*})\ et\ L_j \notin L_d(\{a\}_b) \\ & pour\ \{..\}_{..}^p\ ou\ pour\ \{..\}_{..}^s\ avec\ b^* = b \end{cases}$$

en confondant $L_c(t)$ ou $L_d(t)$ avec l'unique règle qu'elles représentent.

PREUVE. Soit D une dérivation minimale (en longueur) partant de E et de but t. Nous allons construire une dérivation D' partant également de E, construisant *les mêmes termes* que D dans *le même ordre*, et vérifiant les conditions $(*)$ et minimale en longueur parmi les dérivations vérifiant $(*)$ de E vers t. Pour cela, on va raisonner par induction sur la longueur de D. Si $D = \emptyset$ (i.e. dérivation de longueur nulle, sans applications de règle d'intrus) alors $D' = D$ convient. Sinon, on a par hypothèse d'induction :

$$\begin{aligned} D = &\quad E \rightarrow_{L_1} ... \rightarrow_{L_{n-1}} E, t_1, .., t_{n-1} \rightarrow_{L_n} E, t_1, .., t_n \\ D'_1 = &\quad E \rightarrow_{L'_1} ... \rightarrow_{L'_{n-1}} E, t_1, .., t_{n-1} \end{aligned}$$

On veut alors étendre D'_1 en lui ajoutant une règle L'_n pour obtenir une dérivation D' construisant t_n tout en satisfaisant les conditions $(*)$. Rappelons que par définition d'une dérivation, on ne peut pas créer deux fois (i.e. avec deux règles différentes) le même terme. Pour cela, on a plusieurs cas :

1. Si $L_n = L_s(\{\{a\}_b\}_{b^*})$, alors $L_r(\{\{a\}_b\}_{b^*}) \notin D'_1$ car sinon $t_n = a \in E, t_1, .., t_{n-1}$, et on a trois cas :
   - S'il existe i tel que $L'_i = L_c(\{\{a\}_b\}_{b^*})$, alors grâce à L'_i on a $\{\{a\}_b, b^*\} \subseteq E, t_1, .., t_{n-1}$, et donc $L'_n = L_d(\{a\}_b)$ construit $t_n = a$.
   - S'il existe i tel que $L'_i = L_c(\{a\}_b)$, alors $t_n = a \in E, t_1, .., t_{i-1}$ ce qui est impossible.
   - Sinon, $L'_n = L_n$ construit t_n.

2. Si $L_n = L_r(\{\{a\}_b\}_{b^*})$, alors $L_s(\{\{a\}_b\}_{b^*}) \notin D'_1$ car sinon $t_n = \{\{a\}_b\}_{b^*} \in E, t_1, .., t_{n-1}$, et on a trois cas :
   - S'il existe i tel que $L'_i = L_d(\{\{a\}_b\}_{b^*})$, alors $t_n = \{\{a\}_b\}_{b^*} \in E, t_1, .., t_{n-1}$ ce qui est impossible.
   - S'il existe i tel que $L'_i = L_d(\{a\}_b)$, alors grâce à L'_i on a $\{\{a\}_b, b^*\} \subseteq E, t_1, .., t_{n-1}$, et donc $L'_n = L_c(\{\{a\}_b\}_{b^*})$ convient.
   - Sinon, $L'_n = L_n$ convient.

3. Si $L_n = L_c(\{\{a\}_b\}_{b^*})$ et s'il existe i tel que $L'_i = L_s(\{\{a\}_b\}_{b^*})$, alors $t_i = \{\{a\}_b\}_{b^*} = t_n$ ce qui est impossible.

4. Si $L_n = L_c(\{a\}_b)$ et s'il existe i tel que $L'_i = L_s(\{\{a\}_b\}_{b^*})$, alors nécessairement $\{\{a\}_b\}_{b^*}$ et b sont dans $E, t_1, .., t_{n-1}$, et donc $L'_n = L_d(\{\{a\}_b\}_{b^*})$ construit $t_n = \{a\}_b$.

5. Si $L_n = L_d(\{\{a\}_b\}_{b^*})$ et s'il existe i tel que $L'_i = L_r(\{\{a\}_b\}_{b^*})$, alors nécessairement $\{a, b\} \subseteq E, t_1, .., t_{n-1}$ et donc $L'_n = L_c(\{a\}_b\}$ construit $t_n = \{a\}_b$.

6. Si $L_n = L_d(\{a\}_b)$ et s'il existe i tel que $L'_i = L_r(\{\{a\}_b\}_{b^*})$, alors $t_n = a \in E, t_1, .., t_{i-1}$ ce qui est impossible.

Dans tous ces cas, la dérivation D' obtenue en ajoutant L'_n à D'_1 vérifie la condition$(*)$ par construction, satisfait l'hypothèse d'induction. On peut donc itérer, et la propriété est vraie pour toute dérivation D, ce qui prouve le lemme. □

A l'aide de ce lemme, il devient possible de donner équivalent de la proposition 3.3.0.4, pour une définition de *Deriv* un peu modifiée :

Définition 3.6.1.2 *Dérivation minimale.*

Soient E un ensemble de messages et t un message tels que $t \in forge_{Inv}(E)$. On note $Deriv_t(E)$ l'une des dérivations partant de E, de but t, avec l'intrus involutif, vérifiant les conditions $(*)$, et minimale en longueur parmi toutes ces dérivations.

Le lemme précédent permet d'assurer qu'une telle dérivation existe. On obtient alors :

Proposition 3.6.1.3 *L'intrus involutif est bien formé.*

Soient un ensemble de messages E et un message t tels que $t \in forge_{Inv}(E)$. Alors $Deriv_t(E)$ est une dérivation bien formée partant de E et de but t.

La preuve de ce lemme (ainsi que les autres propriétés des sections précédentes) est fondamentalement la même que pour l'intrus de Dolev-Yao, en remplaçant L_c par $L_c \cup L_r$ et L_d par $L_d \cup L_s$, le lemme 3.6.1.1 permettant d'éliminer les combinaisons problématiques de règles. On obtient donc :

Théorème 3.6.1.4 *Le problème de l'insécurité de protocoles, face à l'intrus involutif, est NP-complet.*

3.6.2 Points de choix (structure CASE)

On peut également étendre le modèle de protocole de manière à permettre l'utilisation de points de choix. Par exemple, le contenu d'un message peut contenir une information sur le type de cryptage utilisé pour le reste de la session, et donc les pas de protocoles utilisés peuvent varier en fonction du contenu de ce message.

Notre définition de protocoles permet déjà de faire des points de choix, mais ils sont non déterministes (i.e. personne ne sait à l'avance quelle choix le principal va faire), ou de manière équivalente le choix est déterminé par l'intrus (puisque l'on explore tous les cas). Ici, on veut que le choix soit conditionné par des conditions de priorité entre plusieurs règles. Par exemple, si le principal doit choisir entre deux pas $A_1 : R_1 \Rightarrow S_1$ et $A_2 : R_2 \Rightarrow S_2$ (non comparables selon $<_{\mathcal{I}}$), au moment de recevoir un message m, il prendra A_1 s'il existe une substitution σ telle que $R_1\sigma = m$, et prendra A_2 sinon. On va donc ajouter des priorités sur les pas d'un protocole :

Définition 3.6.2.1 *Pas de protocole avec priorité.*

Étant donné un protocole $P = (\{R_\iota \Rightarrow S_\iota \mid \iota \in \mathcal{I}\}, <_{\mathcal{I}}, S_0)$, une priorité $\ll$ sur les pas de P est un ordre partiel sur $\mathcal{I}$.

Cet ordre partiel $\ll$ va être utilisé différemment de $<_{\mathcal{I}}$. Il permet de créer des sous ensemble de $\mathcal{I}$ totalement ordonnés par $\ll$ tels que si un principal exécute un pas de l'un de ces ensembles, alors aucun autre pas plus petit (selon $\ll$) n'aurait pu être exécuté à sa place. De cette manière, on ne modifie pas la définition de protocole existante, mais on lui ajoute des contraintes supplémentaires. Cette définition est un peu laxiste dans la mesure où elle permet aussi de définir des protocoles non exécutables (si $\ll$ et $<_{\mathcal{I}}$ se contredisent, notamment), mais une définition plus stricte ne faciliterait de toute façon pas les preuves et donnerait le même résultat. Ainsi, une exécution avec priorité est une exécution (π, σ) de P compatible avec une priorité $\ll$ donné, i.e. pour tout $i \in \{1, .., \#\pi\}$, pour tout $\iota \ll \pi^{-1}(i)$, et pour toute substitution close γ, on a $R_\iota\gamma \neq R$ (i.e. le choix ι n'était pas possible). Toutes les définitions qui en découlent, comme celle d'attaque, restent inchangées modulo l'utilisation d'exécutions avec priorité. De plus, une donnée initiale du problème de l'insécurité sera à présent un protocole P et une priorité associée $\ll$.

Pour faciliter les preuves, on suppose l'existence dans le protocole P à vérifier d'un atome $Init_x$ pour chaque variable $x \in Var$. Les atomes de cet ensemble $\{Init_x \mid x \in Var\}$, de même cardinal que Var, ne doivent pas apparaître dans les pas de protocole de P, mais sont initialement connus par l'intrus (pour tout $x \in Var$, $Init_x \in S_0$). De plus, on impose que dans toute attaque, $Init_x$ ne puisse apparaître *que* dans $\sigma(x)$, pour tout $x \in Var$. Ceci n'est en rien restrictif puisque l'intrus n'est pas tenu d'utiliser $Init_x$. Enfin, la notion de taille DAG ne tenant pas compte de $Init$, on demande également qu'elle ne tienne pas compte de $\{Init_x\}_{x \in Var}$. Ceci non plus n'a pas d'importance, car $\#Var \leq |P|_{dag}$.

On peut à présent examiner les preuves des lemmes et propriétés déjà énoncées. Tout d'abord, toutes les preuves concernant les dérivations sont toujours valides, puisque l'intrus n'a pas changé. En fait, la seule preuve où la modification des ordres d'exécutions intervient est celle du lemme 3.4.1.2 de caractérisation des pré termes. On doit vérifier que la nouvelle attaque (π, σ') construite à partir de (π, σ) est bien une attaque (et une exécution avec priorités). Supposons tout d'abord qu'au lieu de remplacer $\sigma(x)$ par $Init$ dans cette preuve, on remplace $\sigma(x)$ par $Init_x$ pour tout $x \in Var$. C'est strictement équivalent, puisque $Init$ et $Init_x$ sont deux atomes de S_0. On a donc une nouvelle substitution $\sigma' = \sigma\delta$ avec $\delta = [\sigma(x) \leftarrow Init_x]$. Vérifions que (π, σ') est compatible avec la priorité $\ll$ par contradiction. Suppose qu'il existe $i \in \{1, .., \#\pi\}$, $\iota \ll \pi^{-1}(i)$, et une substitution close γ tels que $R_\iota\gamma = R_i\sigma'$. On a alors deux cas :

- Si x apparaît dans R_i, alors $Init_x$ est sous terme de $R_i\sigma'$ et de $R_\iota\gamma$. Alors $R_\iota(\gamma[Init_x \leftarrow \sigma(x)]) = R_i(\sigma'[Init_x \leftarrow \sigma(x)]) = R_i\sigma$ puisque $Init_x$ n'apparaît pas dans R_i, et on a donc une contradiction avec la compatibilité de (π, σ) avec la priorité.
- Sinon, x n'apparaît pas dans R_i, et donc $R_i\sigma = R_i\sigma' = R_\iota\delta$ ce qui donne la même contradiction.

On a donc montré que (π, σ') est bien une exécution avec priorité de P, et le reste de la preuve du lemme 3.4.1.2 reste inchangé.

De cette manière, toutes le bornes sur les dérivations et sur les substitutions des sections précédentes sont toujours valides, et on obtient :

Théorème 3.6.2.2 *Le problème de l'insécurité de protocoles avec priorité, face à l'intrus de Dolev-Yao, est NP-complet.*

3.7 Conclusion.

Nous avons présenté dans ce chapitre un algorithme NP pour l'insécurité de protocoles cryptographiques face à l'intrus de Dolev-Yao et face à l'intrus involutif. De plus, nous avons généralisé un peu ces preuves pour permettre l'utilisation de points de choix dans la spécification des protocoles (i.e. un équivalent des structures CASE en programmation). La justification de cet algorithme présente une structure bien précise : on borne les dérivations nécessaires à l'intrus pour construire un message attendu par un principal, et on borne les substitutions nécessaires à la création d'une attaque. Cette idée sera conservée et largement développée dans les chapitres suivants pour calculer la complexité de ce même problème d'insécurité, mais face aux intrus xor, DH, et EC (i.e. avec les opérateurs algébriques définis au chapitre 2).

4

Insécurité avec xor

Sommaire

Nous avons vu aux chapitres 1 et 3 différentes méthodes de décision du problème de la sécurité de protocoles cryptographiques, et plus particulièrement à nombre de sessions borné pour le chapitre 3, avec une étude de la complexité de ce problème. Certaines de ces méthodes formelles ont permis à des outils de découvrir de nouvelles failles sur des protocoles connus (voir par exemple [8, 27, 57, 72, 90]).

La plupart des analyses formelles de protocoles cryptographiques à nombre borné de sessions [4, 16, 1, 82, 70, 56] ou avec d'autres restrictions [43, 15, 45, 32, 3] ignorent les propriétés de "bas niveau" des opérateurs qu'elles utilisent, comme par exemple les propriétés de chaînage de l'encryption par blocs (Cipher Bloc Chaining). C'est l'hypothèse de chiffrement parfait, appliqué à tous les opérateurs du protocole. Cependant, de nombreux protocoles utilisent des opérateurs (algébriques) dont on ne peut pas ignorer les propriétés. Par exemple, P. Ryan et S. Schneider présentent dans [84] une attaque assez simple sur un protocole d'authentification récursive. Ce protocole distribue une chaîne de clefs liant tous les agents de l'initiateur de protocole jusqu'à un serveur. Cependant, si une clef est compromise alors à cause des propriétés du xor, toutes les autres clefs le sont également. En revanche, L. Paulson a montré à l'aide du démonstrateur Isabelle que si l'opérateur xor est vu comme un symbole libre, alors cette distribution de clefs est sûre (c.f. [75]). Il existe assez peu de procédures de décision pour les protocoles cryptographiques munis d'opérateurs algébriques. Dans le cas du xor, H. Comon et V. Shmatikov présentent dans [34] une procédure de décision DEXPTIME pour un modèle un peu plus général que le modèle par rôles, mais dont les protocoles supplémentaires qu'il peut traiter ne peuvent pas à priori être réalisés en pratique.

Dans ce chapitre, nous allons étudier la complexité du problème de l'insécurité de protocole face aux intrus xor et préfixe, deux extensions de l'intrus de Dolev-Yao, dans notre modèle (et donc à nombre de sessions borné). Pour regrouper au maximum les preuves communes à ces deux intrus, nous allons définir une notion d'oracle, i.e. un ensemble particulier de règles d'intrus possédant toutes les propriétés nécessaires à nos preuves. Après avoir démontré que les deux intrus xor et préfixe définissent bien des règles d'oracle, nous ferons abstraction des intrus eux mêmes et donnerons des bornes sur les attaques minimales faces à n'importe quel intrus définis par des règles d'oracle. Nous en déduirons un algorithme NP pour le problème de l'insécurité face à ces intrus.

Dans tout ce chapitre, nous n'utiliserons jamais les opérateurs particuliers produit, exponentielle et encryption commutative. En revanche, nous utiliserons pleinement l'opérateur xor (et bien sûr les opérateurs standards de Dolev-Yao). En conséquence, les protocoles seront supposés spécifiés sans les opérateurs produit, exponentielle, et encryption commutative, de même que pour tous les termes et messages considérés, et donc toutes les attaques recherchées.

4.1 Oracle

4.1.1 Définition d'oracle et intérêt

La structure de ce chapitre est la suivante : Nous allons tout d'abord présenter une notion d'oracle, c'est-à-dire une classe de règles d'intrus possédant des propriétés intéressantes. Puis nous montrerons que les intrus xor et préfixe satisfont aux conditions d'oracles. Il est d'ailleurs tout-à-fait raisonnable de penser que d'autres intrus non mentionnés ici et tout aussi utiles que préfixe satisfont également à ces conditions. Nous montrerons enfin que pour tout intrus satisfaisant aux conditions d'oracle, on peut décider du problème de l'insécurité de protocole en temps NP. Comme on sait déjà que ce problème est NP-difficile (puisque les codages de 3-SAT du chapitre 3 sont toujours valides), ceci prouvera la NP-complétude de ce problème. On définit donc, avec la convention de notation $\mathcal{L} \in D$ ssi $\exists L \in \mathcal{L}$ t.q. $L \in D$:

Définition 4.1.1.1 *Règles d'oracle.*

Soit un intrus (normalisé) $\mathcal{L}$ disposant des règles d'intrus normalisé $L_c \cup L_d \cup L_{oc} \cup L_{od}$ avec $L_c \cup L_d$, L_{oc}, et L_{od} disjoints, L_{oc} des règles d'intrus normalisé de composition, et L_{od} des règles d'intrus normalisé de décomposition. Alors $\mathcal{L}$ est un oracle (ou dispose de règles d'oracle) ssi :

1. *Pour tous E et t, si $t \in forge_{\mathcal{L}}(E)$ alors il existe une dérivation bien formée (construite sur $\mathcal{L}$) partant de E et de but t.*
2. *Pour tous E, a et t, si $E \rightarrow_{L_{oc}} E, t$ et $E, t \rightarrow_{L_d(t)} E, t, a$, alors il existe une dérivation D partant de E et de but a telle que $L_d(t) \notin D$.*
3. *Il existe une fonction $\epsilon : Terme \backslash Atomes \rightarrow \ulcorner Terme \urcorner$ telle que pour tout message non atomique u, $|\epsilon(u)|_{dag} < |\ulcorner u \urcorner|_{dag}$, et pour tout ensemble fini de messages F et un message t, si $F \rightarrow_{L_c \cup L_{oc}} F, u$ (i.e. u peut être composé en un pas) et si $F, u \rightarrow_{L_{oc} \cup L_{od}} F, u, t$ (i.e. t n'est pas créé par DY), alors $\ulcorner t[u \leftarrow \epsilon(u)] \urcorner \in forge_{\mathcal{L}}(\ulcorner F[u \leftarrow \epsilon(u)] \urcorner)$ et $\epsilon(u) \in forge_{\mathcal{L}}(F)$.*

La première condition, déjà utilisée dans le chapitre précédent, va nous permettre de borner la longueur et la nature des dérivations dont on a besoin pour prouver la validité d'une attaque. Les condition 2 et 3 quant à elles vont nous permettre d'effectuer des remplacements d'un terme par un autre plus petit, à la manière du chapitre précédent où on remplaçait $\sigma(x)$ par $Init$. Ces deux conditions nous permettront donc de borner les tailles des substitutions de certaines attaques.

L'intérêt de cette définition est de rassembler toutes les propriétés dont on aura besoin pour prouver la NP-complétude du problème de l'insécurité face à un intrus donné. Ainsi, on aura une preuve commune à tous les intrus vérifiant ces trois conditions. En contre-partie, nous devons commencer par montrer qu'il existe des intrus intéressants vérifiant ces conditions. C'est le but des deux sections suivantes.

4.1.2 Les règles xor sont des règles d'oracle

On veut montrer ici que les règles de l'intrus xor sont des règles d'oracle. Nous allons commencer par examiner la première condition de la définition 4.1.1.1. Pour alléger les notations, nous noterons $\mathcal{L}_c(t) = L_c(t) \cup L_{xc}(t)$ pour tout terme t, $\mathcal{L}_c = \bigcup_t \mathcal{L}_c(t)$, et $\mathcal{L}_d = L_d \cup \bigcup_t L_{xd}(t)$. On n'utilise pas la notation $\mathcal{L}_d(t)$, car dans $L_d(t)$, t désigne le terme décomposé, alors que dans $L_{xd}(t)$, t désigne le terme créé : il y aurait confusion entre les deux. En revanche, dans la notation $\mathcal{L}_c(t)$, t désigne bien le terme généré (par l'une de ces règles). De plus, on notera $L_x = L_{xc} \cup L_{xd}$. Toutes les règles d'intrus considérées dans ce chapitre seront toujours celles de l'intrus xor, i.e. $\mathcal{L}_d \cup \mathcal{L}_c$ et on omettra donc de le rappeler en permanence dans les énoncés.

L'intrus xor est bien formé.

Tout d'abord, nous donnons une condition suffisante pour qu'une dérivation soit bien formée :

Lemme 4.1.2.1 *Condition suffisante sur les dérivations.*
Soit $D = E \rightarrow_{L_1} ... \rightarrow_{L_n} E, t_1, .., t_n$ une dérivation telle que :

1. *Pour tout $i \in \{1, .., n\}$ tel que $L_i \in \mathcal{L}_d$, soit il existe $j < i$ tel que t_i soit un sous terme de t_j avec $L_j \in \mathcal{L}_d$, soit t_i est un sous terme de E.*
2. *Pour tout $i \in \{1, .., n-1\}$ tel que $L_i \in \mathcal{L}_c$, il existe $j > i$ tel que t_i soit un sous terme de $E \cup \{t_j\}$.*

Alors D est une dérivation bien formée.

Preuve. Le premier point nous permet immédiatement de réaliser une récurrence (décroissante) sur $i \in \{1, .., n\}$ tel que $L_i \in \mathcal{L}_d$. Ainsi, pour tout $i \in \{1, .., n\}$ tel que $L_i \in \mathcal{L}_d$, on obtient t_i sous terme de E.

Le second point nous permet également de réaliser une induction, sur $n - i$ pour $i \in \{1, .., n\}$ cette fois, pour montrer que si $L_i \in \mathcal{L}_c$ alors t_i est sous terme de $E \cup \{t_n\}$. Si $n - i = 0$, i.e. $i = n$, alors t_n est bien sous terme de $E \cup \{t_n\}$. Et pour le pas de l'induction, le point 2 implique qu'il existe $j > i$ tel que t_i soit un sous terme de $E \cup \{t_j\}$. On a alors deux cas : soit $L_j \in \mathcal{L}_d$, et grâce au point n°1 on a t_i sous terme de E, soit $L_j \in \mathcal{L}_c$ et par induction t_j (et donc t_i) est sous terme de $E \cup \{t_n\}$.

Ainsi, on a bien prouvé les points 2 et 3 de la définition de dérivations bien formées. De plus, on a déjà le point 1 puisque l'intrus xor est un intrus normalisé, et le lemme est prouvé. □

On peut à présent montrer que les règles d'intrus xor admettent des dérivations bien formées.

Proposition 4.1.2.2 *L'intrus xor est bien formé.*
Soient E un ensemble fini de messages normalisés et g un message normalisé, tels que $g \in forge_{xor}(E)$. Alors il existe une dérivation bien formée partant de E et de but g (construite sur $\mathcal{L}_{xor}$).

PREUVE. Soit $D = E \rightarrow_{L_1} E, t_1 \rightarrow_{L_2} ... \rightarrow_{L_n} E, t_1, .., t_n$ une dérivation de but $g = t_n$ et minimale en longueur parmi toutes les dérivations partant de E et de but g. Nous allons montrer que D vérifie les points 1 et 2 du lemme 4.1.2.1.

Tout d'abord, on constate que si $F \rightarrow_{L_x} F, t \rightarrow_{L_x} F, t, u$ alors $F \rightarrow_{L_x} F, u$. En effet, par définitions des règles L_x, u est une somme normalisée d'éléments de F, t et t est lui-même une somme normalisée d'éléments de F. En conséquence, u est une somme normalisée d'éléments de F, et on peut donc le construire directement.

Grâce à cette propriété, on peut supposer sans perte de généralité que les termes de D utilisés comme membre gauche d'une règle xor (L_x) ne sont jamais générés par une règle xor. Formellement, on a $(*)$: pour tout $i \in \{1, .., n\}$ tel que $L_i = M \rightarrow t_i \in L_x$, il n'existe pas de $j < i$ tel que $L_j \in L_x$ et $t_j \in M$. On peut à présent prouver les points 1 et 2 du lemme 4.1.2.1. Pour tout $i \in \{1, .., n\}$:

1. Si $L_i \in L_d(s)$, alors pour tout $j < i$ tel que $L_j \in L_{xc} \cup L_c$, on a $s \neq t_j$ car les règles L_{xc} ne créent que des termes non standard et par définition des dérivations. En conséquence, soit $s \in E$, soit il existe $j < i$ tel que $s = t_j$ et $L_j \in \mathcal{L}_d$ ce qui prouve le point 1 dans ce cas car t_i est sous terme de s.
   Si $L_i \in L_{xd}$, alors t_i est standard, et par $(*)$ et la définition de L_{xd}, il existe $t' \in E, t_1, .., t_{i-1}$ non standard avec t_i sous terme de t' et $L_j \notin L_{xc}(t')$ pour tout j. On a deux cas. Si $t' \in E$, alors le point est prouvé. Sinon, il existe $j < i$ tel que $t' = t_j$. Comme t' est non standard, on a $L_j \notin L_c \cup L_{xd}$, et grâce à $(*)$ $L_j \notin L_{xc}$. Il ne reste donc que $L_j \in L_d$ et le point est prouvé.
2. Si $L_i \in L_c$, avec $i < n$, alors t_i est standard, et par minimalité de D, il existe $j > i$ tel que t_i appartienne au membre gauche de L_j. De plus, par définition d'une dérivation on a $L_j \notin L_d$. Si $L_j \in L_c$, alors t_i est sous terme de t_j et le point est prouvé. Sinon, $L_j \in L_x$, alors comme t_i est standard, soit t_i est un facteur (et donc un sous terme) de t_j, soit il existe $t \in E, t_1, .., t_{j-1}$ non standard avec t_i sous terme de t (i.e. t est utilisé pour simplifier t_i). Or $(*)$ impose que t ne soit pas généré par une règle de L_x, et comme il est non standard il ne peut pas non plus être généré par une règle de L_c. En conséquence, soit $t \in E$ et le point est prouvé, soit t est généré par une règle de L_d et le point 1 donne également $t \in E$.
   Si $L_i \in L_{xc}$, avec $i < n$, alors $(*)$ impose que soit $t_i = t_n$ (impossible), soit il existe $j > i$ tel que $L_j \in L_c$ et t_i sous terme de t_j, ce qui prouve le point 2.

Ainsi, les conditions du lemme 4.1.2.1 sont remplies, ce qui prouve la proposition. □

L'intrus xor est un oracle.

A présent que l'on a montré la première condition de la définition d'oracle (i.e. l'intrus xor permet l'utilisation de dérivations bien formées), il ne nous reste (plus) qu'à prouver les autres conditions de la définition 4.1.1.1 Voici le théorème correspondant :

Théorème 4.1.2.3 *Les règles de l'intrus xor sont des règles d'oracle.*

PREUVE. On va examiner chaque condition séparément :

1. Condition 1. C'est une conséquence directe de la proposition 4.1.2.2.
2. Condition 2. Aucun terme t généré par une règle L_{xc} ne peut être décomposé par une règle de L_d, car il est non standard.

3. Condition 3. On pose $\epsilon(u) = 0$ pour tout message u. Soient u un message non atomique, F un ensemble de messages t.q. $0 \in F$, et t un message tels que :

$$F \rightarrow_{\mathcal{L}_c} F, u \text{ et } F, u \rightarrow_{L_x} F, u, t$$

On a évidement $\epsilon(u) = 0 \in forge_{xor}(F)$. Posons $\delta = [u \leftarrow 0]$ le remplacement de u par $\epsilon(u)$. On a trois cas :
- Si $u = t$, alors $t\delta = 0 \in forge_{xor}(F)$.
- Si $u \neq t$ mais u est une paire ou une encryption, alors par définition des règles xor, on a

$$\ulcorner F\delta,\, u\delta \urcorner \rightarrow_{L_x} \ulcorner F\delta,\, u\delta,\, t\delta \urcorner$$

- Si $u \neq t$ mais u est non standard, alors nécessairement $F \rightarrow_{L_{xc}} F, u$ avec $u = t_1 \oplus ... \oplus t_n$ et pour tout $i \in \{1, .., n\}$, $t_i \in F$. Ainsi, $F \backslash u \rightarrow_{L_x} F, t$ (i.e. on peut utiliser $t_1, .., t_n$ pour construire t au lieu d'utiliser u), ce qui donne $\ulcorner F\delta \urcorner \rightarrow_{L_x} \ulcorner F\delta,\, t\delta \urcorner$.

□

On a donc montré que l'intrus xor définit bien un ensemble de règles d'oracle. De plus, pour pouvoir appliquer correctement l'algorithme NP que nous présenterons à la fin de ce chapitre, nous avons besoin de savoir que toute application d'une règle de l'oracle xor peut être vérifiée en temps polynomial. Pour cela :

Proposition 4.1.2.4 *Les règles xor sont applicables en temps polynomial.*
Le problème de savoir si $E \rightarrow_{L_x} E, t$ pour un ensemble fini normalisé de messages E et un message t peut être décidé en temps polynomial en $|E, t|_{dag}$.

Preuve. Soit B l'ensemble des facteurs des termes de E et S l'ensemble des facteurs de t. B et S peuvent être calculés en temps polynomial, et on peut aussi décider en temps polynomial si $S \subseteq B$. On a deux cas :
- Si $S \not\subseteq B$, alors il est impossible de construire t à partir de E avec une règle de L_x, car il manque un ou plusieurs facteurs de t.
- Sinon on peut représenter t par ses facteurs $Facteurs(t) \subseteq B$, et cet ensemble peut être représenté par un vecteur de longueur $\#B$ sur $\{0, 1\}$, où chaque coefficient 1 ou 0 représente la présence ou l'absence d'un élément de B comme facteur de t. On peut voir ce vecteur comme un élément d'un espace vectoriel de dimension $\#B$ à coefficients dans $\{0, 1\}$, et de la même manière chaque message de E peut être représenté par un élément de cet espace vectoriel. A présent, décider si $E \rightarrow_{L_x} E, t$ équivaut à décider si le vecteur représentant t peut être représenté comme une combinaison linéaire des vecteurs représentant E. Or ceci est décidable en temps polynomial par une élimination de Gauss.

□

4.1.3 Les règles Préfixe sont aussi des règles d'oracle

On veut à présent montrer que l'intrus Préfixe définit aussi des règles d'oracle. Même si cet intrus ne tire pas partie de l'opérateur xor, il possède des règles dont la structure s'adapte bien à l'oracle que l'on a défini. Ceci nous permet de montrer à moindre coût que le problème de l'insécurité face à l'intrus préfixe est NP-complète, et justifie pleinement l'utilisation d'une définition commune d'oracle. Nous allons à nouveau commencer par montrer que les règles préfixe permettent l'utilisation de dérivations bien formées :

Proposition 4.1.3.1 *L'intrus préfixe est bien formé.*

Soient E un ensemble de messages et t un message tels que $t \in forge_{Prefixe}(E)$. Alors il existe une dérivation bien formée partant de E et de but t (construite sur les règles de l'intrus préfixe).

PREUVE. On sait déjà que les règles de l'intrus préfixe sont des règles d'intrus normalisé. Soit $D = E \rightarrow_{L_1} ... \rightarrow_{L_n} E, t_1, .., t_n$ une dérivation (partant de E, de but t_n). Nous allons construire une nouvelle dérivation D' partant de E et de but t_n. Pour cela, soit D' la dérivation obtenue en appliquant itérativement le système de déduction suivant sur D, où les règles de déduction sont utilisées avec l'ordre de priorité décroissant de la première à la quatrième :

1. S'il existe $i < j$ tels que $L_i \in L_c(\{M'\}^s_K)$ et $L_j \in L_{prefix}(\{M'\}^s_K)$ avec $t_j = \{M\}^s_K$ et $M' = \langle .. \langle M, M_1 \rangle ..., M_n \rangle$, alors on remplace L_j par une séquence de décompositions de M' à M suivi d'une encryption $L_c(\{M\}^s_K)$. De plus, on élimine toutes les applications redondantes de règles, de manière à obtenir une dérivation. Dans cette opération, le nombre total de règles L_{prefix} diminue strictement.

2. S'il existe $i < j$ tels que $L_i \in L_{prefix}(\{M\}^s_K)$ et $L_j \in L_{prefix}(t_i)$, alors on remplace L_j par la règle préfixe $\{M\}^s_K \rightarrow t_j$. Dans cette opération, le nombre de règles L_{prefix} ne change pas, mais la taille de l'argument de la règle L_{prefix} modifiée ici augmente strictement (il est borné par le terme de taille maximale de cette dérivation).

3. S'il existe $i < j$ tels que $L_i \in L_{prefix}(t_i)$ décomposant $\{M\}^s_K$, et $L_j \in L_d(t_i)$, alors on remplace L_j par une décomposition de $\{M\}^s_K$, $L_d(\{M\}^s_K)$, suivi d'une séquence de décompositions de M (avec L_{p1}) jusqu'à avoir t_j. De plus, on élimine toutes les applications redondantes de règles, de manière à obtenir une dérivation. Dans cette opération, les règles préfixes ne changent pas (ni leur nombre, ni leurs arguments), mais le nombre de règles $L_d(t)$ telles qu'il existe k avec $t_k = t$ et $L_k \in L_{prefix}$ diminue strictement. En effet, grâce au point n°2, on a pour tout $k < i$, $L_k \notin L_{prefix}$ ou $t_k \neq \{M\}^s_K$, et la définition de dérivation nous donne le résultat pour $k \geq i$.

4. Enfin, on élimine toute règle L_i, $i < n$, telle que pour tout $j > i$, le membre de gauche de L_j ne contienne pas t_i (i.e. on enlève toutes les règles devenues inutiles).

Ce système de déduction termine, car le triplet

$$(NbPrefix,\ TailleArgs,\ NbDec,\ LongDeriv)$$

avec $NbPrefix$ le nombre de règles préfixe dans la dérivation,
et $TailleArgs = \sum_{L_{prefix}(t) \in D} |t_{max}|_{dag} - |t|_{dag}$ pour $t_{max} = \max_i |t_i|_{dag}$,
et $NbDec$ le nombre de règles L_d décrites au point 3 ci dessus,
et $LongDeriv$ la longueur de la dérivation

diminue strictement à chaque pas pour l'ordre lexicographique. On a donc une dérivation D' (pas nécessairement unique) partant de E, de but t_n, et sur laquelle aucune des quatre règles précédentes ne peut être appliquée. Il nous reste à montrer que D' est bien formée. Supposons que $D' = E \rightarrow_{L'_1} ... \rightarrow_{L'_m} E, t'_1, .., t'_m$.

Soit $i \in \{1, .., m\}$ tel que $L'_i \in L_d(s)$. Alors soit $s \in E$, soit il existe $j < i$ tel que $t_j = s$. Mais dans ce cas, $L_j \notin L_c$ par définition d'une dérivation, et $L_j \notin L_{prefix}$ grâce au point 3 ci dessus.

Ainsi, $L_j \in L_d$ nécessairement, et l'on peut itérer sur j, avec t_i sous terme de t_j. On obtient donc s sous terme de E (l'itération est finie car $i > 0$).

Soit $i \in \{1, .., m\}$ tel que $L'_i \in L_c \cup L_{prefix}$. Cette fois, on va raisonner par induction sur $m - i$. Si $m - i = 0$, alors $t_i = t_n$ et donc t_i est sous terme de E, t_n. Sinon, il existe $j > i$ tel que t_i appartienne au membre gauche de L'_j (grâce au point 4 ci dessus). On a $L'_j \notin L_d(t_i)$, car soit $L'_i \in L_c$ et on aurait une contradiction avec la définition de dérivation, soit $L'_j \in L_{prefixe}$ et on aurait une contradiction avec le point 1 ci dessus. De plus, $L'_j \notin L_{prefix}$ grâce aux points 1 et 2 ci dessus. Il ne reste donc que $L'_j \in L_c$, avec t_i sous terme de t_j. Par induction, on obtient donc t_i sous terme de E, t_n.

Ainsi, tous les points de la définition de dérivation bien formée sont vérifiés par D', ce qui prouve la proposition. □

Nous pouvons à présent montrer que l'intrus préfixe définit bien un ensemble de règles d'oracle :

Théorème 4.1.3.2 *Les règles de l'intrus préfixe sont des règles d'oracle.*

Preuve. On va examiner chaque condition séparément : □

1. Condition 1 : C'est une conséquence immédiate de la proposition 4.1.3.1.
2. Condition 2 : Si $F \rightarrow_{L_{prefixe}} F, \{M\}^s_K \rightarrow_{L_d(\{M\}^s_K)} F, \{M\}^s_K, M$, alors on peut effectuer (sur cette petite dérivation) le même remplacement de $L_d(\{M\}^s_k)$ que celui effectué à la règle de déduction n°3 de la preuve de la proposition 4.1.3.1. On obtient alors une dérivation partant de F, de but M, et sans $L_d(\{M\}^s_K)$.
3. Condition 3 : Soit u un terme non atomique. On pose $\epsilon(u) = a$ si $u = \langle a, b\rangle$ et $\epsilon(u) = 0$ sinon. Soit F un ensemble (fini) de termes avec $0 \in F$ et $F \rightarrow_{L_c \cup L_{prefixe}} F, u$, et soient $M' = \langle .. \langle M, M_1\rangle, ...M_n\rangle$, $\{M\}^s_K$ deux messages tels que $\{M'\}^s_K \in F, u$ et $\{M'\}^s_k \rightarrow \{M\}^s_K \in L_{Prefixe}$. Posons $\theta = [u \leftarrow \epsilon(u)]$. On a quatre cas :
   (a) Si $u = \{M\}^s_K$, alors $\{M\}^s_K \theta = 0 \in forge_{Prefixe}(F\theta)$
   (b) Si $u = \{M'\}^s_K$, alors on a deux cas : soit $F \rightarrow_{L_c} F, u$, d'où $M', K \subseteq F$ et $\{M\}^s_K \in forge_{Prefixe}(F)$ (M déduit de M' par décompositions), soit $F \rightarrow_{L_{prefixe}} F, u$, d'où $F \rightarrow_{L_{prefixe}} F, \{M\}^s_K$ (concaténation de règles préfixes, idem règle 2 de la preuve de la proposition 4.1.3.1). Dans ces deux cas, on a bien $\{M\}^s_K \theta \in forge_{Prefixe}(F\theta)$.
   (c) Si $u = M'$, alors $\{M'\}^s_K \theta = \{M''\}^s_{K\theta} \in F\theta$ avec $M'' = \langle .. \langle M, M_1\rangle, ...M_{n-1}\rangle$ (ou $M'' = M$ si $n = 1$) et $\{M\}^s_K \theta = \{M\}^s_{K\theta}$. On a donc bien $\{M\}^s_K \theta \in forge_{Prefixe}(F\theta)$.
   (d) Sinon, u n'intervient pas dans $\{M'\}^s_k \rightarrow \{M\}^s_K$ et donc $\{M\}_K \theta \in forge_{Prefixe}(F\theta)$.

On a donc montré que l'intrus préfixe définit bien un ensemble de règles d'oracle. De plus, pour pouvoir appliquer correctement l'algorithme NP que nous présenterons à la fin de ce chapitre, nous avons besoin de savoir que toute application d'une règle de l'oracle préfixe peut être vérifiée en temps polynomial (comme pour xor). Pour cela :

Proposition 4.1.3.3 *Les règles préfixe sont applicables en temps polynomial.*

Le problème de savoir si $E \rightarrow_{L_{prefixe}} E, t$ pour un ensemble fini normalisé de messages E et un message t peut être décidé en temps polynomial en $|E, t|_{dag}$.

Preuve. Ce problème est nettement plus simple que pour l'intrus xor. En effet, si $t = \{M\}_K$ il suffit de rechercher dans E la présence d'un terme $\{\langle M, ...\rangle\}^s_K$, ce qui se fait en temps polynomial en $|E, t|_{dag}$. Si un tel terme existe, la règle est applicable, et s'il n'y a pas de terme de cette forme ou si t n'est pas une encryption symétrique, cette règle n'est pas applicable. □

4.2 Borner les substitutions des attaques minimales

Nous allons à présent travailler sur un ensemble de règles d'oracle en général, et non plus sur un intrus (xor ou préfixe) particulier. Pour toute la suite, nous allons reprendre les hypothèses et notations définies à la section 3.4 pour l'intrus DY. Rappelons les :

On suppose fixé un protocole $P = (\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}\}, <_{\mathcal{I}}, S_0)$, avec S_0 non vide, sur lequel il existe au moins une attaque. On note $\mathcal{SP}$ l'ensemble des sous termes de P, i.e. $\bigcup_{\iota\in\mathcal{I}} STermes(R_\iota) \cup STermes(S_\iota)$. De plus, dans toute attaque (π, σ) sur P, on verra toujours π comme une bijection de $\mathcal{J} \subset \mathcal{I}$ sur $\{1, .., \#\mathcal{I}\}$, et en conséquence on notera $R_i \Rightarrow S_i$ le pas de protocole $R_{\pi^{-1}(i)} \Rightarrow S_{\pi^{-1}(i)}$ pour $i \in \{1, .., \#\mathcal{J}\}$, à partir du moment où une attaque (π, σ) sera définie.

On rappelle par ailleurs que la taille du protocole P étant le nombre de sous termes des messages le constituant, on a $|P|_{dag} = |\mathcal{SP}|_{dag}$ $(= \#\mathcal{SP})$. Pour que la notion de taille de protocole soit cohérente avec la taille DAG des termes qui le composent, on suppose que $\#\mathcal{I} \leq |P|_{dag}$. De toute façon, une représentation concrète (classique) de P doit au moins décrire $\mathcal{I}$. Enfin, on suppose fixé un intrus $Oracle$ normalisé défini par un ensemble de règles d'oracle $L_{DY} \cup L_{oc} \cup L_{od}$ (avec L_{oc} des règles de composition, L_{od} des règles de décomposition, et L_{oc} disjoint de L_{od}). Toutes les règles d'intrus considérées ici seront toujours des règles d'$Oracle$, avec ϵ la fonction de $Terme \backslash Atomes \rightarrow \ulcorner Terme \urcorner$ associée.

La structure de preuve que nous allons développer est la suivante : Dans les deux premières sous sections (4.2.1 et 4.2.2), nous allons présenter trois lemmes cruciaux, portant sur les sous termes des messages transmis par l'intrus et sur les effets de bord entre remplacement et dérivation ou substitution. Ces trois lemmes nous permettront de caractériser les facteurs des substitutions des attaques minimales (sous section 4.2.3), pour les borner complètement (sous section 4.2.4).

4.2.1 Caractérisation des sous termes d'une substitution.

Le premier lemme critique est le suivant. Il nous permet d'affirmer que tout sous terme standard apparaissant dans une substitution (définissant une attaque minimale) est soit construit à partir de la spécification du protocole, soit apparaît comme sous terme d'un message reçu par un principal. A la différence du chapitre précédent, ce n'est plus immédiat, car l'intrus pourrait utiliser un message dont une partie, éliminée par normalisation, n'apparaît jamais dans les messages transmis pendant l'attaque. On montre donc que tous les sous termes (standards) d'une substitution d'une attaque normale sont "vus" tôt ou tard par un principal. Formellement :

Lemme 4.2.1.1 *Propriété des sous termes standards des substitutions.*

Soit (π, σ) une attaque minimale de P, soit $x \in Var(R_i)$ pour $i \in \{1, .., \#\pi\}$, et soit s un sous terme standard de $\sigma(x)$. Alors soit il existe $t \in \mathcal{SP}$ tel que $t \sqsubseteq_\sigma s$, soit il existe $j \leq i$ tel que $s \in STermes(\ulcorner R_j\sigma \urcorner)$.

Preuve. Supposons qu'il existe $x \in Var(R_i)$, $i \in \{1, .., \#\pi\}$, tel que pour tout $j \leq i$, $s \notin STermes(\ulcorner R_j\sigma \urcorner)$. Nous allons donc montrer qu'il existe un pré terme t de s (selon σ) qui est sous terme de la spécification de protocole. Pour cela, posons :

$$j = \min\{\, i' \mid \exists y \in Var(R_{i'}) \text{ et } s \text{ sous terme de } \sigma(y) \,\}$$

et soit y une variable de R_j telle que s soit un sous terme de $\sigma(y)$ (l'existence de j et y est donnée par i et x). On a $j \leq i$, et donc $s \notin STermes(\ulcorner R_j\sigma \urcorner)$. Posons :

$$S = \{\, t \text{ sous terme de } R_j \text{ t.q. } y \in Var(t) \text{ et } s \text{ sous terme de } \ulcorner t\sigma \urcorner \,\}$$

et choisissons l'un de ses représentants maximaux t pour l'ordre sous terme :

$$t \in S \text{ tel que } \forall t' \in S,\ t \text{ n'est pas un sous terme propre de } t'$$

Il est important de remarquer que S n'est pas vide, puisque $y \in S$. De plus, $t \neq R_j$ car sinon, s serait un sous terme de $\ulcorner R_j\sigma \urcorner$. On pose alors r l'un des plus petits sous termes (selon l'ordre sous terme) de R_j ayant t comme sous terme propre, i.e. :

$$r \in STermes(R_j) \text{ tel que } t \in stp(r) \text{ et } \forall r' \in stp(r),\ t \notin stp(r')$$

avec $stp(t) = STermes(t)\backslash\{t\}$ pour tout terme t. Intuitivement, on vient de définir une limite dans R_j où s est éliminé lors de la normalisation de $R_j\sigma$: s est sous terme de $\ulcorner t\sigma \urcorner$, mais il n'est pas sous terme de $\ulcorner r\sigma \urcorner$ alors que t est un fils direct de r selon l'ordre sous terme. Ainsi, r est nécessairement de la forme $t \oplus r_1 \oplus ... \oplus r_n$, $n \geq 1$, avec t, r_1, ..,r_n standards (puisque R_j est normalisé). De plus, s n'est pas sous terme de $\ulcorner\urcorner t\sigma \urcorner \oplus \ulcorner r_1\sigma \urcorner \oplus .. \ulcorner r_n\sigma \urcorner\urcorner$ alors que s est sous terme de $\ulcorner t\sigma \urcorner$: nécessairement, il existe $i \in \{1, .., n\}$ tel que s soit un sous terme de $\ulcorner r_i\sigma \urcorner$ (sinon s ne serait pas éliminé lors de la normalisation de r). On supposera qu'il s'agir de r_1. Soit M l'ensemble des sous termes de r_1 dont s est sous terme modulo σ et la normalisation, i.e. :

$$M \ = \ \{\, r' \in STermes(r_1) \mid s \in STermes(\ulcorner r'\sigma \urcorner) \,\}$$

De plus, soit t_s un élément minimal de M selon l'ordre sous terme, i.e. :

$$t_s \in M \text{ tel que } \forall t'_s \in M,\ t'_s \text{ n'est pas sous terme propre de } t_s$$

Grâce à la définition de protocole bien formé 2.4.4.2, point 2, et comme $y \in Var(t)$ apparaît (par définition) pour la première fois dans R_j (i.e. pour tout $i < j$, $y \notin Var(R_i)$) on a pour tout $i \in \{1, .., n\}$, $Var(r_i) \subseteq \bigcup_{k<j} Var(R_k)$. En particulier, pour toute variable $z \in Var(r_1)$, il existe $j_z < j$ tel que $z \in Var(R_{j_z})$. On a donc, par minimalité de j, que s n'est pas un sous terme de $\sigma(z)$ pour tout $z \in Var(r_1)$, et donc t_s n'est pas réduit à une variable. Ainsi, par minimalité de t_s, on a $\ulcorner t_s\sigma \urcorner = s$, car sinon un sous terme propre de t_s serait dans M. De plus, comme s est standard, t_s l'est aussi, car sinon un facteur de t_s serait dans M. Au final, on a identifié un sous terme standard t_s de R_j, i.e. $t_s \in \mathcal{SP}$, non réduit à une variable et tel que $\ulcorner t_s\sigma \urcorner = s$, i.e. $t_s \sqsubseteq_\sigma s$, ce qui prouve le lemme. □

Ce lemme nous permettra d'examiner tous les sous termes standards des substitutions d'attaques minimales. Ainsi, on montrera qu'en fait, tous les facteurs d'une telle substitution (sous ensemble des sous termes standards) admettent des pré termes dans la spécification du protocole, $\mathcal{SP}$.

4.2.2 Interactions entre remplacement et normalisation.

L'introduction d'une fonction de normalisation complique nettement la structure des messages envoyés par l'intrus (du point de vue des bornes à calculer sur les tailles des attaques), car elle crée de nombreuses interactions avec les remplacements, à la fois au niveau des dérivations et au niveau des substitutions. Pour traiter ces interactions, nous allons prouver ici deux lemmes critiques. Le premier portera sur l'interaction entre remplacement et dérivations, et le second portera sur l'interaction entre remplacement et substitution.

Pour faciliter la compréhension, nous commençons par montrer un lemme technique sur les dérivations. Ce lemme prouve que si un message γ peut être créé à partir d'un ensemble E de messages à l'aide d'une règle de composition de Dolev-Yao, i.e. en rassemblant deux messages

γ_1 et γ_2 tous deux dérivés de E par exemple, alors il est toujours possible d'éviter de devoir décomposer γ, avec une règle $L_d(\gamma)$, dans toute dérivation partant de E et de but t. L'idée est qu'une telle décomposition de γ générerait γ_1 ou γ_2, alors que l'on peut les obtenir autrement (sans décomposer γ). Formellement :

Lemme 4.2.2.1 *Éviter de décomposer un message dans une dérivation.*

Soit E un ensemble de messages (normalisés), et soient t et γ deux messages tels que $\{t, \gamma\} \subseteq forge_{Oracle}(E)$. On suppose qu'il existe une dérivation D_γ partant de E, de but γ, et finissant par l'application d'une règle de $L_c \cup L_{oc}$. Alors il existe une dérivation D partant de E, de but t, et telle que $L_d(\gamma) \cap D = \emptyset$.

Preuve. Commençons par éliminer un cas trivial : si $t \in E$ alors une dérivation vide (sans aucune application de règle) convient, et si t est généré dans D_γ alors il suffit de tronquer D_γ pour obtenir D. On suppose donc que $t \notin E$ et que D_γ ne génère pas t. Comme γ est généré à la dernière règle de D_γ, on sait par définition d'une dérivation qu'aucune règle de $L_d(\gamma)$ n'est présente dans D_γ. Notons :

$$D_\gamma \;=\; E \rightarrow_{L_1} E, t_1 \rightarrow_{L_2} ... \rightarrow_{L_n} E, t_1, .., t_n \rightarrow_{L} E, t_1, .., t_n, \gamma$$

avec $L \in L_c \cup L_{oc}$, et en permettant $n = 0$ (et donc aucune règle L_i) si γ est généré en un pas. De plus, notons :

$$Deriv_t(E) \;=\; E \rightarrow_{L'_1} E, t'_1 \rightarrow_{L'_2} ... \rightarrow_{L'_p} E, t'_1, .., t'_p$$

Nous allons rassembler les règles de D_γ privé de L et les règles de $Deriv_t(E)$ en une seule dérivation. Pour cela, posons l'ensemble d'entiers $I = \{\, i \,|\, t'_i \notin \{t_1, .., t_n\}\}$ et la fonction $f : \{1, .., \#I\} \rightarrow I$ respectant l'ordre naturel sur les entiers (i.e. $i < j$ implique $f(i) < f(j)$). On pose alors, avec $q = \#I$:

$$D' \;=\; E \rightarrow_{L_1} ... \rightarrow_{L_n} E, t_1, .., t_n \rightarrow_{L'_{f(1)}} ... \rightarrow_{L'_{f(q)}} E, t_1, .., t_n, t'_{f(1)}, .., t'_{f(q)}$$

Si $\#I = 0$, alors on n'a aucune règle L'_i dans cette définition. Alors D' est bien une dérivation, car la fonction f ne retire que les règles redondantes avec $t_1, .., t_n$ (toutes les règles $L_{f(i)}$ sont donc applicables et non redondantes). De plus, D' est une dérivation de but t, i.e. $t'_{f(q)} = t$, car $t \notin \{t_1, .., t_n\}$. On distingue deux cas, selon la nature de la règle L :

- Si $L \in L_c$, i.e. $\gamma = \langle a, b \rangle$ ou $\gamma = \{a\}^s_b$ ou $\gamma = \{a\}^p_b$, alors $L_d(\gamma) \notin D'$ puisque a et b sont nécessairement générés par D_γ. Ainsi, $D = D'$ est la dérivation prouvant le lemme dans ce cas.
- Si $L \in L_{oc}$, alors on a deux (sous) cas. Si $L_d(\gamma) \cap D'\emptyset$, alors $D = D'$ est la dérivation recherchée. Sinon, il existe k tel que $L'_{f(k)} \in L_d(\gamma)$, et on peut appliquer le point 2 de la définition de règles d'oracle 4.1.1.1 à $F \rightarrow_L F, \gamma$ et $F, \gamma \rightarrow_{L_d(\gamma)} F, \gamma, \beta$ avec $F = E, t_1, .., t_n, t'_{f(1)}, .., t'_{f(k-1)}$. Ainsi, on peut remplacer toutes les règles de $L_d(\gamma)$ de D' par des dérivations de même but mais sans règle de $L_d(\gamma)$. En éliminant (à nouveau) toutes les règles devenues inutiles (comme ci-dessus), on obtient une dérivation D partant de E, générant t, et sans aucune règle de $L_d(\gamma)$, ce qui prouve le lemme dans ce cas.

Dans ces deux cas, on obtient une dérivation D'_γ partant de E, de but t, et sans aucune règle de $L_d(\gamma)$, ce qui prouve le lemme. □

Ce premier lemme va nous permettre de montrer qu'il est possible, sous certaines conditions, de remplacer un terme s par un plus petit ($\epsilon(s)$) dans $t \in forge(E)$ tout en conservant une propriété de ce genre. Ceci nous permettra typiquement d'effectuer des remplacements d'un terme s par $\epsilon(s)$ dans une attaque tout en étant sûr que l'intrus sera toujours capable de créer les messages attendus par les principaux. Formellement, cela donne :

Lemme 4.2.2.2 *Remplacements dans des dérivations.*

Soient E et F deux ensembles de messages normalisés, avec $0 \in E \cup F$, soit t un message tel que $t \in forge_{Oracle}(E, F)$, et soit s un message non atomique tel que $s \in forge_{Oracle}(E)$ et $s \notin STermes(E)$. Soit enfin $\delta = [s \leftarrow \epsilon(s)]$. Alors $\ulcorner t\delta \urcorner \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$.

Preuve. Soit $D_s = E \rightarrow_{L_1} ... \rightarrow_{L_n} E, t_1, .., t_n \rightarrow_{L_c \cup L_{oc}} E, t_1, .., t_n, s$ une dérivation bien formée partant de E et de but s. On sait que cette dérivation finit par une règle de composition, car sinon la définition de dérivation bien formée donne une contradiction avec $s \notin STermes(E)$. De plus, pour toute règle d'intrus normalisé $M \rightarrow a$, tous les sous termes propres de a sont sous termes de M (direct à partir des définitions de règles d'intrus normalisé de composition et de décomposition). Ainsi, pour tout $i \in \{1, .., n\}$, tous les sous termes propres de t_i sont sous termes de $E, t_1, .., t_{i-1}$. De plus, s'il existe i tel que $s \in STermes(t_i)$, cela implique que $s \in STermes(E, t_1..t_{i-1})$ puisque $s \neq t_i$. Par induction sur i, on obtiendrait s sous terme de E, et donc une contradiction. Ainsi, $s \notin STermes(E, t_1, .., t_n)$, ce qui donne pour tout $i \in \{1, .., n\}$, $\ulcorner t_i\delta \urcorner = \ulcorner t_i \urcorner = t_i$ (t_i est normalisé car on utilise des règles d'intrus normalisé) et $\ulcorner t_i\delta \urcorner \in forge_{Oracle}(\ulcorner E\delta \urcorner)$.

Si $t \in E, F, t_1, .., t_n$ alors le lemme est prouvé car $\ulcorner E\delta, F\delta, t_1\delta, .., t_n\delta \urcorner \subseteq forge_{Oracle}(E\delta, F\delta)$. Sinon, soit

$$D = E, F \rightarrow_{L_1} ... \rightarrow_{L_n} E, F, t_1, .., t_n \rightarrow_{L_{n+1}} \ldots \rightarrow_{L_p} E, F, t_1, .., t_p$$

la dérivation obtenue à partir de $t \in forge_{Oracle}(E, F, t_1, .., t_n)$ et D_s grâce au lemme 4.2.2.1 (la preuve de ce lemme permet de conserver les mêmes règles $L_1, .., L_n$ que pour D_s). On a donc $t_p = t$ et pour tout $i \in \{n+1, .., p\}$, $L_i \notin L_d(s)$. Nous allons montrer par induction sur i de n à p que $\ulcorner t_i\delta \urcorner \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$. Pour $i = n$, c'est le cas précédent. Pour $i > n$, si l'on a $\ulcorner t_j\delta \urcorner \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$ pour tout $j < i$, alors :

- Si $L_i \in L_c(\langle a, b \rangle)$, alors soit $t_i = s$ et donc $\ulcorner t_i\delta \urcorner = \epsilon(s) \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$ par définition de la fonction ϵ, soit $\ulcorner t_i\delta \urcorner = \langle \ulcorner a\delta \urcorner, \ulcorner b\delta \urcorner \rangle \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$ car a et b sont dans $E, F, t_1, .., t_{i-1}$. De même pour $L_c(\{a\}_b^s)$ et $L_c(\{a\}_b^p)$.
- Si $L_i \in L_{p1}(\langle t_i, b \rangle)$, alors $s \neq \langle t_1, b \rangle$ puisque $L_i \notin L_d(s)$. Ainsi, $\ulcorner t_i\delta \urcorner \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$ puisque $\langle \ulcorner t_i\delta \urcorner, \ulcorner b\delta \urcorner \rangle \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$. De même pour L_{p2}, L_{sd} et L_{ad}.
- Si $L_i \in L_{oc} \cup L_{od}$, alors on peut utiliser le point 3 de la définition d'oracle 4.1.1.1 : $\ulcorner t_i\delta \urcorner \in forge_{Oracle}(\ulcorner E\delta, F\delta, t_1\delta, .., t_{i-1}\delta \urcorner)$ et donc $\ulcorner t_i\delta \urcorner \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$.

A la fin de l'itération, on obtient $\ulcorner t \urcorner \in forge_{Oracle}(\ulcorner E\delta, F\delta \urcorner)$, ce qui prouve le lemme. □

Il est intéressant de remarquer que dans ce lemme, la raison pour laquelle on n'effectue pas l'itération finale directement sur la dérivation donnée par le lemme 4.2.2.1 appliqué à $t \in forge_{Oracle}(E, F)$ et D_s, i.e. sans construire au préalable $t_1, .., t_n$, est due au point 3 de la définition d'oracle. En effet, ce point n'est utilisable que si s peut être créé avec une seule règle (de composition), et non avec toute une dérivation comme D_s. D'où l'ajout de D_s (moins sa dernière règle) au début de D.

On vient donc de voir que modulo quelques conditions raisonnables, $forge_{Oracle}$ est invariant par remplacement et normalisation. En particulier, cette propriété sera utilisée sur les messages construits par l'intrus, i.e. pour montrer que $\ulcorner\urcorner R_i\sigma\urcorner\delta\urcorner$ peut être créé par l'intrus à partir de $\ulcorner\urcorner E\sigma\urcorner\delta\urcorner$ quand $\ulcorner R_i\sigma\urcorner$ peut être créé à partir de $\ulcorner E\sigma\urcorner$. Cependant, on voudra au final construire une nouvelle substitution $\sigma' = \sigma\delta$ qui définisse toujours une attaque, et pour cela, on aura besoin de savoir que $\ulcorner R_i\sigma'\urcorner = \ulcorner\urcorner R_i\sigma\urcorner\delta\urcorner$. Tel est le but du lemme (critique) suivant.

Lemme 4.2.2.3 *Interaction entre remplacement et substitution.*

Soit σ une substitution close et normalisée, soit E un ensemble de messages normalisés, et soit s un message standard normalisé et non atomique. Alors, pour $\delta = [s \leftarrow \epsilon(s)]$ et $\sigma' = \sigma\delta$, s'il n'existe aucun sous terme standard t de E tel que $t \sqsubseteq_\sigma s$, on a $\ulcorner E\sigma'\urcorner = \ulcorner\urcorner E\sigma\urcorner\delta\urcorner$.

Preuve. On sait déjà que $(E\sigma)\delta = E(\sigma\delta)$, car s n'admet aucun pré terme t qui soit sous terme de E : le remplacement δ appliqué à $E\sigma$ ne concerne que les valeurs de σ. Ceci nous donne le point de départ de la preuve, à savoir $\ulcorner E\sigma'\urcorner = \ulcorner(E\sigma)\delta\urcorner$. Nous allons montrer par induction sur la structure des termes que pour tout sous terme t de E, $\ulcorner t\sigma'\urcorner = \ulcorner\urcorner t\sigma\urcorner\delta\urcorner$. En effet :

- Si t est atomique, alors $t \neq s$ par hypothèse (s est non atomique), et donc $\ulcorner t\sigma'\urcorner = t = \ulcorner\urcorner t\sigma\urcorner\delta\urcorner$.
- Si t est une variable, alors $\ulcorner t\sigma\urcorner = t\sigma$ (puisque σ est normalisée), et donc $\ulcorner t\sigma'\urcorner = \ulcorner(t\sigma)\delta\urcorner = \ulcorner\urcorner t\sigma\urcorner\delta\urcorner$.
- Si $t = \langle a, b\rangle$, alors :

$$\begin{array}{rcll} \ulcorner t\sigma'\urcorner & = & \ulcorner\langle\ulcorner a\sigma'\urcorner, \ulcorner b\sigma'\urcorner\rangle\urcorner & \text{car } \ulcorner..\urcorner \text{ est confluent.} \\ & = & \ulcorner\langle\ulcorner\urcorner a\sigma\urcorner\delta\urcorner, \ulcorner\urcorner b\sigma\urcorner\delta\urcorner\rangle\urcorner & \text{par induction sur } a \text{ et } b \\ & = & \ulcorner\langle\ulcorner a\sigma\urcorner, \ulcorner b\sigma\urcorner\rangle\delta\urcorner & \text{car } s \neq \ulcorner t\delta\urcorner \text{ (sinon } t \sqsubseteq_\sigma s) \\ & = & \ulcorner\urcorner t\sigma\urcorner\delta\urcorner & \end{array}$$

  Ce raisonnement s'applique exactement de la même manière à $t = \{a\}^s_b$ et $t = \{a\}^p_K$.
- Enfin, si $t = t_1 \oplus ... \oplus t_n$, on a :

$$\begin{array}{rcll} \ulcorner t\sigma'\urcorner & = & \ulcorner t_1\sigma' \oplus ... \oplus t_n\sigma'\urcorner & \\ & = & \ulcorner\urcorner t_1\sigma'\urcorner \oplus ... \oplus \ulcorner t_n\sigma'\urcorner\urcorner & \text{car } \ulcorner..\urcorner \text{ est confluent.} \\ & = & \ulcorner\urcorner\urcorner t_1\sigma\urcorner\delta\urcorner \oplus ... \oplus \ulcorner\urcorner t_1\sigma\urcorner\delta\urcorner\urcorner & \text{par induction sur chaque } t_i \\ & = & \ulcorner\urcorner t_1\sigma\urcorner\delta \oplus ... \oplus \ulcorner t_1\sigma\urcorner\delta\urcorner & \text{car } \ulcorner..\urcorner \text{ est confluent.} \\ & = & \ulcorner(\ulcorner t_1\sigma\urcorner \oplus ... \oplus \ulcorner t_1\sigma\urcorner)\delta\urcorner & \text{car } s \text{ est un terme standard} \\ & = & \ulcorner\urcorner t\sigma\urcorner\delta\urcorner & \end{array}$$

Ainsi, on a $\ulcorner t\sigma'\urcorner = \ulcorner\urcorner t\sigma\urcorner\delta\urcorner$ pour chaque t sous terme de E, et donc y compris pour tout $t \in E$, ce qui démontre le lemme. □

4.2.3 Caractérisation des facteurs des substitutions d'attaques minimales

Nous avons à présent suffisamment d'outils pour présenter la propriété clef de cette section. Il s'agit de démontrer que pour toute attaque minimale de substitution σ, tous les facteurs de σ (i.e. les plus grands sous termes standards) admettent en fait toujours comme pré terme un sous terme de la spécification du protocole (i.e. la "tête" de tout facteur est dans $\mathcal{SP}$). Ainsi, chaque facteur est construit sur un sous terme du protocole, ce qui permettra au théorème ? ? de borner la taille des substitutions (après avoir borné le nombre de facteurs). La preuve de cette proposition suit la même idée que dans le modèle de base. En effet, les lemmes précédents vont en fait nous permettre de gérer simplement les problèmes dus à la normalisation. Concrètement :

Proposition 4.2.3.1 *Caractérisation des facteurs d'attaques minimales.*

Soit (π, σ) une attaque minimale. Alors pour toute variable $x \in Var$ et tout facteur v_x de $\sigma(x)$, il existe un terme $t \in \mathcal{SP}$ tel que $t \sqsubseteq_\sigma v_x$.

PREUVE. Nous allons effectuer une preuve par contradiction. On suppose donc qu'il existe une variable x vérifiant la propriété suivante :

$$(*) \quad \text{Pour tout } t, \text{ si } t \sqsubseteq_\sigma v_x \text{ alors } t \notin \mathcal{SP}.$$

Nous allons obtenir une contradiction avec la notion d'attaque minimale en remplaçant v_x par $\epsilon(v_x)$ tout en conservant les propriétés d'attaque. Tout d'abord, nous allons définir le premier pas de l'attaque (π, σ) où v_x apparaît comme sous terme. On sait que v_x est standard grâce à la définition de facteur. On peut donc utiliser le lemme 4.2.1.1 avec la propriété $(*)$. En conséquence, il existe $j \in \{1, .., \#\pi\}$ tel que v_x soit un sous terme de $\ulcorner R_j\sigma \urcorner$. On peut donc poser :

$$N_x \ = \ \min \{\ j \mid v_x \text{ est un sous terme de } \ulcorner R_j\sigma \urcorner \ \}$$

De plus, supposons que v_x soit un sous terme de $\ulcorner S_i\sigma \urcorner$ pour $i \in \{1, .., N_x - 1\}$. On saurait alors, grâce à $(*)$, qu'il existe $y \in Var(S_i)$ telle que v_x soit sous terme de $\sigma(y)$. En effet, dans le cas contraire v_x serait obtenu par normalisation d'un sous terme t de $S_i\sigma$ non sous terme de σ, i.e. $t \in STermes(S_i)$ et $\ulcorner t\sigma \urcorner = v_x$. Or, par définition de protocole cryptographique, y apparaît nécessairement dans un membre gauche d'une règle de protocole précédant i, i.e. il existe $i' \leq i$ tel que $y \in Var(R_{i'})$. On peut alors à nouveau utiliser le lemme 4.2.1.1, sur v_x (encore grâce à $(*)$) et $y \in Var(R_{i'})$ cette fois. En conséquence, il existe $i'' \leq i'$ tel que v_x soit un sous terme de $\ulcorner R_{i''}\sigma \urcorner$. Mais comme $i'' \leq i < N_x$, on a une contradiction avec la minimalité de N_x, et donc pour tout $i \in \{1, .., N_x - 1\}$, v_x n'est pas un sous terme de $\ulcorner S_i\sigma \urcorner$. En résumé, on a :

$$\begin{array}{lll} & v_x \in STermes(\ulcorner R_{N_x}\sigma \urcorner) & \\ \text{mais} & v_x \notin STermes(E_0) & \text{avec } E_0 = \ulcorner S_0, S_1\sigma, .., S_{N_x-1}\sigma \urcorner \end{array}$$

puisque v_x n'est pas non plus un sous terme de S_0 (car $STermes(S_0) \subseteq \mathcal{SP}$). Le but à présent est de montrer que $v_x \in forge_{Oracle}(E_0)$. Pour cela, posons :

$$Deriv_{\ulcorner R_{N_x}\sigma \urcorner}(E_0) \quad = \quad E_0 \rightarrow_{L_1} E_0, t_1 \rightarrow_{L_2} ... \rightarrow_{L_n} E, t_1, .., t_n$$

Comme v_x est sous terme de t_n mais pas de E_0, il existe un plus petit $i \in \{1, .., n\}$ tel que v_x soit un sous terme de t_i. Si v_x est un sous terme propre de t_i, alors grâce à la définition de règles d'intrus normalisé (de composition on de décomposition), v_x serait un sous terme du membre gauche de L_i, et donc sous terme de $t_1, .., t_{i-1}$ ce qui est impossible par minimalité de i. En conséquence, $v_x = t_i$ et donc :

$$v_x \in forge_{Oracle}(\ulcorner S_0, S_1\sigma, ..., S_{N_x-1}\sigma \urcorner)$$

On veut à présent remplacer v_x par un message plus petit. Posons $\delta = [v_x \leftarrow \epsilon(v_x)]$ et $\sigma' = \sigma\delta$. Comme (π, σ) est une attaque, on a

$$\text{Pour tout } j \in \{1, .., \#\pi\}, \ulcorner R_j\sigma \urcorner \in forge_{Oracle}(\ulcorner S_0\sigma \urcorner, .., \ulcorner S_{j-1}\sigma \urcorner)$$

et l'on veut montrer les mêmes propriétés pour σ'. On distingue deux cas :

– Pour $j < N_x$, on sait par minimalité de N_x que v_x n'est ni un sous terme de $\ulcorner R_j\sigma \urcorner$ ni un sous terme de $\ulcorner S_0\sigma, .., S_{j-1}\sigma \urcorner$. Ces messages sont donc invariants par application de δ, ce qui donne $\ulcorner\ulcorner R_j\sigma \urcorner\delta \urcorner \in forge_{Oracle}(\ulcorner\ulcorner S_0\sigma \urcorner\delta, .., \ulcorner S_{j-1}\sigma \urcorner\delta \urcorner)$.

– Pour $j \geq N_x$, on a $\ulcorner R_j\sigma \urcorner \in forge_{Oracle}(E_0, F)$ avec $F = \ulcorner S_{N_x+1}\sigma, .., S_j\sigma \urcorner$, et $v_x \in forge_{Oracle}(E_0)$ avec v_x non sous terme de E_0 et v_x non atomique (puisque $Atomes \subseteq \mathcal{SP}$). On peut donc utiliser le lemme 4.2.2.2 avec E_0, F, t et v_x, ce qui donne $\ulcorner\ulcorner R_j\sigma \urcorner\delta \urcorner \in forge_{Oracle}(\ulcorner\ulcorner S_0\sigma \urcorner\delta, .., \ulcorner S_{j-1}\sigma \urcorner\delta \urcorner)$.

En résumé, on a montré que :

$$\text{Pour tout } j \in \{1, .., \#\pi\},\ \ulcorner\ulcorner R_j\sigma \urcorner\delta \urcorner \in forge_{Oracle}(\ulcorner\ulcorner S_0\sigma \urcorner\delta, .., \ulcorner S_{j-1}\sigma \urcorner\delta \urcorner)$$

Il ne nous reste alors plus qu'à utiliser le lemme 4.2.2.3 sur $S_0, .., S_{j-1}$ et sur $\{R_j\}$ pour tout j, ce qui donne :

$$\text{Pour tout } j \in \{1, .., \#\pi\},\ \ulcorner R_j\sigma' \urcorner \in forge_{Oracle}(\ulcorner S_0\sigma', .., S_{j-1}\sigma' \urcorner)$$

En effet, aucun sous terme de S_j ou R_j n'est un pré terme de v_x. En conséquence, $(\pi, \ulcorner\sigma' \urcorner)$ est une attaque. On doit normaliser σ' car le remplacement pourrait créer une substitution non normalisée. Mais grâce à la confluence de la normalisation, cela ne change pas les propriétés précédentes. De plus, $\left|\ulcorner\sigma'(x)\urcorner\right|_{dag} \leq |\sigma'(x)|_{dag} < |\sigma(x)|_{dag}$ et pour tout $y \in Var\backslash\{x\}$, $\left|\ulcorner\sigma'(y)\urcorner\right|_{dag} \leq |\sigma(y)|_{dag}$. On a donc une contradiction avec la minimalité de l'attaque (π, σ), et la proposition et donc démontrée. □

4.2.4 Borne sur les substitutions d'attaques minimales.

Nous allons maintenant utiliser le lemme clef de caractérisation des facteurs des substitutions d'attaques minimales pour borner entièrement les tailles des valeurs de ces substitutions. L'idée est d'itérer ce lemme sur les valeurs d'une substitution de manière à la décomposer en sous termes de la spécification du protocole.

Théorème 4.2.4.1 *Borne sur les substitutions d'attaques minimales.*

Soit (π, σ) une attaque minimale sur P. Alors $|\{\sigma(x) \,|\, x \in Var\}|_{dag} \leq 4.|P|_{dag}$, où $|P|_{dag} = |\mathcal{SP}|_{dag}$.

Preuve. Soit F l'ensemble des facteurs de σ, i.e. $F = \{s \,|\, \exists x \in Var,\ s \text{ facteur de } \sigma(x)\}$. Pour chaque élément s de F, on définit une nouvelle variable x_s. On pose $Var' = \{x_s \,|\, s \in F\}$ l'ensemble formé par ces nouvelles variables. On a $\#Var' = \#F$. Ces nouvelles variables (temporaires) n'étant utilisées que dans cette preuve, elles n'ont pas à apparaître dans la spécification du protocole $P : Var \cap Var' = \emptyset$. Nous allons les utiliser pour définir une substitution équivalente à σ mais à valeurs standards. On défini donc la substitution σ' telle que $\sigma'(x_s) = s$ pour toute variable $x_s \in Var'$, et $\sigma'(x) = x$ pour $x \in Var$.

Pour que ces nouvelles variables Var' soient utiles, nous devons commencer par prouver qu'elles sont en nombre raisonnable, i.e. que $\#Var' \leq |P|_{dag}$. On sait déjà, grâce au lemme 4.2.3.1, que pour tout $s \in F$, il existe $t_s \in \mathcal{SP}$ tel que $t_s \sqsubseteq_\sigma s$. On définit alors la fonction $f : Var' \to \mathcal{SP}$ telle que pour tout $x_s \in Var'$, $f(x_s) = t_s$. Cette fonction est injective, car si $t_s = t_{s'}$, alors $\ulcorner t_s\sigma \urcorner = \ulcorner t_{s'}\sigma \urcorner$, i.e. $s = s'$ et donc $x_s = x_{s'}$. En conséquence, on a :

$$\#Var' \leq |\mathcal{SP}|_{dag} = |P|_{dag}$$

La substitution σ' seule ne permet pas de remplacer complètement σ. Nous avons besoin d'une autre substitution, σ'', pour faire le lien entre les variables à valeurs non standards et leurs facteurs. On pose donc σ'' la substitution telle que :

$$\begin{array}{rl} \sigma''(x) = x & \text{si } \sigma(x) \text{ est standard, et} \\ \sigma''(x) = x_{s_1} \oplus ... \oplus x_{s_n} & \text{si } \sigma(x) = s_1 \oplus ... \oplus s_n \text{ avec } s_i \in F \text{ pour tout } i \end{array}$$

Par construction, on a $\sigma(x) = \sigma'(\sigma''(x))$ pour tout $x \in Var$, ce qui va nous permettre d'utiliser σ' à la place de σ :

$$\ulcorner\urcorner t\sigma''\urcorner\sigma'\urcorner = \ulcorner t\sigma\urcorner \qquad \text{pour tout terme } t$$

Ainsi, $\ulcorner t\sigma''\urcorner$ est un pré terme de $\ulcorner t\sigma\urcorner$ selon σ'. Cependant, $\ulcorner t\sigma''\urcorner\sigma'$ n'est pas nécessairement normalisé, ce qui peut poser un problème pour la suite de cette preuve. Nous allons donc construite un terme $\ulcorner t\sigma''\urcorner^{\sigma'}$ à partir de $\ulcorner t\sigma''\urcorner$, tel que $\ulcorner t\sigma''\urcorner^{\sigma'}\sigma' = \ulcorner t\sigma\urcorner$. Intuitivement, il s'agit d'éliminer les facteurs des sous termes de $\ulcorner t\sigma''\urcorner$ simplifiés lors de la normalisation de $\ulcorner t\sigma''\urcorner\sigma'$. Posons :

$$(v_1 \oplus ... \oplus v_n)^{\sigma'} = \begin{cases} (v_1 \oplus ... \oplus v_n \backslash v_i \backslash v_j)^{\sigma'} & \text{si } \ulcorner v_i\sigma'\urcorner = \ulcorner v_j\sigma'\urcorner \text{ et } i \neq j \text{ minimaux.} \\ v_1^{\sigma'} \oplus ... \oplus v_n^{\sigma'} & \text{sinon.} \end{cases}$$

$$\langle a, b\rangle^{\sigma'} = \left\langle a^{\sigma'}, b^{\sigma'} \right\rangle \qquad \text{idem pour } \{a\}_b^{p \text{ ou } s}.$$

$$a^{\sigma'} = a \qquad \text{si } a \in Var \cup Var' \cup Atomes.$$

avec $v_1 \oplus ... \oplus v_n \backslash v_i \backslash v_j$ le xor de v_1 à v_n privé de v_i et de v_j. On constate immédiatement que $\ulcorner\urcorner t\sigma''\urcorner^{\sigma'}\sigma'\urcorner = \ulcorner t\sigma\urcorner$ pour tout t, puisque dans la transformation ci-dessus on a $\ulcorner (v_1 \oplus ... \oplus v_n)^{\sigma'}\sigma'\urcorner = \ulcorner (v_1 \oplus ... \oplus v_n)\sigma'\urcorner$. En outre, pour tout $u_1 \oplus ... \oplus u_k$ sous terme de $\ulcorner t\sigma''\urcorner^{\sigma'}$, on a $u_i\sigma'$ standard (car $\ulcorner t\sigma''\urcorner$ est normalisé et $\sigma'(x_s)$ est standard, pour tout $s \in F$), d'où $\ulcorner u_i\sigma'\urcorner \neq 0$, et $\ulcorner u_i\sigma'\urcorner \neq \ulcorner u_j\sigma'\urcorner$ pour tous $i \neq j$, par définition de $\ulcorner t\sigma''\urcorner^{\sigma'}$. En conséquence, $\ulcorner t\sigma''\urcorner^{\sigma'}\sigma'$ est normalisé (c.f. Table 2.4), et donc :

$$\ulcorner t\sigma''\urcorner^{\sigma'}\sigma' = \ulcorner t\sigma\urcorner \qquad \text{pour tout terme } t.$$

Nous allons utiliser ceci pour borner $|S|_{dag}$, avec $S = F \cup \{\, \sigma(x) \mid x \in Var\}$. Pour cela, étant donné un ensemble Z de messages normalisés, on pose :

$$\begin{array}{ll} V_Z = & \{\, x \in Var \mid \sigma(x) \text{ non standard et } \sigma(x) \notin Z \,\}, \\ P_Z = & \{\ulcorner t\sigma''\urcorner^{\sigma'} \mid t \in \mathcal{SP} \text{ et } \ulcorner t\sigma\urcorner \notin Z\}. \end{array}$$

Intuitivement, V_Z est l'ensemble des variables (à valeur) non standard et (à valeur) hors de Z, et P_Z est l'ensemble des sous termes du protocole dont on a éliminé toutes les variables de Var et éliminé tous les facteurs de σ, i.e. $\ulcorner \mathcal{SP}\sigma''\urcorner^{\sigma'}$, moins tous les pré termes et variables à valeur dans Z, i.e. $\{t' \mid t'\sigma' \in Z\}$. L'idée derrière ceci est de construire une suite d'ensembles Z_i, partant de S, tels que $|Z_i \cup V_{Z_i} \cup P_{Z_i}|_{dag}$ croit avec i, mais dont la taille du dernier élément est connue. En particulier, on peut commencer par remarquer que si $Z \subseteq Z'$, alors $V_{Z'} \subseteq V_Z$ et $P_{Z'} \subseteq P_Z$. De plus, $V_S = \emptyset$ car S contient au moins toutes les valeurs $\sigma(x)$, $x \in Var$.

Nous allons montrer que $|S \cup P_S|_{dag} \leq |V_\emptyset \cup P_\emptyset|_{dag}$. Pour cela, on construit une séquence d'ensembles

$$S = Z_1 \supset Z_2 \supset \cdots \supset Z_n = \emptyset \text{ avec } Z_{i+1} = Z_i \backslash v_i \text{ et } |v_i|_{dag} = \max_{v \in Z_i} |v|_{dag}$$

Cette séquence décroît de S vers $\emptyset$ en éliminant à chaque pas un message maximal en taille DAG dans Z_i. Il est important de remarquer que S contient $n-1$ éléments, et que pour tout i, v_i n'est jamais un facteur de $t \in Z_{i+1}$. On veut montrer que

$$\text{Pour tout } i,\ |Z_i \cup V_{Z_i} \cup P_{Z_i}|_{dag} \leq \left|Z_{i+1} \cup V_{Z_{i+1}} \cup P_{Z_{i+1}}\right|_{dag}$$

En effet, à chaque pas i, deux cas peuvent se produire :

- S'il existe $x \in Var$ tel que $v_i = \sigma(x) = r_1 \oplus ... \oplus r_k$ non standard, avec $F' = \{r_i\}_{1..k} \subseteq F$ alors :

$$\begin{array}{rcl} STermes(Z_i \cup V_{Z_i} \cup P_{Z_i}) & = & STermes(Z_i \backslash \{v_i\} \cup V_{Z_i} \cup P_{Z_i}) \cup \{\sigma(x)\} \cup STermes(F') \\ \text{d'où } |Z_i \cup V_{Z_i} \cup P_{Z_i}|_{dag} & \leq & |Z_i \backslash \{v_i\} \cup V_{Z_i} \cup P_{Z_i} \cup F'|_{dag} + 1 \\ \text{d'où } |Z_i \cup V_{Z_i} \cup P_{Z_i}|_{dag} & \leq & |Z_i \backslash \{v_i\} \cup V_{Z_i} \cup \{x\} \cup P_{Z_i} \cup F'|_{dag} \\ \text{d'où } |Z_i \cup V_{Z_i} \cup P_{Z_i}|_{dag} & \leq & \left|Z_{i+1} \cup V_{Z_{i+1}} \cup P_{Z_{i+1}}\right|_{dag} \end{array}$$

  car $x \notin Z_i \cup V_{Z_i} \cup P_{Z_i} \cup F$, $x \in V_{Z_{i+1}}$, et $F' \subseteq Z_i \backslash \{v_i\}$.

- Si $v_i \in F$, alors il existe $t \in \mathcal{SP}$ tel que $t \sqsubseteq_\sigma v_i$. Posons $t' = \ulcorner t\sigma'^{\urcorner\sigma'}$. Par construction, on a $t'\sigma' = \ulcorner t\sigma \urcorner = v_i$. Alors :

$$\begin{array}{rcl} |Z_i \cup V_{Z_i} \cup P_{Z_i}|_{dag} & \leq & \left|Z_{i+1} \cup \{t'\} \cup V_{Z_{i+1}} \cup P_{Z_i}\right|_{dag} \\ & \leq & \left|Z_{i+1} \cup V_{Z_{i+1}} \cup P_{Z_{i+1}}\right|_{dag} \end{array}$$

  car $P_{Z_{i+1}} = P_{Z_i} \cup \{t'\}$ et pour tout $y \in Var(t')$, $\sigma'(y) \in Z_i \backslash v_i = Z_{i+1}$.

Dans les deux cas, la propriété est vérifiée. On obtient donc $|S \cup V_S \cup P_S|_{dag} \leq |\emptyset \cup V_\emptyset \cup P_\emptyset|_{dag}$, i.e. $|S \cup P_S|_{dag} \leq |V_\emptyset \cup P_\emptyset|_{dag}$. De plus, comme :

$$|P_\emptyset| \ = \left|\ulcorner \mathcal{SP}\sigma''^{\urcorner}\right|_{dag} \leq |\mathcal{SP}\sigma''|_{dag} \leq |\mathcal{SP}|_{dag} + |Var'|_{dag}, \quad \text{et}$$

$$\begin{array}{rcl} \text{on a } |\{\sigma(x) \,|\, x \in Var\}|_{dag} & \leq & |S| \leq |S \cup P_S| \leq |V_\emptyset \cup P_\emptyset| \\ \text{et } |\{\sigma(x) \,|\, x \in Var\}|_{dag} & \leq & |Var|_{dag} + |\mathcal{SP}|_{dag} + |Var'|_{dag} \leq 3.|P|_{dag} \end{array}$$

Et le théorème est donc démontré. □

Pour faciliter l'utilisation de ce théorème, on en donne un corollaire immédiat :

Corollaire 4.2.4.2 *Borne sur les messages transmis.*
Pour toute attaque minimale (π, σ), et pour tout $i \in \{1, .., \#\pi\}$, on a :

$$|R_i\sigma, S_0\sigma, .., S_{i-1}\sigma|_{dag} \leq 4.|P|_{dag} \text{ et } |Secret, S_0\sigma, .., S_{\#\pi}\sigma|_{dag} \leq 4.|P|_{dag}$$

4.3 Algorithme NP pour le problème de l'insécurité

Nous avons maintenant à notre disposition deux types de bornes : d'une part, tous les messages transmis lors d'une attaque minimale sont bornés linéairement par la taille de la spécification de protocole, et d'autre part il existe une dérivation servant à créer un message $R_i\sigma$ à partir de $S_0\sigma, .., S_{i-1}\sigma$ linéairement bornée par la taille de $R_i\sigma, S_0\sigma, .., S_{i-1}\sigma$ et donc par la taille de la spécification de protocole. De plus, tout protocole admettant une attaque en admet une minimale. Il nous suffit donc de décrire un algorithme NP (correcte, i.e. pas d'approximation du résultat) permettant de trouver une attaque minimale dés qu'il en existe au moins une. Ainsi, on peut certifier que le protocole est sûr ssi l'on n'a pas trouvé d'attaque. Le détail de l'algorithme est donné dans la Figure 4.1. Pour simplifier les notations, on suppose que :

1. Choisir un ordre d'exécution π sur P.
2. Choisir une substitution close σ t.q. pour tout $x \in Var$, $|\sigma(x)|_{dag} \leq 3.|P|_{dag}$
5. Pour chaque $i \in \{1, .., \#\pi + 1\}$, tester si $\ulcorner R_i\sigma \urcorner \in forge_{Oracle}(\ulcorner S_0\sigma, .., S_{i-1}\sigma \urcorner)$.
6. Si toutes ces vérifications sont réussies, alors répondre VRAI.

FIG. 4.1: Algorithme NP pour l'insécurité de protocoles avec oracle (xor).

$$\boxed{R_{\#\pi+1} = Secret}$$

Cet algorithme commence par choisir une attaque potentielle (π, σ) de taille linéairement bornée par n (points 1 et 2). Si le protocole admet effectivement une attaque minimale, alors il est possible de la choisir ici (Théorème 4.2.4.1). Ensuite, il faut vérifier que l'attaque potentielle choisie est bien une attaque. Pour cela, on doit vérifier que pour tout $i \in \{1, .., \#\pi\}$, $\ulcorner R_i\sigma \urcorner \in forge_{Oracle}(\ulcorner S_0\sigma \urcorner, .., \ulcorner S_{i-1}\sigma \urcorner)$ et $Secret \in forge_{Oracle}(\ulcorner S_0\sigma \urcorner, .., \ulcorner S_{\#\pi}\sigma \urcorner)$. Pour cela, il faut que le problème d'appartenance à

$$Derive(Oracle) \ = \ \{ \ (t, E) \mid t \in forge_{Oracle}(E) \ \}$$

avec E un ensemble de messages normalisés et t un message normalisé soit décidable en temps polynomial en $|E, t|_{dag}$. En effet, on sait déjà que $|R_i\sigma, S_0\sigma, .., S_{i-1}\sigma|_{dag}$ est bornée polynomialement par $|P|_{dag}$. Nous allons le montrer pour les intrus xor et préfixe. Cette procédure est donc complète, et comme elle effectue toutes les vérifications nécessaires pour prouver qu'une attaque potentielle (π, σ) est bien une attaque (pas forcément minimale, d'ailleurs), elle est aussi correcte.

Examinons la complexité de cette procédure. Si $n = |P|_{dag}$, alors :

- L'ordre d'exécution π peut être choisi en temps polynomial, car ce n'est qu'une fonction injective partielle de $\mathcal{J} \subset \mathcal{I}$ dans $\{1, .., \#\mathcal{J}\}$ avec $\#\mathcal{I} \leq n$.
- Une représentation DAG de la substitution σ peut être choisie en temps polynomial en n, puisque pour tout $x \in Var$, $|\sigma(x)|_{dag} \leq 3.n$.
- Pour toutes règles d'oracle avec lesquelles on veut utiliser cet algorithme, on doit prouver que le problème *Derive* correspondant est décidable en temps polynomial.

Il ne nous reste donc plus qu'à donner la complexité du problème $DERIVE$ pour les intrus xor et préfixe :

Proposition 4.3.0.3 *Le problème Derive est décidable en temps polynomial.*

Soit Oracle un ensemble de règles d'oracle telles que décider si $F \rightarrow_{Oracle} F, u$ est polynomial en $|F, u|_{dag}$, pour tous F et u normalisés. Alors on peut décider si $\ulcorner t \urcorner \in forge_{Oracle}(\ulcorner E \urcorner)$ en temps polynomial en $|E, t|_{dag}$, pour tout ensemble de messages E et tout message t.

PREUVE. Tout d'abord, on peut supposer que E et t sont normalisés. En effet, la normalisation de E et t prend un temps au plus polynomial en $|E, t|_{dag}$, et on a $\left|\ulcorner E, t \urcorner\right|_{dag} \leq |E, t|_{dag}$. A présent,

nous pouvons effectuer exactement la même preuve que pour l'intrus DY à la proposition 3.5.0.3. En effet, grâce à l'existence de dérivations bien formées nous savons que pour tout $u \in \mathcal{F}$ avec $\mathcal{F} = forge_{Oracle}(E) \cap STermes(E,t)$, il existe une dérivation partant de E, de but t, dont tous les messages intermédiaires sont dans $\mathcal{F}$, et de longueur bornée par $|E,t|_{dag}$. De plus, nous savons par hypothèse que l'on décider si $F \rightarrow_{Oracle} F,u$, avec $F \subseteq \mathcal{F}$ et $u \in STermes(E,t)$, en temps polynomial en $|E,t|_{dag}$. On a donc bien la proposition annoncée. □

On en déduit immédiatement :

Corollaire 4.3.0.4 *Pour les intrus xor et préfixe, le problème Derive est polynomial.*

Ainsi, on obtient un algorithme NP pour décider de l'insécurité de protocoles cryptographiques face à un intrus disposant de règles d'oracle dans notre modèle de protocoles, d'où le théorème suivant :

Théorème 4.3.0.5 *L'insécurité modulo des règles d'oracle est NP.*

Pour tout intrus disposant de règles d'oracle $\mathcal{L}$ (selon la définition 4.1.1.1), basé sur les opérateurs standards et le xor, et dont on peut décider en temps polynomial en $|E,t|_{dag}$ si $E \rightarrow t \in \mathcal{L}$, le problème de l'insécurité de protocoles cryptographiques (basé sur ces mêmes opérateurs) est NP.

Dans le cas des intrus xor et préfixe, on a déjà prouvé que toutes les conditions de ce théorème sont remplies, d'où :

Corollaire 4.3.0.6 *L'insécurité modulo xor ou préfixe est NP-complète.*

Le problème de l'insécurité de protocoles cryptographiques (basé sur les opérateurs standards et xor) face à l'intrus xor est NP-complet.

Le problème de l'insécurité de protocoles cryptographiques (basé sur les opérateurs standards et xor) face à l'intrus préfixe est NP-complet.

En effet, les codages de problèmes 3-SAT du chapitre 3 sont toujours valables. On prendra cependant soin, dans le cas de l'intrus préfixe, de remplacer l'opérateur d'encryption symétrique par l'opérateur d'encryption asymétrique (en définissant des couples de clefs) dans le premier codage pour éviter toute interaction avec les règles de cet intrus.

4.4 Conclusion

Nous avons présenté dans ce chapitre un résultat de complexité pour l'insécurité de protocoles cryptographiques face aux intrus xor et préfixe. Ce résultat est basé sur une notion relativement générale d'oracle, i.e. un genre particulier de règles d'intrus munies de propriétés nécessaires à nos preuves. Cette notion d'oracle nous a permis de séparer d'une part, les propriétés dépendant de la nature des règles d'intrus (preuves que les intrus xor et préfixes définissent des règles d'oracle), et d'autre part les propriétés dépendant plus des opérateurs utilisés (opérateurs standards et xor) que de la nature des règles d'intrus. De cette manière, d'autres intrus non décrits ici, mais n'utilisant que ces opérateurs standards et xor, peuvent très bien définir également des règles d'oracle.

5

Insécurité avec exponentielle (Diffie-Hellman)

Sommaire

Nous avons vu aux chapitres précédents qu'il est possible de décider de la sécurité de protocoles cryptographiques à nombre de sessions borné, avec ou sans l'opérateur xor, avec une complexité raisonnable. Cependant, différents protocoles comme A-GDH.2 utilisent d'autres opérateurs algébriques comme l'exponentiation, et admettent des attaques si l'on en prend en compte les propriétés algébriques (c.f. [78]). Dans [66, 55], des méthodes d'unification sont proposées pour capturer des propriétés de Diffie-Hellman des opérateurs. Même si le problème de l'unification est important, cela ne permet pas de résoudre le problème de l'insécurité qui est plus

complexe. Par ailleurs, il existe plusieurs tentatives infructueuses pour décider du problème de la sécurité des protocoles cryptographiques avec exponentiation. Par exemple, dans [69] J.Millen et V.Shmatikov proposent une réduction de ce problème à une résolution de systèmes d'équations quadratiques. Malheureusement, résoudre ces systèmes est en général indécidable. Un problème similaire a également été étudié par M.Boreale et M.G. Buscemi dans [17]. Ils proposent une procédure correcte et complète par rapport à une sémantique opérationnelle, mais il n'est pas clair qu'elle prenne en compte toutes les propriétés de Diffie-Hellman de l'exponentiation. De plus, la méthode nécessite une borne à priori sur le nombre de facteurs des produits (dans les exposants).

Dans ce chapitre, nous allons étudier la complexité du problème de l'insécurité de protocole face à l'intrus DH, une extension de l'intrus de Dolev-Yao avec des opérateurs de produit et d'exponentielle (c.f. chapitre 2, section 2.4 pour les définitions), dans notre modèle à nombre de sessions borné. Cet opérateur exponentielle a pour but de modéliser l'exponentiation modulo un nombre premier public, connu de l'intrus. Pour cette raison, tous les participants peuvent inverser un exposant connu. Typiquement, connaissant $m = g^{a \cdot b} \bmod p$ et b, avec g générant le groupe multiplicatif engendré par p, l'intrus peut facilement calculer b^{-1} i.e. l'inverse de b modulo $p-1$, et donc calculer $g^a = m^{(b^{-1})}$. La structure générale de la première partie de ce chapitre va suivre la structure de preuve déjà présentée aux deux chapitres précédents (en particulier, nous réutiliserons la notion de règles oracle en la modifiant un peu). Néanmoins, les propriétés particulières des opérateurs exponentielle et produit vont nécessiter des preuves différentes, utilisant des lemmes nettement plus techniques. De plus, borner les tailles DAG des substitutions et des dérivations ne suffit plus ici. En effet, un terme construit avec l'opérateur produit utilise des coefficients entiers, non pris en compte dans la taille DAG. En conséquence, nous aurons une section supplémentaire assez importante pour borner ces coefficients entiers.

Le plan de ce chapitre sera donc le suivant : Nous commencerons par fixer les conventions et la définition d'oracle utilisées dans ce chapitre (section 5.2), puis nous montrerons que l'intrus DH définit bien un ensemble de règles d'oracle (section 5.4). Nous pourrons alors borner les substitutions d'attaques minimales en trois temps : Nous montrerons tout d'abord que les facteurs d'attaques minimales admettent un pré terme dans la spécification du protocole (section 5.5, propriété identique au cas xor mais nécessitant des preuves différentes et plus complexes). Puis nous montrerons que les tailles DAG des substitutions d'attaques minimales sont bornées polynomialement par la taille de la spécification du protocole considéré. Enfin, nous montrerons à la section 5.7 que l'intrus DH permet de ne considérer que des attaques à coefficients entiers de taille bornée (polynomialement en la taille de la spécification du protocole).

Tout ceci nous permettra enfin de justifier à la section 5.8 un algorithme NP de décision du problème de l'insécurité de protocole avec les opérateurs produit et exponentielle, et face à l'intrus DH ou à tout ensemble de règles d'oracle, modulo quelques propriétés supplémentaires.

5.1 Exemple : protocole A-GDH.2

Pour montrer l'intérêt de l'opérateur exponentiel, et surtout les propriétés qu'on lui a donné, nous allons présenter un protocole de groupe relativement complexe basé sur l'opérateur exponentielle (c.f. [78]). Le but de ce protocole est de permettre à un ensemble de principaux de construire une clef (de session, i.e. temporaire) connue uniquement par les membres du groupe (à condition bien sûr que tous les membres du groupe soient honnêtes). L'intrus sera tantôt un membre du groupe tentant d'utiliser sa position pour attaquer un autre groupe, tantôt un simple élément extérieur tentant de modifier à son avantage la clef choisie par un groupe donné.

Soient A_1, A_2, A_3, $..A_n$ un ensemble de principaux désirant créer un clef de session secrète partagée (avec le protocole A-GDH.2). L'un de ces principaux va avoir un rôle particulier dans la création de la clef partagée, nous l'appellerons le "maître". Il aura pour rôle de centraliser l'ensemble des messages émis par les autres membres du groupe, puis de distribuer les messages permettant à chacun de construire la clef secrète partagée. Cette dernière phase du protocole va nécessiter l'existence préalable d'une clef secrète partagée entre le maître et chaque membre du groupe. Nous les nommerons K_1, K_2, $..K_{n-1}$, avec K_i la clef entre secrète partagée par A_i et A_n. En outre, chaque principal du groupe doit choisir préalablement un nombre aléatoire r_i, $i \in \{1, .., n\}$, qui sera utilisé dans la construction de la clef partagée par le groupe. On suppose que ces choix sont réalisés avant le début du protocole. Enfin, on suppose que le groupe s'est déjà mis d'accord sur une racine commune g pour toutes les exponentielles considérées dans le protocole. En effet, cette racine n'a aucune influence particulière dans l'exécution du protocole, et peut être fixée une fois pour toute (commune à plusieurs sessions). Cependant, notre intrus peut utiliser n'importe quelle autre racine, ce qui pourrait être utile si l'on voulait étendre ce protocole. En résumé, les connaissances initiales des principaux sont :

- Pour tout $i \in \{1, .., n-1\}$, A_i connaît initialement g, K_i, et r_i.
- Initialement, A_n connaît g, r_n, et $\{ K_i \mid i \in \{1, .., n-1\} \}$.

Le but du protocole est de permettre aux seuls principaux A_1, $..A_n$ de construire la clef secrète partagée $Exp(g, r_1^1 \cdot .. \cdot r_n^1)$. Pour cela, le protocole est divisé en deux phases :

Phase de collecte (tour de table des participants) :

Pour tout $i \in \{1, .., n-1\}$, $A_i \rightarrow A_{i+1}$: $Exp(g, M_i \bullet r_1^{-1})$, .., $Exp(g, M_i \bullet r_i^{-1})$, $Exp(g, M_i)$ avec $M_i = r_1^1 \cdot .. \cdot r_i^1$

Phase de broadcast (distribution des résultats) :

Pour tout $i \in \{1, .., n-1\}$, $A_n \rightarrow A_i$: $Exp(g, r_1^1 \cdot .. \cdot r_n^n \cdot r_i^{-1} \cdot K_i)$

A la fin de l'exécution du protocole, chaque principal A_i peut alors calculer la clef secrète :

$$\begin{array}{rcl} \forall i \in \{1, .., n-1\}, \quad Exp(g, r_1^1 \cdot .. \cdot r_n^1) & = & Exp(Exp(g, r_1^1 \cdot .. \cdot r_n^n \cdot r_i^{-1} \cdot K_i), r_i^1 \cdot K_i^{-1}) \\ Exp(g, r_1^1 \cdot .. \cdot r_n^1) & = & Exp(Exp(g, r_1^1 \cdot .. \cdot r_{n-1}^1), r_n^1) \end{array}$$

Pour savoir si chaque principal a effectivement calculé une clef secrète, nous complétons ce protocole en obligeant les principaux participants à montrer qu'ils ont confiance dans la clef calculée : ils envoient tous l'atome *Secret* encrypté par leur nouvelle clef à l'intrus. Cela n'a bien sûr de sens que si l'intrus ne fait pas partie du groupe de principaux.

Phase de test (ajoutée pour vérifier le protocole) :

Pour tout $i \in \{1, .., n\}$, $A_i \rightarrow I$: $\{Secret\}^s_{Exp(g, r_1^1 \cdot .. \cdot r_n^1)}$

Ainsi, l'intrus est capable de calculer *Secret* ssi il connaît la clef partagée par au moins un des principaux du groupe. On remarque que l'absence d'attaque permettant à l'intrus de calculer *Secret* ne serait pas une preuve que les principaux partagent nécessairement la même clef à la fin de toute exécution. Mais il suffit d'examiner le protocole pour se rendre compte que cette second propriété ne peut pas être garantie : il suffirait à l'intrus d'envoyer n'importe quels nombres aléatoires aux principaux : ceux-ci ne pouvant pas vérifier s'il s'agit ou non d'une exponentielle, ils calculeraient ainsi des clefs inconnues de l'intrus mais différentes deux à deux.

Nous nous concentrons donc sur la propriété principale, i.e. le secret de la clef construite par chaque principal. Pour cela, nous allons donner une spécification d'une instance de ce protocole dans le modèle par rôles, à savoir quatre principaux (honnêtes) $Luke$, $Leia$, $Yahn$, et $Yoda$ dans le rôle du maître, et deux sessions en séquence. Dans la première session, l'intrus (alias $CasqueNoir$) prend officiellement la place de $Yahn$ et est donc est placé en troisième position. On ne demandera donc pas ici que la clef de session créée reste secrète. En revanche, dans la seconde session une clef secrète est créée entre les quatre principaux honnêtes $Luke$, $Leia$, $Yahn$, et $Yoda$ uniquement. Pour celle-ci nous ajoutons la phase de test. De plus, nous utiliserons naturellement deux jeux de nombres aléatoires r_1,..,r_4 et r'_1,..,r'_4 (un pour chaque session), et deux clefs K_3 (pour l'intrus, session 1) et K'_3 (pour $Yahn$, session 2). En revanche, les clefs initialement partagées K_1 et K_2 avec $Yoda$ sont utilisées pour les deux sessions (sinon créer une clef de session n'a plus de sens, autant utiliser les K_i directement). Pour alléger un peu la présentation, nous allons donner la forme générale des pas de ces principaux, selon la phase (collecte, broadcast ou vérification). Nous instencierons ensuite ces pas génériques avec $n = 4$ pour créer le protocole que l'on souhaite vérifier.

Collecte n°1 : Le protocole commence par la phase de collecte de la première session.

Le premier principal du groupe se contente de générer la base des échanges de messages suivants. Nommons C_1^1 ce pas :

$$C_1^1 \ : \ Init \ \Rightarrow \ g, \, Exp(g, \, r_1^1)$$

Tout principal dans cette phase va ensuite exécuter le pas suivant, nommé C_{i+1}^1 pour collecte numéro 1 avec le principal numéro $i+1 \in \{2, ..n-1\}$:

$$C_{i+1}^1 \ : \ x_{1..i\backslash 1}, .., x_{1..i\backslash i}, x_{1..i} \ \Rightarrow \ Exp(x_{1..i\backslash 1}, \, r_{i+1}), .., Exp(x_{1..i\backslash i}, \, r_{i+1}), \, x_{1..i}, \, Exp(x_{1..i}, \, r_{i+1})$$

Les variables $x_{1..i\backslash j}$ ont pour valeurs $Exp(g, \, r_1^1 \cdot .. \cdot r_i^1 \cdot r_j^{-1})$ dans une exécution (théorique) du protocole sans intervention "frauduleuse" de l'intrus. C'est donc ce que le principal numéro $i+1$ croit connaître. Il est important de remarquer ici que même si $x_{1..i\backslash i}$ et $x_{1..i-1}$ sont sensés avoir les mêmes valeurs dans une exécution du protocole sans intervention de l'intrus, ce sont des connaissances de deux principaux différents (numéros $i+1$ et i resp.). Elles n'auront donc pas nécessairement les mêmes valeurs dans une attaque. Par ailleurs, l'intrus doit en principe contrôler l'un de ces principaux. Nous le spécifierons uniquement avec les connaissances initiales de l'intrus. Il nous reste à donner le dernier pas de protocole de cette phase, i.e. celui du maître :

$$C_n^1 \ : \ x_{1..n-1\backslash 1}, \, .., \, x_{1..n-1\backslash n-1}, \, x_{1..n-1} \ \Rightarrow \ End$$

Broadcast n°1 : Le principal maître ayant reçu le message envoyé par l'avant dernier principal, il crée $n-1$ messages adaptés à chaque principal, de 1 à $n-1$, pour permettre à chacun de créer la clef partagée de la session n°1. Comme on ne spécifie pas ici à quel principal un message est envoyé, le principal maître se contente d'exécuter un seul pas rassemblant tous les messages pour tous les autres principaux :

$$B^1 \ : \ Init \ \Rightarrow \ Exp(x_{1..n-1\backslash 1}, \, r_n^1 \cdot K_1), \, .., \, Exp(x_{1..n-1\backslash n-1}, \, r_n^1 \cdot K_{n-1})$$

On utilise K_I au lieu de K_{n-1} car l'intrus prend la place du principal $n-1$. Il ne connaît bien sûr pas sa clef secrète K_{n-1}.

Collecte n°2 : La seconde phase de collecte (i.e. la collecte pour la seconde session) est identique à la collecte pour première session. On doit cependant renommer toutes les variables et tous les

atomes r_i (pour avoir deux session indépendantes). Pour tout $i \in \{1, .., n\}$, on pose donc C_i^2 le pas C_i^1 où l'on remplace toutes les variables $x_{1..i\setminus j}$ par $y_{1..i\setminus j}$, toutes les variables $x_{1..i}$ par $y_{1..i}$, et tous les atomes r_k par r'_k.

Broadcast n°2 : De la même manière, la seconde phase de broadcast, nommée B^2, est identique à B^1 excepté que l'on remplace toutes les variables $x_{1..i\setminus j}$ par $y_{1..i\setminus j}$, toutes les variables $x_{1..i}$ par $y_{1..i}$, tous les atomes r_k par r'_k, et naturellement K_{n-1} par K'_{n-1} puisque l'intrus ne participe pas à cette session.

Vérification : Comme l'intrus ne participe (normalement) par à la seconde sessions, nous ajoutons maintenant la phase de vérification. Pour cela, chaque principal $i \in \{1, .., n-1\}$ va recevoir le message z_i créé pour lui par le principal maître, et envoyer *Secret* encrypté par la clef ainsi générée. Le principal maître enverra quant à lui directement *Secret* encrypté. On obtient un pas V_i pour chaque principal i :

$$\begin{array}{lrcl} \text{Pour } i \in \{1, .., n-2\}, & V_i : \quad z_i & \Rightarrow & \{Secret\}^s_{Exp(z_i, r'_i \cdot K_i^{-1})} \\ & V_{n-1} : \quad z_{n-1} & \Rightarrow & \{Secret\}^s_{Exp(z_{n-1}, r'_{n-1} \cdot K'^{-1}_{n-1})} \\ & V_n : \quad Init & \Rightarrow & \{Secret\}^s_{Exp(y_{1..n-1}, r'_n)} \end{array}$$

On peut (enfin) construire la spécification de protocole qui nous intéresse. On pose $n = 4$, puisque l'on utilise les quatre principaux *Luke*, *Leia*, *Yahn* et *Yoda*. Les pas de principaux sont donc $\mathcal{I} = \{ C_i^1, C_i^2, V_i \mid i \in \{1, .., n\} \} \cup \{ B^1, B^2 \}$, avec $C_1^1 <_{\mathcal{I}} C_1^2 <_{\mathcal{I}} V_1$ pour *Luke*, $C_2^1 <_{\mathcal{I}} C_2^2 <_{\mathcal{I}} V_2$ pour *Leia*, $C_3^2 <_{\mathcal{I}} V_3$ pour *Yahn* (l'intrus prend sa place à la première session), et $C_4^1 <_{\mathcal{I}} B^1 <_{\mathcal{I}} C_4^2 <_{\mathcal{I}} B^2 <_{\mathcal{I}} V_4$ pour maître *Yoda*. De plus, les connaissances initiales de l'intrus sont $S_0 = \{ Init, r_3, K_3, g\}$. Ce protocole admet l'attaque suivante :

$$\begin{array}{ll} \pi : & C_2^1 \, \pi \, C_4^1 \, \pi \, B^1 \, \pi \, C_2^2 \, \pi \, V_2 \\ \sigma : & \sigma(y_{1..1}) = Exp(g, r_4) \text{ et } \sigma(z_2) = Exp(g, r_4 \cdot K_2) \\ & \sigma(x) = g \text{ pour toute autre variable } x \notin \{y_{1..1}, z_2\} \end{array}$$

En effet, (π, σ) est une attaque car l'intrus est capable de construire les messages réclamés par les principaux et obtient *Secret* à la fin :

- Pour le pas C_2^1, l'intrus connaît S_0 et doit construire g, g, ce qui est évidement possible. En retour, ses connaissances deviennent $S_1 = S_0 \cup \{Exp(g, r_2)\}$.
- Pour le pas C_4^1, l'intrus connaît S_1 et doit construire g, g, g, g, ce qui est évidement possible. En retour, ses connaissances deviennent $S_2 = S_1 \cup \{End\}$.
- Pour le pas B^1, l'intrus connaît S_2 et doit construire $Init$, ce qui est évidement possible. En retour, ses connaissances deviennent $S_3 = S_2 \cup \{Exp(g, r_4 \cdot K_1), Exp(g, r_4 \cdot K_2), Exp(g, r_4 \cdot K_I)\}$.
- Pour le pas C_2^2, l'intrus connaît S_3 et doit construire $g, Exp(g, r_4)$, ce qui est possible puisqu'il a $Exp(g, r_4 \cdot K_I)$ et K_I. En retour, ses connaissances deviennent $S_4 = S_3 \cup \{Exp(g, r'_2), Exp(g, r_4 \cdot r'_2)\}$.
- Pour le pas V_2, l'intrus connaît S_3 et doit construire $Exp(g, r_4 \cdot K_2)$, ce qui est évidement possible puisqu'il a reçu ce message au pas B^1. En retour, ses connaissances deviennent $S_5 = S_4 \cup \{\{Secret\}^s_{Exp(g, r_4 \cdot r'_2)}\}$ car $Exp(Exp(g, r_4 \cdot K_2), r'_2 \cdot K_2^{-1}) = Exp(g, r_4 \cdot r'_2)$.
- Enfin, l'intrus peut calculer *Secret* à partir de S_5 : comme il connaît $Exp(g, r_4 \cdot r'_2)$, il lui suffit de décomposer $\{Secret\}^s_{Exp(g, r_4 \cdot r'_2)}$.

On constate ainsi que l'intrus est capable, grâce aux propriétés des opérateurs exponentielle et produit, d'utiliser une session officielle du protocole pour attaquer une seconde session dont il ne fait normalement pas partie. Les lecteurs les plus attentifs auront cependant remarqués un petit défaut dans notre traduction de ce protocole du modèle Alice-Bob vers le modèle par rôles : pour simplifier la spécification, le second principal du groupe reçoit (entre autres) une variable $x_{1..1\backslash 1}$. Mais la valeur théorique de cette variable est g. Le principal n°2 devrait donc normalement attendre g à la place de $x_{1..1\backslash 1}$, puisqu'il est capable de reconnaître g. Cependant, cela ne gênerais pas l'attaque puisque l'on a choisis $\sigma(x_{1..1\backslash 1}) = \sigma(y_{1..1\backslash 1}) = g$. La seule différence serait une spécification un peu plus lourde.

5.2 Généralités et Règles d'oracle

Dans tout ce chapitre, nous n'utiliserons jamais les opérateurs xor et encryption commutative. En revanche, nous utiliserons pleinement les opérateurs d'exponentielle et de produit (et bien sûr les opérateurs standards de Dolev-Yao). En conséquence, tous les protocoles considérés ici seront toujours supposés spécifiés sans les opérateurs xor et encryption commutative, et tous les termes et messages considérés n'utiliseront jamais ces opérateurs. En conséquence, toutes les attaques recherchées dans ce chapitre n'utiliseront ni le xor ni l'encryption commutative.

Nous utiliserons dans ce chapitre une notion d'oracle très proche de celle utilisée pour les intrus xor et préfixe. La différence essentielle réside dans la suppression de la fonction ϵ. En effet, cette fonction est inutile dans le cas de l'intrus DH (on utilise les remplacements $[u \leftarrow 1]$ au lieu de $[u \leftarrow \epsilon(u)]$). Il serait possible de traiter le cas DH avec une telle fonction (pour ouvrir la porte à d'autres intrus type préfixe), mais cela ne ferait que surcharger encore plus des preuves déjà complexes. Pour de tels intrus, il parait plus raisonnable d'utiliser les résultats du chapitre précédent. Concrètement, toutes les règles d'oracle considérées dans ce chapitre devront répondre à la définition suivante, avec la convention de notation $\mathcal{L} \in D$ ssi $\exists L \in \mathcal{L}$ t.q. $L \in D$:

Définition 5.2.0.7 *Règles d'oracle.*

Soit un intrus normalisé $\mathcal{L}$ disposant des règles d'intrus normalisé $L_{DY} \cup L_{oc} \cup L_{od}$ avec L_{DY}, L_{oc}, et L_{od} disjoints, L_{oc} des règles d'intrus normalisé de composition, et L_{od} des règles d'intrus normalisé de décomposition. Alors $\mathcal{L}$ est un oracle (ou dispose de règles d'oracle) ssi :

1. *Pour tous E et t, si $t \in forge_{\mathcal{L}}(E)$ alors il existe une dérivation bien formée (construite sur $\mathcal{L}$) partant de E et de but t.*
2. *Pour tous E, a et t, si $E \rightarrow_{L_{oc}} E, t$ et $E, t \rightarrow_{L_d(t)} E, t, a$, alors il existe une dérivation D partant de E et de but a telle que $L_d(t) \notin D$.*
3. *Pour tout message non atomique u, pour tout ensemble fini de messages (standards) F t.q. $1 \in F$, et pour tout message t, si $F \rightarrow_{L_c \cup L_{oc}} F, u$ (i.e. u peut être composé en un pas) et si $F, u \rightarrow_{L_{oc} \cup L_{od}} F, u, t$ (i.e. t n'est pas créé par DY), alors $\ulcorner t[u \leftarrow 1] \urcorner \in forge_{\mathcal{L}}(\ulcorner F[u \leftarrow 1] \urcorner)$.*

La première condition, comme dans le chapitre précédent, va nous permettre de borner la longueur et la nature des dérivations dont on a besoin pour prouver la validité d'une attaque. Les conditions 2 et 3 quant à elles vont nous permettre d'effectuer des remplacements d'un terme par un autre plus petit, à la manière du chapitre 3 où on remplaçait $\sigma(x)$ par $Init$. Ces deux conditions nous permettront donc de borner les tailles DAG des substitutions de certaines attaques.

Pour toute la suite, nous supposerons fixé un protocole $P = (\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}\}, <_{\mathcal{I}}, S_0)$ dont il existe au moins une attaque. On suppose sans perte de généralité que les connaissances

initiales de l'intrus S_0 vérifient $1 \in S_0$ (et donc ne sont pas vides). On note $\mathcal{SP}$ l'ensemble des sous termes de P, i.e. $\bigcup_{\iota\in\mathcal{I}} STermes(R_\iota) \cup STermes(S_\iota)$. De plus, dans toute attaque (π, σ) sur P, on verra toujours π comme une bijection de $\mathcal{J} \subset \mathcal{I}$ sur $\{1, .., \#\mathcal{I}\}$, et en conséquence on notera $R_i \Rightarrow S_i$ le pas de protocole $R_{\pi^{-1}(i)} \Rightarrow S_{\pi^{-1}(i)}$ pour $i \in \{1, .., \#\mathcal{J}\}$, à partir du moment où une attaque (π, σ) sera définie. Le taille du protocole P étant le nombre de sous termes des messages le constituant, on a $|P|_{dag} = |\mathcal{SP}|_{dag}$ $(= \#\mathcal{SP})$. Pour que la notion de taille de protocole soit cohérente avec la taille DAG des termes qui le composent, on suppose comme au chapitre précédent que $\#\mathcal{I} \leq |P|_{dag}$. Enfin, on suppose fixé un intrus $Oracle$ normalisé défini par un ensemble *de règles d'oracle* $L_{DY} \cup L_{oc} \cup L_{od}$ (avec L_{oc} des règles de composition, L_{od} des règles de décomposition, et L_{oc} disjoint de L_{od}). Toutes les règles d'intrus considérées ici seront toujours des règles d'$Oracle$. En conséquence, toutes les dérivations seront toujours implicitement construites sur ces règles d'oracle, et toute mention de $forge(...)$ désignera en fait $forge_{Oracle}(...)$.

5.3 Appliquer un remplacement à des termes génériques

Le but de cette première section est de présenter quelques lemmes de base sur les remplacements appliqués à des termes particuliers, normalisés, ou construits par substitution. Ce sont des propriétés ne dépendant que des définitions de terme, de normalisation, et de substitution, et sont donc indépendantes des règles d'intrus ou d'oracle. Néanmoins, ces lemmes seront utilisés dans toutes les sections suivantes, par exemple pour montrer le troisième point de la définition d'oracle appliqué aux règles DH, ou pour construire une attaque (plus petite) à partir d'une attaque supposée minimale.

5.3.1 Appliquer un remplacement à un produit ou une exponentielle.

La première propriété des remplacements que nous allons considérer dans cette section porte essentiellement sur les produits, et par extension sur les exponentielles, modulo la normalisation. Ce lemme, relativement technique à démontrer, va nous permettre dans toute la suite d'appliquer facilement un remplacement sur des termes normalisés modulo quelques hypothèses simples. A titre d'exemple, ceci sera utilisé dans les preuves des lemmes 5.3.2.1 et 5.3.3.1, et du théorème 5.4.2.2.

Lemme 5.3.1.1 *Appliquer un remplacement à des produits et à des exponentielles.*

Soit u un terme normalisé, et soit M un produit à facteurs normalisés (i.e. pour tout $t \in Facteur(M)$, t est normalisé). De plus, soit s un terme standard et normalisé et soit δ le remplacement $[s \leftarrow 1]$. On a alors :

1. $\ulcorner M\delta \urcorner = \ulcorner\urcorner M \urcorner \delta \urcorner$.
2. $\ulcorner Exp(u, M)\delta \urcorner = \ulcorner\urcorner Exp(u, M) \urcorner \delta \urcorner$ *si* $s \neq \ulcorner Exp(u, M) \urcorner$ *et pour le cas où* $s = Exp(\cdot, \cdot)$, *également* $s \neq u$.

Preuve. Le point n°1 est évident, puisque δ ne peut remplacer que des termes normalisés et que deux termes sont égaux si et seulement si leurs formes normalisées sont égales. Nous allons donc nous concentrer sur le point n°2. Supposons donc que s vérifie les restrictions correspondantes, i.e. $s \neq \ulcorner Exp(u, M) \urcorner$, avec en plus $s \neq u$ si $s = Exp(\cdot, \cdot)$. Nous allons prouver ce point en deux temps, selon la nature de u.

Supposons tout d'abord que u ne soit *pas* de la forme $Exp(\cdot,\cdot)$. On a alors soit i) $\ulcorner Exp(u,\, M)\urcorner = u$, et donc $\ulcorner M\urcorner = 1$, soit ii) $\ulcorner Exp(u, M)\urcorner = Exp(u, \ulcorner M\urcorner)$ avec $\ulcorner M\urcorner \neq 1$. Nous allons examiner ces deux (sous) cas.

- Si i) est vérifié, alors naturellement $\ulcorner M\urcorner\delta = 1$. Grâce au point 1 précédent, nous savons alors que $\ulcorner M\delta\urcorner = \ulcorner\ulcorner M\urcorner\delta\urcorner (= 1)$, et en conséquence :

$$\begin{array}{lll} \ulcorner Exp(u,\, M)\delta\urcorner & = \ulcorner Exp(u\delta,\, M\delta)\urcorner & (1) \\ & = \ulcorner Exp(u\delta,\, \ulcorner M\delta\urcorner)\urcorner & \\ & = \ulcorner u\delta\urcorner & \\ & = \ulcorner\ulcorner Exp(u,\, M)\urcorner\delta\urcorner & \end{array}$$

en utilisant $Exp(u, M) \neq s$ pour (1) (sinon $\ulcorner Exp(u, M)\urcorner = s$ puisque s est normalisé).

- Si ii) est vérifié, on a :

$$\begin{array}{lll} \ulcorner\ulcorner Exp(u,\, M)\urcorner\delta\urcorner & = \ulcorner Exp(u,\, \ulcorner M\urcorner)\delta\urcorner & \\ & = \ulcorner Exp(u\delta,\, \ulcorner M\urcorner\delta)\urcorner & (1) \\ & = \ulcorner Exp(u\delta,\, \ulcorner\ulcorner M\urcorner\delta\urcorner)\urcorner & \\ & = \ulcorner Exp(u\delta,\, \ulcorner M\delta\urcorner)\urcorner & (2) \\ & = \ulcorner Exp(u\delta,\, M\delta)\urcorner & \\ & = \ulcorner Exp(u,\, M)\delta\urcorner & (3) \end{array}$$

En effet, pour (1) on utilise $Exp(u, \ulcorner M\urcorner) \neq s$ (sinon $\ulcorner Exp(u,\, M)\urcorner = \ulcorner Exp(u, \ulcorner M\urcorner)\urcorner = s$), pour (2) on utilise le point 1 précédent, et pour (3) un utilise le fait que $Exp(u, M) \neq s$. Il est important de remarquer que pour (1) et (2), le fait que $u \neq s$ si s est de la forme $Exp(\cdot,\cdot)$ n'est pas nécessaire. Ainsi, on a prouvé le lemme quand u n'est pas une exponentielle.

Supposons à présent que $u = Exp(v,\, M')$, pour n'importe quels v et M'. On a alors :

$$\begin{array}{lll} \ulcorner\ulcorner Exp(u, M)\urcorner\delta\urcorner & = \ulcorner\ulcorner Exp(v,\, M' \bullet M)\urcorner\delta\urcorner & \\ & = \ulcorner Exp(v,\, M' \bullet M)\delta\urcorner & (1) \\ & = \ulcorner Exp(v\delta,\, (M'\delta \bullet M\delta)\urcorner & (2) \\ & = \ulcorner Exp(Exp(v\delta,\, M'\delta),\, M\delta)\urcorner & \\ & = \ulcorner Exp(u\delta,\, M\delta)\urcorner & (3) \\ & = \ulcorner Exp(u,\, M)\delta\urcorner & (4) \end{array}$$

(1) est obtenu comme pour le premier cas : en effet, on utilise le fait que v n'est pas de la forme $Exp(\cdot,\cdot)$ et que $Exp(v,\, M' \bullet M) \neq s$ (car sinon $\ulcorner Exp(v,\, M' \bullet M)\urcorner = \ulcorner Exp(u,\, M)\urcorner = s$). On utilise ici la remarque selon laquelle le premier cas est vrai même sans l'hypothèse $v \neq s$. Pour (2), on utilise à nouveau le fait que $Exp(v,\, M' \bullet M) \neq s$. Pour (3), on utilise $u \neq s$ (sinon $u = s$ et s est de la forme $Exp(\cdot,\cdot)$). Enfin, pour (4) on utilise simplement le fait que $Exp(u,\, M) \neq s$. Le lemme est donc aussi démontré dans le cas où u est une exponentielle, ce qui termine la preuve. □

Remarque 1 : La définition de $\bullet$, donnée au chapitre 2 sous section 2.4.2, est la suivante :

Pour $M = v_1^{z_1} \cdot .. \cdot v_n^{z_n}$ et $M' = w_1^{z'_1} \cdot .. \cdot w_p^{z'_p}$, on a $M \bullet M' = v_1^{z_1} \cdot .. \cdot v_n^{z_n} \cdot w_1^{z'_1} \cdot .. \cdot w_p^{z'_p}$.
On préfère ne pas utiliser $\cdot$ à la place de $\bullet$ (ce serait par abus de notation), car dans toute écriture de $t_1 \cdot t_2$ on veut être certain que t_1 et t_2 sont des termes et non des produits (pour éviter de le rappeler à chaque utilisation).

Remarque 2 : Il est intéressant de constater que ce lemme devient faux si l'on omet les restrictions de s. Considérons par exemple l'exemple suivant, où $s = \ulcorner Exp(u, M)\urcorner$: posons $s = Exp(a, b)$, $u = a$, et $M = b \cdot c \cdot c^{-1}$. On a alors $s = \ulcorner Exp(u, M)\delta\urcorner \neq \urcorner Exp(u, M)\urcorner\delta\urcorner = 1$, et le lemme serait faux si on n'imposait pas $s \neq \ulcorner Exp(u, M)\urcorner$. De la même manière, l'exemple suivant montre pourquoi $s \neq u$ est nécessaire : posons $s = u = Exp(a, b)$ et $M = c$. On a alors $Exp(1, c) = \ulcorner Exp(u, M)\delta\urcorner \neq \urcorner Exp(u, M)\urcorner\delta\urcorner = Exp(a, b \cdot c)$, ce qui contredirait le lemme.

5.3.2 Appliquer un remplacement modulo une substitution.

Nous allons présenter ici une première utilisation du lemme 5.3.1.1 précédent. Ce résultat prouve que sous certaines conditions, et modulo la normalisation, un remplacement appliqué à un terme $t\sigma$, pour toute substitution close σ, est équivalent au terme $t\sigma'$ où $\sigma' = \sigma\delta$. Ceci signifie que sous certaines conditions, le remplacement δ ne s'applique en fait qu'à la substitution σ dans $t\sigma$, et permet donc la création d'une nouvelle substitution σ'. Ce sera utilisé pour créer une attaque (π, σ') à partir d'une attaque (π, σ). Concrètement, on a :

Lemme 5.3.2.1 *Appliquer un remplacement modulo une substitution (et la normalisation).*

Soit σ une substitution close et normalisée, soit E un ensemble de termes normalisés, et soit s un terme standard et normalisé. Posons δ le remplacement $[s \leftarrow 1]$, et $\sigma' = \sigma\delta$.[11] *On suppose que si $s = Exp(\cdot, \cdot)$, i.e. s est une exponentielle, alors $s \neq \sigma(x)$ pour toute variable $x \in Var$. Alors soit il existe un sous terme standard t de E tel que $t \sqsubseteq_\sigma s$, soit $\ulcorner E\sigma\urcorner = \urcorner E\sigma\urcorner\delta\urcorner$.*

Preuve. Supposons qu'il n'existe aucun sous terme standard t' de E tel que $t' \sqsubseteq_\sigma s$. On a alors $(E\sigma)\delta = E(\sigma\delta)$ et donc $\ulcorner E\sigma\urcorner = \ulcorner(E\sigma)\delta\urcorner$. Nous allons montrer, par induction sur la structure des termes, que pour tout $t \in STermes(E)$, on a $\ulcorner t\sigma\urcorner = \urcorner t\sigma\urcorner\delta\urcorner$, ce qui prouvera le lemme. Soit $t \in STermes(E)$ avec $|t|_{dag} = 1$ (point de départ de l'itération). On n'a que deux cas :

- Si $t \in Atomes$, alors $t \neq s$ par hypothèse. Ainsi, $\urcorner t\sigma\urcorner\delta\urcorner = t = \ulcorner t\sigma\urcorner$.
- Si $t \in Var$, alors $\ulcorner t\sigma\urcorner = t\sigma$, et donc $\ulcorner t\sigma\urcorner = \ulcorner(t\sigma)\delta\urcorner = \urcorner t\sigma\urcorner\delta\urcorner$.

Nous pouvons à présent examiner l'itération elle-même. Soit $t \in STermes(E)$ avec $|t|_{dag} > 1$ tel que tous ses sous termes t' vérifient $\ulcorner t'\sigma\urcorner = \urcorner t'\sigma\urcorner\delta\urcorner$. On a plusieurs cas, selon la structure de t :

- Si $t = \langle v, w\rangle$, alors $s \neq \langle \ulcorner v\sigma\urcorner, \ulcorner w\sigma\urcorner\rangle = \ulcorner t\sigma\urcorner$ puisque sinon $t \sqsubseteq_\sigma s$, et donc $\ulcorner t\sigma\urcorner = \langle \ulcorner v\sigma\urcorner, \ulcorner w\sigma\urcorner\rangle$. Ainsi, par induction on obtient $\ulcorner t\sigma\urcorner = \langle \urcorner v\sigma\urcorner\delta\urcorner, \urcorner w\sigma\urcorner\delta\urcorner\rangle$, et donc $\ulcorner t\sigma\urcorner = \urcorner\langle v\sigma, w\sigma\rangle\urcorner\delta\urcorner = \urcorner t\sigma\urcorner\delta\urcorner$ puisque $s \neq \ulcorner t\sigma\urcorner$. On traite les cas $t = \{u\}_v^s$ et $t = \{u\}_K^p$ exactement de la même manière.
- Si $t = v_1^{z_1} \cdot .. \cdot v_p^{z_p}$, alors :

$$\begin{array}{lll}
\ulcorner t\sigma\urcorner & = \ulcorner (v_1\sigma')^{z_1} \cdot .. \cdot (v_p\sigma')^{z_p}\urcorner & \\
& = \urcorner v_1\sigma\urcorner^{z_1} \cdot .. \cdot \ulcorner v_p\sigma\urcorner^{z_p}\urcorner & \\
& = \urcorner\urcorner v_1\sigma\urcorner\delta\urcorner^{z_1} \cdot .. \cdot \urcorner v_p\sigma\urcorner\delta\urcorner^{z_p}\urcorner & \text{par induction} \\
& = \ulcorner(\ulcorner v_1\sigma\urcorner\delta)^{z_1} \cdot .. \cdot (\ulcorner v_p\sigma\urcorner\delta)^{z_p}\urcorner & \\
& = \ulcorner(\ulcorner v_1\sigma\urcorner^{z_1} \cdot .. \cdot \ulcorner v_p\sigma\urcorner^{z_p})\delta\urcorner & (1) \\
& = \urcorner\urcorner v_1\sigma\urcorner^{z_1} \cdot .. \cdot \ulcorner v_p\sigma\urcorner^{z_p}\urcorner\delta\urcorner & (2) \\
& = \urcorner t\sigma\urcorner\delta\urcorner &
\end{array}$$

  En effet, on a (1) car s est standard et normalisé, et (2) est une conséquence immédiate du lemme 5.3.1.1.

[11] Remarque : σ' n'est pas forcément normalisée.

– Si $t = Exp(u, M)$, alors on a :

$$\begin{array}{lll}
\ulcorner t\sigma\urcorner & = \ulcorner Exp(u\sigma', M\sigma')\urcorner & \\
& = \ulcorner Exp(\ulcorner u\sigma'\urcorner, \ulcorner M\sigma'\urcorner)\urcorner & \\
& = \ulcorner Exp(\ulcorner\ulcorner u\sigma\urcorner\delta\urcorner, \ulcorner\ulcorner M\sigma\urcorner\delta\urcorner)\urcorner & \text{par induction} \\
& = \ulcorner Exp(\ulcorner u\sigma\urcorner\delta, \ulcorner M\sigma\urcorner\delta)\urcorner & \\
& = \ulcorner Exp(\ulcorner u\sigma\urcorner, \ulcorner M\sigma\urcorner)\delta\urcorner & (1) \\
& = \ulcorner\ulcorner Exp(\ulcorner u\sigma\urcorner, \ulcorner M\sigma\urcorner)\urcorner\delta\urcorner & (2)
\end{array}$$

On a (1) car $Exp(\ulcorner u\sigma\urcorner, \ulcorner M\sigma\urcorner) \neq s$. En effet, on aurait sinon $\ulcorner Exp(\ulcorner u\sigma\urcorner, \ulcorner M\sigma\urcorner)\urcorner = \ulcorner t\sigma\urcorner = s$, ce qui serait en contradiction avec l'hypothèse selon laquelle $t \not\sqsubseteq_\sigma s$. De plus, on a (2) grâce au lemme 5.3.1.1. En effet, les conditions d'application de ce lemme sont satisfaites car d'une part, $\ulcorner Exp(\ulcorner u\sigma\urcorner, \ulcorner M\sigma\urcorner)\urcorner = \ulcorner t\sigma\urcorner \neq s$, et d'autre part $\ulcorner u\sigma\urcorner \neq s$ si s est de la forme $Exp(\cdot,\cdot)$: si u n'est pas une variable, alors $u \in \mathcal{SP}$ avec $u \sqsubseteq_\sigma s$, d'où une contradiction avec les hypothèses, et si u est une variable et s est de la forme $Exp(\cdot,\cdot)$, alors par hypothèse $\ulcorner\sigma(u)\urcorner = \sigma(u) \neq s$.

□

5.3.3 Passage d'un remplacement sous une exponentielle.

Nous allons présenter ici une seconde application du lemme 5.3.1.1. Ce résultat va nous permettre, sous certaines conditions, de déplacer un remplacement appliqué à une exponentielle vers les sous termes maximaux de cette exponentielle.

Lemme 5.3.3.1 *Appliquer un remplacement sous une exponentielle.*

Soient $u, t, t', t_1, .., t_n$ des termes standards et normalisés avec $t = \ulcorner Exp(t', t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner$ et $u \neq t$, soient $z_1, .., z_n \in \boldsymbol{Z}$, et soit δ de remplacement $[u \leftarrow 1]$. De plus, si $t' = Exp(\cdot,\cdot)$ alors on suppose également que $u \neq t'$. Alors :

$$\ulcorner t\delta\urcorner = \ulcorner\ulcorner Exp(t', t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner\delta\urcorner = \ulcorner Exp(\ulcorner t'\delta\urcorner, \ulcorner t_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner t_n\delta\urcorner^{z_n})\urcorner$$

Preuve. Nous allons prouver ce lemme en deux temps, selon la forme de t' :

- Supposons que $t' = Exp(v, M)$, avec v un terme normalisé et M un produit normalisé. Il est important de remarquer que $v \neq Exp(\cdot,\cdot)$, i.e. v n'est pas une exponentielle, puisque t' est normalisé. On a alors :

$$\begin{array}{lll}
\ulcorner Exp(\ulcorner t'\delta\urcorner, \ulcorner t_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner t_n\delta\urcorner^{z_n})\urcorner & = \ulcorner Exp(\ulcorner v\delta\urcorner, \ulcorner M\delta\urcorner \bullet \ulcorner t_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner t_n\delta\urcorner^{z_n})\urcorner & (1) \\
& = \ulcorner Exp(v\delta, M\delta \bullet (t_1\delta)^{z_1} \cdot .. \cdot (t_n\delta)^{z_n})\urcorner & \\
& = \ulcorner Exp(v, M \bullet t_1^{z_1} \cdot .. \cdot t_n^{z_n})\delta\urcorner & (2) \\
& = \ulcorner\ulcorner Exp(v, M \bullet t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner\delta\urcorner & (3) \\
& = \ulcorner t\delta\urcorner &
\end{array}$$

Pour (1), on utilise le fait que $u \neq t'$, et donc que soit $\ulcorner t'\delta\urcorner = \ulcorner v\delta\urcorner$ (avec $\ulcorner M\delta\urcorner = 1$ dans ce cas), soit $\ulcorner t'\delta\urcorner = Exp(\ulcorner v\delta\urcorner, \ulcorner M\delta\urcorner)$.
Pour (2), on utilise le fait que $u \neq t$: Si $u = Exp(v, M \bullet t_1^{z_1} \cdot .. \cdot t_n^{z_n})$, alors $Exp(v, M \bullet t_1^{z_1} \cdot .. \cdot t_n^{z_n})$ est nécessairement normalisé puisque u est normalisé. Mais on aurait alors $u = Exp(v, M \bullet t_1^{z_1} \cdot .. \cdot t_n^{z_n}) = \ulcorner Exp(v, M \bullet t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner = t$, ce qui est en contradiction avec $u \neq t$.
Pour (3), on utilise le fait que $v \neq Exp(\cdot,\cdot)$ et $u \neq t$, ainsi que le lemme 5.3.1.1.

- A présent, supposons que $t' \neq Exp(\cdot,\cdot)$. Il nous suffit d'appliquer le même genre d'arguments que ci-dessus, en remplaçant v par t' et en supprimant $\ulcorner M\delta \urcorner$, $M\delta$ et M, pour les cas $u \neq t'$ et $u = t'$.

Ainsi, on a prouvé le lemme quelle que soit la forme de t'. □

5.4 L'intrus DH définit un ensemble de règles d'oracle.

Pour prouver que les règles de l'intrus DH (définition 2.4.5.5) forment un ensemble de règles d'oracle, nous avons besoin de quelques conventions de notation. Tout d'abord, dans toute règle L_{DHd} ou L_{DHc} telle que $t_0, ..t_n \rightarrow \ulcorner Exp(t_0, t_1^{z_1} \cdot .. \cdot t_n^{z_n} \urcorner = u$, on appellera $z_1, .., z_n$ les exposants de cette règle, et t_0 la tête de cette règle. t_0 est le terme particulier sur lequel est construit l'exponentiation. On supposera toujours que la tête d'une règle DH de décomposition est toujours de la forme $Exp(\cdot,\cdot)$, car sinon une telle règle ne créerait que $u = t_0$, et serait donc inutile pour l'intrus. De la même manière, dans toute règle DH de la forme précédente, on suppose que $t_i \neq t_j$ pour $i \neq j$ et $0 < i, j$ puisque l'on pourrait sinon rassembler les coefficients z_i et z_j. De plus, nous noterons $L_o = L_{DHc} \cup L_{DHd}$. Pour toute règle $L \in L_{DY} \cup L_{DHd} \cup L_{DHc}$ telle que $L = M \rightarrow t$, on dit que L génère le message t. Enfin, on dit qu'une règle L_i précède une règle L_j dans une dérivation $D = E \rightarrow_{L_1} .. \rightarrow_{L_n} E, t_1, .., t_n$ quand $i < j$.

5.4.1 L'intrus DH admet des dérivations bien formées.

De manière à prouver que l'intrus DH admet toujours des dérivations bien formées, nous commençons par montrer le lemme suivant.

Lemme 5.4.1.1 *Dérivations à têtes de règles DH générées par DH.*

Soit E un ensemble fini de messages standards et normalisés, et soit t un message tel que $t \in forge_{DH}(E)$. Soit D une dérivation partant de E et de but t. Alors il existe une dérivation $D' = E \rightarrow_{L_1} .. \rightarrow_{L_n} E, t_1, .., t_n$ partant de E et de but t telle que :

1. *D' est de même longueur que D.*
2. *Pour toute règle $L_i \in L_{DHc}$ de tête t', soit $t' \in E$ soit il existe une règle $L_j \in L_{DY}$ telle que $t_j = t'$, pour $i, j \in \{1, .., n\}$.*
3. *Pour toute règle $L_i \in L_{DHd}$ de tête t', soit $t' \in E$ soit il existe une règle $L_j \in L_d$ telle que $t_j = t'$, pour $i, j \in \{1, .., n\}$.*

Preuve. Soient E, t, et D définis comme ci-dessus. Nous allons construire D' à partir de D. Pour cela, supposons qu'il existe $L \in D \cap L_o$ de tête t' telle que $t' \notin E$ et aucune règle $L' \in D \cap L_{DY}$ ne génère t'. Alors nécessairement il existe $L' \in D \cup L_o$ de tête t'. Supposons que L soit de la forme $t', t_1, .., t_n \rightarrow t$ avec $z_1, .., z_n$ comme exposants, et que L' soit de la forme $t'', t'_1, .., t'_{n'} \rightarrow t'$ avec $z'_1, .., z'_{n'}$ comme exposants. Alors nécessairement $t = \ulcorner Exp(t'', t_1^{z_1} \cdots t_n^{z_n} \cdot t'^{z'_1}_1 \cdots t'^{z'_{n'}}_{n'}) \urcorner$ et donc t peut être généré par la règle DH $\hat{L} = t'', t_1, \ldots, t_n, t'_1, .., t'_{n'} \rightarrow t$ de tête t''. En conséquence, on peut remplacer la règle L par $\hat{L}$ dans D tout en conservant une dérivation de même ensemble de départ, de même but, et de même longueur. Il nous suffit alors d'itérer ce remplacement pour obtenir la dérivation D' satisfaisant naturellement le point n°1 ci-dessus, et telle que pour toute règle DH $L \in D' \cap L_o$ de tête t', il n'existe aucune règle $L' \in D' \cap L_o$ générant t' et précédant L dans D'. On en déduit donc naturellement les points n°2 et 3 ci-dessus. En particulier, si L est une règle DH de décomposition, alors t' est de la forme $Exp(\cdot,\cdot)$, et donc ne peut pas être généré par une règle de L_c.

Ainsi, nous pourrons utiliser un genre particulier de dérivations (pas encore bien formées, mais on s'en rapproche), où les règles DH n'utilisent comme terme de tête que des éléments de l'ensemble de départ ou des termes créé par une règle de Dolev-Yao. Le lemme suivant va nous fournir un critère pour savoir quand une dérivation donnée est bien formée, ce qui nous permettra d'identifier des relations bien formées pour chaque relation de la forme $t \in forge_{DH}(E)$. □

Lemme 5.4.1.2 *Critère pour qu'une dérivation soit bien formée.*

Soit $D = E \rightarrow_{L_1} ... \rightarrow_{L_n} E, t_1, .., t_n$ une dérivation (de but t_n). On a les deux propriétés suivantes :

1. *Supposons que pour tout $j \in \{1, .., n\}$ tel que $L_j \in L_d \cup L_{DHd}$, il existe $t' \in E, t_1, .., t_{j-1}$ avec que t_j un sous terme de t' et soit $t' \in E$, soit il existe $i < j$ avec $t_i = t'$ et $L_i \in L_d \cup L_{DHd}$. Alors pour toute règle $L_i \in L_d \cup L_{DHd}$, on a t_i sous terme de E.*
2. *Supposons que pour tout $i < n$ tel que $L_i \in L_c \cup L_{DHc}$, il existe $j > i$ tel que $t_i \in STermes(E, t_j)$. Alors pour toute règle $L_i \in L_c \cup L_{DHc}$, on a t_i sous terme de E, t_n.*

De plus, si les hypothèses des deux propriétés précédentes sont vérifiées, alors la dérivation D est bien formée.

Preuve. Pour le point n°1, il nous suffit de faire une simple induction sur $j \in \{1, .., n\}$, puisque soit $t' \in E$ ce qui prouve le point, soit t_j est sous terme de t_i, avec $i < j$ et $L_i \in L_d \cup L_{DHd}$, ce qui permet d'itérer sur L_i. Pour le point n°2, supposons que ses hypothèses soient vérifiées. On va prouver par induction sur $n - i$ que pour tout $i \in \{1, \ldots, n\}$, si $L_i \in L_c \cup L_{DHc}$ alors $t_i \in STermes(E, t_n)$. Nous avons deux cas pour cela. Si $n - i = 0$, alors $t_i = t_n$ et donc $t_i \in STermes(E, t_n)$. Sinon, (pas de l'induction) par hypothèse du point n°2, il existe $j > i$ tel que $t \in STermes(E, t_j)$. De plus, si $L_j \in L_d \cup L_{DHd}$, alors t_j est un sous terme de E (cas ci dessus), et si $L_j \in L_c \cup L_{DHc}$, alors par induction t_j est un sous terme de E, t_n, et donc t_i aussi.

Les deux points du lemme sont donc démontrés. De plus, si les hypothèses de ces deux propriétés sont vérifiées, alors on peut conclure immédiatement que D est une dérivation bien formé par application directe de la définition. □

Les deux lemmes précédents vont nous permettre de montrer que l'intrus DH admet toujours des dérivations bien formées. Ainsi :

Proposition 5.4.1.3 *L'intrus DH admet des dérivations bien formées.*

Pour tout ensemble E de messages standards et normalisés, et tout message g standard et normalisé t.q. $g \in forge_{DH}(E)$, il existe une dérivation bien formée partant de E et de but g.

Preuve. Soit $D = E \rightarrow_{L_1} \ldots \rightarrow_{L_n} E, t_1, .., t_n$ une dérivation de but $g = t_n$ de taille minimale parmi toutes les dérivations partant de E et de but g. Comme le lemme 5.4.1.1 ne change pas la taille des dérivations, on peut supposer que D vérifie les propriétés 2 et 3 de ce lemme. Notre but à présent est de démontrer que D satisfait les conditions des deux propriétés du lemme 5.4.1.2. En effet :

Propriété n°1 :

Tout d'abord, considérons le cas où $L_j \in L_d(s)$ (et donc $t \in STermes(s)$). Alors on a $L_i \notin L_{DHc}(s)$ pour tout $i < j$, puisque les règles de L_{DHc} ne créent pas de terme standard, et $L_i \notin L_c(s)$ pour tout $i < j$, puisque la règle L_j ne peut pas être inutile (i.e. $t_j \notin E, t_1, .., t_{j-1}$) par définition d'une dérivation. En conséquence, on a soit $s \in E$ soit il existe $i < j$ tel que $L_i \in L_d \cup L_{DHd}$ et $t_i = s$.

A présent, supposons que $L_j \in L_{DHd}$ de tête t'. Par définition des règles de décomposition DH, on se rend compte que $t_j \in STermes(t')$. Ainsi, grâce à la propriété n°3 du lemme 5.4.1.1, on obtient que soit $t' \in E$, soit il existe j tel que $L_j \in L_d(t')$.

On a donc montré que l'hypothèse de la propriété n°1 du lemme 5.4.1.2 est vérifiée. En conséquence, pour toute règle $L_i \in L_d \cup L_{DHd}$, on a t_i sous terme de E.

Propriété n°2 :

Si $L_i \in L_c \cup L_{DHc}$ pour $i < n$, alors par minimalité de D, il existe $j > i$ tel que t_i appartienne au membre gauche de la règle L_j. On a 3 cas :

Si $L_j \in L_d$, alors comme dans le cas de la proposition 1, on obtient $t_i \in STermes(E)$.

Si $L_j \in L_c$, alors $t_i \in STermes(t_j)$.

Si $L_j \in L_o$, alors nous avons plusieurs (sous) cas possibles :

- Supposons tout d'abord que t_i soit la tête de la règle L_j. Dans ce cas, le Lemme 5.4.1.1 (points 2 et 3) implique qu'il existe k tel que $L_k \in L_{DY}$ et $t_k = t_i$. Par définition d'une dérivation, on a nécessairement $k = i$, et donc $L_k \in L_c$. Ainsi, t_i n'est pas de la forme $Exp(\cdot, \cdot)$ (i.e. $t_i \neq Exp(a, b)$ pour tous a, b). On a donc, $t_i \in STermes(t_j)$, par définition des règles DH.

- Supposons à présent que t ne soit pas la tête de la règle L_j. Si t_i n'est pas un sous terme de t_j, alors t_i est un sous terme de h avec h la tête de la règle L_j. De plus, h est de la forme $Exp(\cdot, \cdot)$ (Sinon, t_i ne pourrait pas être éliminé par normalisation de t_j). Grâce au lemme 5.4.1.1, on obtient que soit $h \in E$, soit il existe une règle $L_k \in L_{DY}$ générant h. Comme h est de la forme $Exp(\cdot, \cdot)$, on en déduit que $L_k \in L_d$, et l'on peut utiliser la proposition n°1 ci-dessus : $h = t_k$ est un sous terme de E, et donc t_i aussi.

Ainsi, nous avons montré que la dérivation D, obtenue à partir du lemme 5.4.1.1, vérifie les hypothèses des deux propriétés du lemme 5.4.1.2. En conséquence, D est une dérivation bien formée, et la proposition est démontrée. □

5.4.2 Compatibilité entre remplacement et règles DH.

Le lemme de la sous section précédente nous permettra de montrer le premier point de la définition de règles d'oracle. Cependant, pour les autres (et en particulier pour le point n°3 de la définition de règles d'oracle) nous avons besoin d'un lemme technique préalable.

Lemme 5.4.2.1 *Pas de cycle face à la normalisation*[12].

Soient $z_1, .., z_n \in \mathbb{Z}\backslash\{0\}$, et soient $s, s_1, .., s_n$ des termes standards et normalisés tels que pour tout $i \in \{1, .., n\}$, $s_i \notin \{1, u\} \cup \{s_j \mid j \neq i\}$ avec $u = \ulcorner Exp(s, s_1^{z_1} \cdot .. \cdot s_n^{z_n}) \urcorner$, et $s \neq u = Exp(\cdot, \cdot)$. Posons δ le remplacement $[u \leftarrow 1]$. Alors :

$$u = \ulcorner Exp(s, s_1^{z_1} \cdot .. \cdot s_n^{z_n}) \urcorner = \ulcorner Exp(\ulcorner s\delta \urcorner, \ulcorner s_1\delta \urcorner^{z_1} \cdot .. \cdot \ulcorner s_n\delta \urcorner^{z_n}) \urcorner$$

Preuve. On sait par hypothèse que u est un terme standard et normalisé. Nous allons prouver le lemme en deux temps, selon la nature de s.

Supposons que $s \neq Exp(\cdot, \cdot)$. On a alors $u = Exp(s, s_1^{z_1} \cdot .. \cdot s_n^{z_n})$, puisque tous les s_i sont normalisés, non réduits à 1, et différents deux à deux (i.e. $s_1^{z_1} \cdot .. \cdot s_n^{z_n}$ n'est pas normalisable). En conséquence, $u \notin STermes(s, s_1, .., s_n)$, et donc $s = s\delta$ et $s_i = s_i\delta$ pour tout i. On obtient ainsi $u = \ulcorner Exp(\ulcorner s\delta \urcorner, \ulcorner s_1\delta \urcorner^{z_1} \cdot .. \cdot \ulcorner s_n\delta \urcorner^{z_n}) \urcorner$, et le lemme est démontré dans ce cas.

[12] Ce titre n'est pas parfait. Une idée ?

Supposons à présent que $s = Exp(v, M)$, pour v un terme standard normalisé ($v \neq Exp(\cdot,\cdot)$ car s est normalisé) et M un produit normalisé. De plus, comme $u = Exp(\cdot,\cdot)$, on a :

$$u = Exp(v, \ulcorner M \bullet s_1^{z_1} \cdot .. \cdot s_n^{z_n} \urcorner) \text{ avec } \ulcorner M \bullet s_1^{z_1} \cdot .. \cdot s_n^{z_n} \urcorner \neq 1 \text{ et } u \notin STermes(v) \ .$$

Posons $E = Facteur(M) \cup \{s_1, .., s_n\}$. Grâce à la définition de normalisation, il existe un ensemble $E' = \{t_1, .., t_{n'}\} \subseteq E$ et des coefficients $z'_1, .., z'_{n'} \in \mathbf{Z}\backslash\{0\}$ tels que $u = Exp(v, t_1^{z_1} \cdot .. \cdot t_{n'}^{z_{n'}})$ et $u \notin STermes(E', v)$.

Nous allons tout d'abord montrer que $\ulcorner\urcorner M\delta\urcorner \bullet \ulcorner s_1\delta\urcorner^{z_1} \cdots \ulcorner s_n\delta\urcorner^{z_n}\urcorner = t_1^{z_1} \cdot .. \cdot t_{n'}^{z_{n'}}$. Pour cela, posons $M = s_{n+1}^{z_{n+1}} \cdot .. \cdot s_m^{z_m}$ et $C_i = \{j \in \{1, .., m\} \mid s_j = s'_i\}$. On a alors $z'_i = \Sigma_{j \in C_i} z_j$ pour tout i. Posons $C = \bigcup_{i=1}^{n'} C_i$. En conséquence, on a $\ulcorner s_1^{z_1} \cdot .. \cdot s_n^{z_n} \cdot s_{n+1}^{z_{n+1}} \cdot .. \cdot s_{n''}^{z_{n''}}\urcorner = \ulcorner \Pi_{i=1}^{n'} \Pi_{j \in C_i} s_j^{z_j}\urcorner = \Pi_{i=1}^{n'} t_i^{z'_i}$ et $\ulcorner \Pi_{j \notin C} s_j^{z_j}\urcorner = 1$ (avec Π le produit selon l'opérateur $\cdot$). De plus, comme les s_j sont normalisés, on obtient $\ulcorner \Pi_{j \notin C} s_j \delta^{z_j}\urcorner = \ulcorner \Pi_{j \notin C} \ulcorner s_j\delta^{z_j}\urcorner = 1$. Et comme $t_i\delta = t_i$ pour tout i, on a bien $\ulcorner\urcorner M\delta\urcorner \bullet \ulcorner s_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner s_n\delta\urcorner^{z_n}\urcorner = t_1^{z_1} \cdot .. \cdot t_{n'}^{z_{n'}}$.

Il ne nous reste alors plus qu'à remarquer que $s\delta = Exp(v\delta, M\delta)$, $u \neq M$, $u \notin STermes(v)$, et $u \neq s$ pour pouvoir appliquer cette égalité sous l'opérateur d'exponentiation et obtenir :

$$\ulcorner Exp(\ulcorner s\delta\urcorner, \ulcorner s_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner s_n\delta\urcorner^{z_n})\urcorner = \ulcorner Exp(\ulcorner v\delta\urcorner, \ulcorner M\delta\urcorner \bullet \ulcorner s_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner s_n\delta\urcorner^{z_n})\urcorner = u$$

ce qui prouve le lemme. □

Nous avons à présent tous les outils nécessaires pour prouver que l'intrus DH définit bien des règles d'oracle. Concrètement :

Théorème 5.4.2.2 *Les règles de l'intrus DH sont des règles d'oracle.*

Preuve. Nous allons examiner les trois points de la définition de règles d'oracle une par une, en utilisant $L_{oc} = L_{DHc}$ et $L_{od} = L_{DHd}$.

1. Le premier point de la définition est une conséquence directe du lemme 5.4.1.3.
2. Par définition, aucun terme créé avec une règle L_{DHc} ne peut être décomposé avec une règle L_d, puisque le terme généré est non standard.
3. Soient u un message standard et normalisé, F un ensemble de messages standards et normalisés tel que $1 \in F$, et t un message standard tel que $F \rightarrow_{L_c \cup L_{DHc}} F, u$ et $F, u \rightarrow_{L_o} F, u, t$. Posons δ le remplacement $[u \leftarrow 1]$. Si $u = t$, alors $t\delta = 1 \in forge_{DH}(F\delta)$, et le point 3 de la définition est prouvé. On supposons donc que $u \neq t$. Comme $F, u \rightarrow_{L_o} F, u, t$, il existe $t', t_1, .., t_n \in F$ et $z_1, .., z_n \in \mathbf{Z}\backslash\{0\}$ tels que $t_i \neq t_j$ pour tout $i \neq j$ et $t = \ulcorner Exp(t', t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner$. On a deux cas :
   (a) Si $t' \neq Exp(\cdot,\cdot)$ ou $u \neq t'$, alors grâce au lemme 5.3.3.1, on a :
   $$\ulcorner t\delta\urcorner = \ulcorner Exp(\ulcorner t'\delta\urcorner, \ulcorner t_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner t_n\delta\urcorner^{z_n})\urcorner$$
   et donc $\ulcorner t\delta\urcorner \in forge_{DH}(\ulcorner F\delta\urcorner)$.
   (b) Sinon, $u = t' = Exp(v, M)$. On a alors :

$$\begin{array}{lll}
\ulcorner t\delta\urcorner & = \ulcorner\urcorner Exp(v, M \bullet t_1^{z_1} \cdot .. \cdot t_n^{z_n})\urcorner\delta\urcorner & \\
& = \ulcorner Exp(v, M \bullet t_1^{z_1} \cdot .. \cdot t_n^{z_n})\delta\urcorner & (*) \\
& = \ulcorner Exp(v\delta, M\delta \bullet (t_1\delta)^{z_1} \cdot .. \cdot (t_n\delta)^{z_n})\urcorner & \text{car } u \neq t \\
& = \ulcorner Exp(v, M \bullet (t_1\delta)^{z_1} \cdot .. \cdot (t_n\delta)^{z_n})\urcorner & \text{car } u \notin STermes(M, v) \\
& = \ulcorner Exp(v, M \bullet \ulcorner t_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner t_n\delta\urcorner^{z_n})\urcorner & \\
& = \ulcorner Exp(u, \ulcorner t_1\delta\urcorner^{z_1} \cdot .. \cdot \ulcorner t_n\delta\urcorner^{z_n})\urcorner &
\end{array}$$

En effet, pour $(*)$ on applique le lemme 5.3.1.1 en remarquant que $v \neq u$ et $u \neq t$. A présent, pour prouver que $\ulcorner t\delta \urcorner \in forge_{DH}(\ulcorner F\delta \urcorner)$ il nous suffit de montrer que $u \in forge_{DH}(\ulcorner F\delta \urcorner)$. On sait déjà que $F \rightarrow_{L_c \cup L_{DHc}} F, u$, et comme $u = Exp(\cdot,\cdot)$, on a nécessairement $F \rightarrow_{L_{DHc}} F, u$. Ainsi, il existe des termes normalisés $s, s_1, .., s_n \in F$ et des coefficients $z'_1, .., z'_{n'} \in \boldsymbol{Z}\backslash\{0\}$ tels que s et les s_i satisfassent les conditions du lemme 5.4.2.1 avec $u = \ulcorner Exp(s, s_1^{z_1} \cdot .. \cdot s_n^{z_n}) \urcorner$. En conséquence, on obtient $u = \ulcorner Exp(\ulcorner s\delta \urcorner, \ulcorner s_1\delta \urcorner^{z_1} \cdot .. \cdot \ulcorner s_n\delta \urcorner^{z_n}) \urcorner$, et donc $u \in forge_{DH}(\ulcorner F\delta \urcorner)$.

Ainsi, les trois points de la définition de règles d'oracle sont vérifiés, et l'intrus DH définit donc bien des règles d'oracle. □

5.4.3 Décider de l'applicabilité d'une règle DH.

Grâce à la définition de règles d'oracle, nous pourrons (dans les sections suivantes) borner les tailles DAG des substitutions d'attaques minimales, et plus tard borner les coefficients des produits de ces substitutions. Cependant, comme pour l'intrus xor, l'algorithme de décision de l'insécurité que nous présenterons aura besoin de vérifier qu'une suite de règles données forme bien une substitution, et donc qu'une règle donnée peut bien être appliquée à un ensemble de messages donnés. Pour n'importe quel ensemble L_o de règles d'oracle, nous définissons :

$$Application(L_o) \;=\; \{\, (E,t) \,|\, E \rightarrow_{L_o} E, t \,\}$$

avec E et t donnés sous forme de DAG. Pour que l'algorithme de décision soit NP, nous aurons besoin de savoir que l'appartenance à cet ensemble peut être décidé en temps (déterministe) polynomial en $|E,t|_{dag}$ pour les règles d'oracle considérées. Ainsi, nous avons :

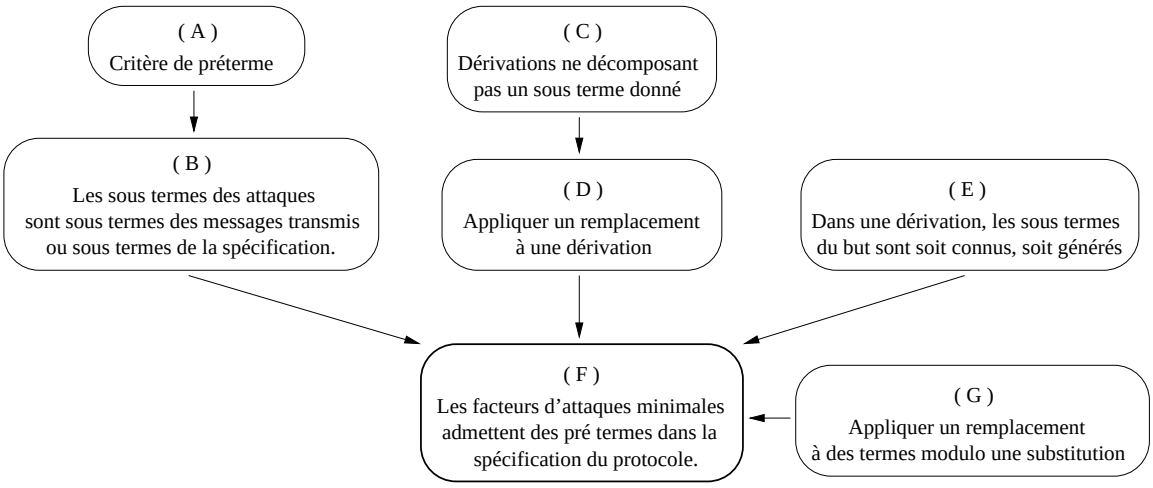
Proposition 5.4.3.1 *Décision de l'applicabilité pour DH.*
Le problème (d'appartenance à) Application(L_{DH}) est décidable en temps (déterministe) polynomial en $|E,t|_{dag}$.

Preuve. On doit donner un algorithme qui, étant donnés E et t, décide de l'existence de $t', t_1, .., t_n \in E$ et $z_1, .., z_n \in \boldsymbol{Z}$ tels que $t = \ulcorner Exp(t', t_1^{z_1} \cdot .. \cdot t_n^{z_n}) \urcorner$. Pour cela, on donne une caractérisation de ce problème : On a $E \rightarrow_{L_{DH}} E, t$ ssi :

1. soit $t \neq Exp(\cdot,\cdot)$ et :
   (a) $t \in E$, ou
   (b) il existe M tel que $Exp(t, M) \in E$ et $Facteur(M) \subseteq E$
2. soit $t = Exp(v, M)$ et :
   (a) $v \in E$ et $Facteur(M) \subseteq E$, ou
   (b) il existe M' tel que $Exp(v, M) \in E$ et $Facteur(\ulcorner M' \bullet M^{-1} \urcorner) \subseteq E$ (i.e. l'ensemble des t' tels que les exposants produits de t' dans M' et M sont différents est inclus dans E).

Tous ces points sont simples à vérifier. De plus, on en déduit un algorithme de décision de $E \rightarrow_{L_o} E, t$ en temps polynomial en $|E,t|_{dag}$ car chaque vérification est polynomiale en $|E,t|_{dag}$. □

Ainsi, on peut décider de l'applicabilité d'une règle de DH en temps polynomial en la taille des termes la composant. Ceci nous servira pour décider si $t \in forge_{DH}(E)$, avec t un message et E un ensemble de messages (problème *Derive*).



avec A : Lemme 5.5.1.1 B : Lemme 5.5.1.2 C : Lemme 5.5.2.1 D : Lemme 5.5.2.2
E : Lemme 5.5.2.3 F : Proposition 5.5.3.1 G : Lemme 5.3.2.1

FIG. 5.1: Schéma de preuve pour la section 5.5.

5.5 Caractérisation des facteurs de substitution d'attaques minimales.

Dans cette section, nous voulons montrer une propriété des facteurs des substitutions d'attaques minimales, à savoir qu'ils admettent tous au moins un pré terme dans l'ensemble des sous termes de la spécification du protocole ($\mathcal{SP}$). Il s'agit de la propriété centrale nous permettant ensuite de montrer que les tailles DAG des substitutions d'attaques minimales sont bornées polynomialement par le nombre de sous termes de la spécification du protocole. Pour rendre le schéma de preuve conduisant au résultat de cette section plus intuitif, nous en donnons un diagramme de dépendance à la Figure 5.1. Une flèche d'une propriété à une autre indique une propriété utilisée pour en prouver une autre. On remarque que l'on a d'ores et déjà prouvé le lemme 5.3.2.1 de ce schéma, i.e. la propriété G dans le diagramme. Nous allons parcourir ce diagramme en trois temps. Tout d'abord, nous prouverons les propriétés A et B relatives aux substitutions d'attaques minimales à la sous section 5.5.1. Ensuite, nous montrerons les propriétés C, D et E relatives aux dérivations à la sous section 5.5.2. Enfin, nous pourrons prouver la propriété F (le but de tout ceci) à la sous section 5.5.3.

5.5.1 Propriétés des substitutions d'attaques minimales.

Nous commençons par prouver la propriété A, car celle-ci est nécessaire à la preuve de la propriété B. Ce lemme nous montre qu'un sous terme s d'un terme $\ulcorner t\sigma \urcorner$ admet un pré terme dans $STermes(t)$ à partir du moment où il n'est pas présent dans les valeurs de la substitution σ. Formellement :

Lemme 5.5.1.1 *Critère de pré terme.*

Soient t un terme générique normalisé, et σ une substitution normalisée. De plus, soit s un sous terme standard de $\ulcorner t\sigma \urcorner$ tel que $s \notin STermes(\sigma(x))$ pour toute variable $x \in Var$. Alors il existe un sous terme standard t' de t tel que $t' \sqsubseteq_\sigma s$.

PREUVE. Soit t' l'un des sous termes minimaux de t (selon l'ordre sous terme) tel que $s \in STermes(\ulcorner t'\sigma \urcorner)$. Un tel sous terme existe, puisque $s \in STermes(\ulcorner t\sigma \urcorner)$. Nous allons montrer que cela suffit pour avoir $t' \sqsubseteq_\sigma s$, i.e. $\ulcorner t'\sigma \urcorner = s$ puisque t' ne peut pas être une variable (sinon s serait sous terme d'une valeur de σ). Pour montrer cette égalité, on distingue plusieurs cas, selon la structure du terme t' :

- Supposons tout d'abord que $t' = \langle u, v \rangle$. On a alors naturellement $\ulcorner t'\sigma \urcorner = \langle \ulcorner u\sigma \urcorner, \ulcorner v\sigma \urcorner \rangle$. De plus, par définition de t', on a $s \notin STermes(\ulcorner u\sigma \urcorner)$ et $s \notin STermes(\ulcorner v\sigma \urcorner)$. Mais on a aussi $s \in STermes(\ulcorner t'\sigma \urcorner)$, d'où $\ulcorner t'\sigma \urcorner = s$. Le lemme est donc prouvé dans ce cas. De la même manière, le lemme est également vrai pour $t' = \{u\}_v^s$ ou $t' = \{u\}_v^p$.
- Supposons à présent que $t' = Exp(u, M)$ avec $M = t_1^{z_1} \cdot .. \cdot t_p^{z_p}$ (les t_i sont standards). Par définition de t', on ait que $s \notin STermes(\ulcorner u\sigma \urcorner)$ et $s \notin STermes(\{\, \ulcorner t_i\sigma \urcorner \,|\, i \in \{1, .., p\}\,\})$. On a deux cas :
  - Si $\ulcorner u\sigma \urcorner$ n'est pas de la forme $Exp(\cdot, \cdot)$, alors on a sait $\ulcorner t'\sigma \urcorner = Exp(\ulcorner u\sigma \urcorner, \ulcorner M\sigma \urcorner)$ soit $\ulcorner t'\sigma \urcorner = \ulcorner u\sigma \urcorner$. Mais les deux cas, on a $\ulcorner t'\sigma \urcorner = s$ puisque s est standard.
  - Sinon, on a $\ulcorner u\sigma \urcorner = Exp(v, M')$, et donc soit $\ulcorner t'\sigma \urcorner = Exp(v, \ulcorner M' \bullet M\sigma \urcorner)$ soit $\ulcorner t'\sigma \urcorner = v$. De plus, on sait que s n'est ni un sous terme de $\{\, \ulcorner t_i\sigma \urcorner \,|\, i \in \{1, .., p\}\,\}$, ni un sous terme de M'. En conséquence, on a $\ulcorner M' \bullet M\sigma \urcorner$, puisque s est standard. Ainsi, avec s sous terme de v, on obtient $\ulcorner t'\sigma \urcorner = s$ dans les deux cas.
- Enfin, supposons que $t' = t_1^{z_1} \cdot .. \cdot t_p^{z_p}$, avec $p > 0$. Si $s \in STermes(\ulcorner t'\sigma \urcorner) \backslash \{\ulcorner t'\sigma \urcorner\}$, alors il existe i tel que s soit un sous terme de $\ulcorner t_i\sigma \urcorner$, ce qui donne un contradiction avec la définition de t'. On a donc nécessairement $\ulcorner t'\sigma \urcorner = s$.

□

Nous pouvons à présent montrer la propriété B (lemme suivant), dont nous aurons besoin pour la proposition centrale de cette section. Ce lemme établit que tout sous terme d'une substitution formant une attaque sur P soit admet un pré terme dans $\mathcal{SP}$, soit est sous terme d'un message $\ulcorner R_i\sigma \urcorner$ reçu par un principal, modulo quelques hypothèses simples. Concrètement :

Lemme 5.5.1.2 *Les sous termes des attaques sont transmis ou partiellement connus.*

Soit (π, σ) une attaque sur P (avec σ normalisé), soient $i \in \{1, .., \#\pi\}$ et $x \in Var(R_i)$, et soit s un sous terme de $\ulcorner R_i\sigma \urcorner$. Si $s = Exp(\cdot, \cdot)$, on suppose de plus que $s \neq \sigma(y)$ pour tout $y \in Var$. Alors soit il existe $j \leq i$ tel que $s \in STermes(\ulcorner R_j\sigma \urcorner)$, soit il existe $t \in \mathcal{SP}$ tel que $t \sqsubseteq_\sigma s$.

PREUVE. Tout d'abord, supposons que $s = 1$. Or par hypothèse (de ce chapitre), 1 appartient aux connaissances initiales de l'intrus, et on a donc immédiatement $1 \in \mathcal{SP}$ avec $1 \sqsubseteq_\sigma s$.

On suppose donc que $s \neq 1$ (et que i, x, et s vérifient les hypothèses du lemme). On suppose également que pour tout $j \leq i$, s n'est pas un sous terme de $\ulcorner R_j\sigma \urcorner$ (sinon le lemme est prouvé). Nous allons montrer qu'il existe ainsi un pré terme de s dans $\mathcal{SP}$. Posons :

$$j = \min\{i' \,|\, y \in Var(R_{i'}) \text{ et } s \text{ sous terme de } \sigma(y)\}$$

i.e. le premier pas i' de protocole où s apparaît comme sous terme de la valeur d'une variable apparaissant dans $R_{i'}$. Attention, la normalisation peut très bien éliminer s des sous termes de

$\ulcorner R_j\sigma\urcorner$. C'est d'ailleurs le cas ici : on a $j \leq i$ par minimalité de j, et donc s n'est pas un sous terme de $\ulcorner R_j\sigma\urcorner$ par hypothèse. De plus, posons :

$$y \in Var(R_j) \text{ telle que } s \text{ sous terme de } \sigma(y)$$

$$t \text{ sous terme maximal de } R_j \text{ t.q. } y \in Var(t) \text{ et } s \text{ sous terme de } \ulcorner t\sigma\urcorner$$

avec t maximal selon l'ordre sous terme, i.e. pour tout autre sous terme t' de R_j vérifiant ces mêmes propriétés, t n'est pas un sous terme de t'. Ce terme t existe, car y lui-même est un candidat (probablement pas maximal) pour t. De plus, on sait que $t \neq R_j$ par hypothèse sur s. Posons :

$$r \text{ sous terme minimal de } R_j \text{ t.q. } t \text{ sous terme propre de } r$$

avec r minimal selon l'ordre sous terme. r est le sous terme de R_j, parent de t, le plus proche de t. Intuitivement, le couple $(r\sigma, t\sigma)$ définit dans $R_j\sigma$ une limite où s est éliminé par normalisation (même s'il est éliminé en tant que sous terme d'un terme plus important). Par exemple, on peut avoir $r\sigma = Exp(a, t\sigma \cdot t')$ avec $\ulcorner t\sigma\urcorner = \ulcorner t'\urcorner$: le terme $t\sigma$ vérifiant $s \in STermes(\ulcorner t\sigma\urcorner)$ est éliminé par normalisation. On peut alors remarquer que comme s n'est pas un sous terme de $\ulcorner r\sigma\urcorner$, r est nécessairement de la forme $v_1^{z_1} \cdots v_p^{z_p}$ ou $Exp(u, v_1^{z_1} \cdots v_p^{z_p})$ avec u et v_i standards, pour permettre l'élimination de s par normalisation. De plus, comme r est un sous terme d'un terme standard (non produit) R_j, il ne peut pas être lui-même un produit. En conséquence, on n'a que deux cas :

1. Si $r = Exp(t, M)$ (i.e. $u = t$), grâce à la définition de protocole bien formé 2.4.4.2, point 2, et par minimalité de j, on a $Var(M) \subseteq Var(R_1, .., R_{j-1})$ (sinon, on aurait $Var(t) \subseteq Var(R_1, .., R_{j-1})$ et j ne serait pas minimal). Ainsi, comme s n'est pas un sous terme de $\ulcorner r\sigma\urcorner$, on en déduit que $\ulcorner t\sigma\urcorner = Exp(\cdot, \cdot)$. On a alors deux cas :
   - Si $s = \ulcorner t\sigma\urcorner = Exp(v, M')$, par hypothèse on a $s \neq \sigma(z)$ pour toute variable z, et en conséquence t n'est pas une variable. On a alors $t \in \mathcal{SP}$ et $t \sqsubseteq_\sigma s$, ce qui prouve le lemme dans ce cas.
   - Si $s \neq \ulcorner t\sigma\urcorner = Exp(v, M')$, alors s est soit un sous terme de v, soit un sous terme de M'. Le premier cas (s sous terme de v) ne peut pas se produire, car sinon on aurait soit $\ulcorner r\sigma\urcorner = Exp(v, \ulcorner M' \cdot M\sigma\urcorner)$, soit $\ulcorner r\sigma\urcorner = v$, et dans ces deux cas s serait un sous terme de $\ulcorner r\sigma\urcorner$. Ainsi, s est nécessairement un sous terme de M'. De plus, comme s n'est pas un sous terme de $\ulcorner r\sigma\urcorner$, on en déduit qu'il n'est pas non plus un sous terme de $\ulcorner M' \cdot M\sigma\urcorner$. Or $s \in STermes(M')$, avec M' normalisé : pour que s soit éliminé par normalisation, il faut nécessairement qu'il soit sous terme de $\ulcorner M\sigma\urcorner$ (Ne pas oublier que $s \neq 1$ par hypothèse). Ainsi, comme $Var(M) \subseteq Var(R_1, \ldots, R_{j-1})$ et comme j est minimal, on peut appliquer le lemme 5.5.1.1. On obtient alors un terme $t_s \in \mathcal{SP}$ tel que $t_s \sqsubseteq_\sigma s$, ce qui prouve le lemme dans ce cas.

2. Si $r = Exp(u, t^z \bullet M')$, (à nouveau) grâce à la définition de protocole bien formé 2.4.4.2, point 2, et par minimalité de j, on a $Var(u) \subseteq Var(R_1, .., R_{j-1})$. On a deux (sous) cas : Si s est un sous terme de $\ulcorner u\sigma\urcorner$, alors le lemme 5.5.1.1 implique qu'il existe $t_s \in \mathcal{SP}$ tel que $t_s \sqsubseteq_\sigma s$, ce qui prouve le lemme. Sinon, s n'est pas un sous terme de $\ulcorner u\sigma\urcorner$. Mais comme il n'est pas non plus un sous terme de $\ulcorner r\sigma\urcorner$, on a nécessairement s sous terme de $\ulcorner (t^z \cdot v_1^{z_1} \cdot .. \cdot v_p^{z_p})\sigma\urcorner$, avec $M' = v_1^{z_1} \cdot .. \cdot v_p^{z_p}$. Ainsi, pour que s puisse être éliminé par normalisation, il existe $k \in \{1, .., p\}$ tel que s soit aussi un sous terme de $\ulcorner v_k\sigma\urcorner$ (On rappelle que $s \neq 1$ par hypothèse). De plus, par définition de protocole bien formé 2.4.4.2, point 2,

et par minimalité de j, on a $Var(v_k) \subseteq Var(R_1, .., R_{j-1})$. On peut alors appliquer le lemme 5.5.1.1, ce qui prouve l'existence d'un terme $t_s \in \mathcal{SP}$ tel que $t_s \sqsubseteq_\sigma s$, et donc le lemme dans ce cas.

Nous avons donc montré le point B de la Figure 5.1, ce qui était le but de cette sous section. □

5.5.2 Propriétés des dérivations.

Dans cette partie, nous allons nous concentrer sur des propriétés "de base" des dérivations. Il s'agit des points C, D, et E de la Figure 5.1, que nous allons montrer dans cet ordre. Le premier lemme va nous permettre de construire des dérivations particulières, où un terme donné n'est jamais décomposé. Ceci nous sera ensuite utile pour appliquer le remplacement d'un terme donné par l'atome 1. En effet, on pourra ainsi ne travailler que sur des dérivations où le terme à remplacer n'est pas décomposé. Si ce n'était pas le cas, les termes obtenus en décomposant ce terme particulier ne serait peut-être plus calculables. (1 ne permet aucun calcul).

Pour toute la suite, nous appellerons $Deriv_t(E)$ une dérivation bien formée partant de E, de but t, et minimale en longueur parmi toutes les dérivations partant de E et de but t. Comme nous travaillons sur des règles d'oracle, nous savons par définition qu'une telle dérivation existe toujours, dès lors que $t \in forge(E)$.

Lemme 5.5.2.1 *Dérivations ne décomposant pas un terme donné.*

Soit E un ensemble de messages (normalisés), et soient t et γ deux messages tels que $\{t, \gamma\} \subseteq forge(E)$. On suppose qu'il existe une dérivation D_γ partant de E, de but γ, et finissant par l'application d'une règle de $L_c \cup L_{oc}$. Alors il existe une dérivation D partant de E, de but t, et telle que $L_d(\gamma) \cap D = \emptyset$.

Preuve. Nous avons en fait déjà prouvé cette propriété dans le chapitre 4 pour les intrus type xor ou préfixe, dans le lemme 4.2.2.1. En examinant la preuve du lemme 4.2.2.1, on se rend compte immédiatement que l'on n'a, à aucun moment, utilisé la structure des termes. Les seules propriétés utilisées étaient celles de règle d'oracle. Or, la définition de règle d'oracle de ce chapitre est un peu plus stricte que dans le chapitre 4 : Cette propriété des dérivations est donc toujours vérifiée, et le lemme est prouvé[13] (La preuve de ce lemme n'utilise pas l'atome 0, remplacé ici par 1). □

La correspondance entre ce lemme et le lemme 4.2.2.1 du chapitre 4 tient au fait que l'on utilise, d'un point de vue général, la même structure de preuve que pour le xor. La plupart des propriétés doivent soit être prouvées différemment avec les opérateurs exponentielle et produit, soit complétées par des lemmes supplémentaires dont on n'avait pas besoin au chapitre 4. Néanmoins, nous avons deux lemmes, n'utilisant que les notions de dérivations et de règles d'oracle, dont les preuves ne changent pas. Il s'agit du lemme 5.5.2.1 précédent, et du lemme suivant. Celui-ci va nous permettre d'appliquer un remplacement sur des termes créés ou reçus par l'intrus.

Lemme 5.5.2.2 *Appliquer un remplacement à une dérivation.*

Soient E et F deux ensembles de messages normalisés, avec $1 \in E \cup F$, soit t un message tel que $t \in forge(E, F)$, et soit s un message non atomique tel que $s \in forge(E)$ et $s \notin STermes(E)$. Soit enfin $\delta = [s \leftarrow 1]$. Alors $\ulcorner t\delta \urcorner \in forge(\ulcorner E\delta, F\delta \urcorner)$.

[13] Alternativement, on peut aussi dire que l'on suit la même preuve que pour le lemme 4.2.2.1.

PREUVE. Nous allons utiliser la preuve du lemme 4.2.2.2, prouvant sensiblement la même propriété au chapitre 4. En effet, les seules différences avec cet autre lemme sont l'utilisation de l'atome 1 au lieu de 0 et la restriction de $\epsilon(s)$ à 1. Ceci ne change pas la preuve, car 0 et $\epsilon(s)$ ne sont utilisés dans la preuve que pour pouvoir appliquer la définition de règle d'oracle. De plus, la preuve du lemme 4.2.2.2 ne dépend que du lemme 4.2.2.1, dont nous venons de voir un équivalent avec le lemme 5.5.2.1. Ainsi, on peut réutiliser ici la preuve du lemme 4.2.2.2, en remplaçant 0 et $\epsilon(s)$ par 1 et en utilisant le lemme 5.5.2.1 à la place du lemme 4.2.2.1, ce qui prouve ce lemme. □

Ainsi, nous avons pu réutiliser deux lemmes du chapitre 4 pour montrer les points C et D de la Figure 5.1. Nous voulons également prouver le point E dans cette sous section, car il s'agir à nouveau d'une propriété sur les dérivations. Il s'agit du lemme suivant. Celui-ci nous montre que tout sous terme d'un message généré par une dérivation est soit obtenu comme sous terme des messages de départ, soit généré par la dérivation et donc connu de l'intrus. Concrètement :

Lemme 5.5.2.3 *Les sous termes d'un but sont connus ou construits.*

Soient E un ensemble de messages (normalisé) et un message (normalisé) t tels que $t \in forge(E)$. De plus, soit $t' \in STermes(t) \backslash STermes(E)$, i.e. un sous terme de t non sous terme de E. Alors il existe une dérivation bien formée partant de E, de but t', et finissant par une règle de composition, i.e. dans $L_c \cup L_{DHc}$.

PREUVE. Soit $D = E \rightarrow_{L_1} E, t_1 ... \rightarrow_{L_n} E, t_1, .., t_n$ une dérivation partant de E et de but $t = t_n$. Comme t' est un sous terme de t_n mais pas de E, il existe nécessairement un $i \in \{1, .., n\}$ minimal tel que t' soit un sous terme de t_i. On a deux cas : Si t' est un sous terme propre de t_i, alors par définition des règles d'intrus normalisé on a nécessairement t' sous terme de $E, t_1, .., t_{i-1}$, ce qui est en contradiction avec la minimalité de i. Ainsi, on a nécessairement $t' = t_i$, et donc $t' \in forge(E)$, et par définition des règles d'oracle, il existe une dérivation bien formée, D', partant de E et de but t. De plus, la dernière règle de cette dérivation ne peut pas être une décomposition L_d ou L_{DHd}, car sinon la définition de dérivation bien formée donnerait t' sous terme de E et donc une contradiction. Ainsi, nous avons une dérivation bien formée D' partant de E, de but t, et finissant par une règle de composition L_c ou L_{DHc}, ce qui prouve le lemme. □

Dans cette preuve, on a utilisé une propriété des sous termes propres des termes générés par des règles d'intrus normalisé. Comme ceci sera utilisé régulièrement, nous en faisons une remarque :

Remarque : Comme l'intrus $\mathcal{L}$ utilisé dans ce chapitre est un intrus normalisé, si $F \rightarrow_L F, t$ alors tous les sous termes propres de t sont sous termes de F.

En effet, on n'a que deux cas : soit L est un règle d'intrus normalisé de composition et la propriété est vraie par définition, soit L est une règle d'intrus normalisé de décomposition, et par définition t est un sous terme de F, et donc ses sous termes propres le sont aussi.

5.5.3 Propriétés des facteurs d'attaques minimales.

Nous sommes à présent en mesure de prouver la propriété centrale de cette section, à savoir une caractérisation des facteurs des substitutions d'attaques minimales. Comme annoncé au début de la section, ceci nous permettra de borner les tailles DAG des substitutions d'attaques minimales. Dans l'énoncé du lemme suivant, il est important de noter que même si les facteurs d'un terme $Exp(u, v_1^{z_1} \cdot .. \cdot v_n^{z_n})$ sont $\{u, v_1, .., v_n\}$, l'emboîtement de plusieurs exponentielles permet d'avoir des facteurs de la forme $Exp(\cdot, \cdot)$. Mais en contre partie, dans l'énoncé suivant, si v_x est de la forme $Exp(\cdot, \cdot)$, alors nécessairement $v_x \neq \sigma(x)$. Formellement, on a :

Proposition 5.5.3.1 *Propriété des facteurs d'attaques minimales.*

Soit (π, σ) une attaque minimale (sur P), soit $x \in Var$ une variable de ce protocole, et soit v_x un facteur de $\sigma(x)$. Si v_x est de la forme $Exp(\cdot, \cdot)$, alors on demande en plus que $v_x \neq \sigma(y)$ pour tout $y \in Var$. Alors il existe $t \in \mathcal{SP}$, i.e. un sous terme du protocole, tel que $t \sqsubseteq_\sigma v_x$.

PREUVE. Soient (π, σ), x, et v_x définis comme ci-dessus. Nous allons effectuer un preuve par contradiction, en construisant une attaque strictement plus petite que (π, σ) si le lemme n'est pas vérifié. Pour cela, on suppose que

$$(*) \ : \quad \text{Pour tout } t, \text{ si } t \sqsubseteq_\sigma v_x \text{ alors } t \notin \mathcal{SP}$$

Comme les atomes appartiennent à la spécification du protocole, i.e. $Atomes \subseteq \mathcal{SP}$, v_x n'est pas un atome. De plus, v_x est standard (i.e. ce n'est pas un produit) par définition des facteurs. Nous pouvons donc appliquer le lemme 5.5.1.2 sur v_x, ce qui nous donne soit une contradiction avec $(*)$, soit il existe j tel que v_x soit un sous terme de $\ulcorner R_j\sigma \urcorner$. Posons :

$$N_x \in I\!N \backslash \{0\} \text{ minimal tel que } v_x \text{ sous terme de } \ulcorner R_{N_x}\sigma \urcorner$$

De plus, supposons qu'il existe $i < N_x$ tel que v_x soit un sous terme de $\ulcorner S_{N_x}\sigma \urcorner$. On peut alors utiliser le lemme 5.5.1.1 sur v_x : puisque ce lemme impliquerait une contradiction avec $(*)$, son hypothèse est nécessairement fausse, i.e. il existe $y \in Var(S_i)$ telle que v_x soit un sous terme de $\sigma(y)$. On peut donc utiliser la définition de protocole bien formé sur y : comme $y \in Var(S_i)$, il existe $i' < i$ tel que $y \in Var(R_{i'})$. Or dans ce cas, on peut appliquer le lemme 5.5.1.2 à nouveau, ce qui donne soit une contradiction avec $(*)$, soit qu'il existe $N'_x \leq i' < N_x$ tel que v_x soit un sous terme de $\ulcorner R_{N'_x}\sigma \urcorner$. Comme v_x n'est pas sous terme des connaissances initiales de l'intrus, $S_0 = \ulcorner S_0\sigma \urcorner$, on a une contradiction avec la minimalité de N_x, et donc v_x n'est pas un sous terme de $\ulcorner S_{N_x}\sigma \urcorner$. En résumé :

$$v_x \in STermes(\ulcorner R_{N_x}\sigma \urcorner) \text{ mais } v_x \notin STermes(\ulcorner S_0\sigma \urcorner, .., \ulcorner S_{N_x-1}\sigma \urcorner)$$

On peut alors appliquer le lemme 5.5.2.3, d'où $v_x \in forge(\ulcorner S_0\sigma \urcorner, .., \ulcorner S_{N_x-1}\sigma \urcorner)$. Soit δ le remplacement $[v_x \leftarrow 1]$. Comme (π, σ) est une attaque, on a pour tout $1 \leq j \leq \#\pi + 1$:

$$\ulcorner R_j\sigma \urcorner \in forge(\ulcorner S_0\sigma \urcorner, .., \ulcorner S_{j-1}\sigma \urcorner)$$

avec $R_{\#\pi+1} = Secret$. Nous allons montrer que $(\pi, \sigma\delta)$ est aussi une attaque. Pour cela, on distingue deux cas :

- Si $j < N_x$, alors par minimalité de N_x, v_x n'est ni un sous terme de $\ulcorner R_j\sigma \urcorner$ ni un sous terme de $\ulcorner S_0\sigma \urcorner, .., \ulcorner S_{j-1}\sigma \urcorner$. On a donc nécessairement

$$\ulcorner\ulcorner R_j\sigma \urcorner\delta \urcorner \in forge(\ulcorner\ulcorner S_0\sigma \urcorner\delta \urcorner, .., \ulcorner\ulcorner S_{j-1}\sigma \urcorner\delta \urcorner)$$

- Si $j \geq N_x$, alors on peut appliquer le lemme 5.5.2.2 avec $t = \ulcorner R_j\sigma \urcorner$, $s = v_x$, $E = \ulcorner S_0\sigma \urcorner, .., \ulcorner S_{N_x-1}\sigma \urcorner$, et $F = \ulcorner S_0\sigma \urcorner, .., \ulcorner S_{j-1}\sigma \urcorner$. On obtient alors

$$\ulcorner\ulcorner R_j\sigma \urcorner\delta \urcorner \in forge(\ulcorner\ulcorner S_0\sigma \urcorner\delta \urcorner, .., \ulcorner\ulcorner S_{j-1}\sigma \urcorner\delta \urcorner)$$

De plus, on peut appliquer le lemme 5.3.2.1 pour chaque j avec $E = \{S_0, .., S_{j-1}\}$ ou $E = \{R_j\}$. On obtient alors

$$\ulcorner R_j\sigma' \urcorner \in forge(\ulcorner S_0\sigma' \urcorner, .., \ulcorner S_{j-1}\sigma' \urcorner)$$

avec $\sigma' = \sigma\delta$. En conséquence, on a bien prouvé que (π, σ') et (π, σ') sont des attaques normalisées sur P. De plus, comme on a remplacé toutes les occurrences d'un terme non atomique v_x par le terme atomique 1, $(\pi, \ulcorner\sigma' \urcorner)$ contredit la minimalité de (π, σ). □

5.6 Borner les tailles DAG des substitutions minimales.

Nous allons à présent utiliser le lemme 5.5.3.1 pour borner les tailles DAG de chaque $\sigma(x)$. Pour cela, nous avons tout d'abord besoin d'introduire une notion de forme normale selon un ensemble F donné.

Soit F un ensemble fini de messages standards et normalisés. On pose $V_F = \{x_s \mid s \in F\}$ où x_s est une nouvelle variable pour chaque $s \in F$. Posons la substitution σ' définie par $\sigma'(x_s) = s$. Pour tout message générique u (i.e. terme générique clos), nous allons définir récursivement sa *forme normale selon* F, notée t_u. Bien que t_u dépende de F, nous écrirons simplement t_u. De plus, nous définissons en même temps $t'_u = x_{\ulcorner u \urcorner}$ si $\ulcorner u \urcorner \in F$, et $t'_u = t_u$ sinon. Nous vérifierons ensuite pas à pas que les quatre propriétés suivantes sont vérifiées :

A) $t_u\sigma' = \ulcorner u \urcorner$ (et donc $t'_u\sigma' = \ulcorner u \urcorner$).

B) $t_u = t_{\ulcorner u \urcorner}$. En particulier, avec A) et B) on obtient immédiatement $t'_u = t'_{\ulcorner u \urcorner}$, $t_{t_u\sigma'} = t_u$, et $t'_{t'_u\sigma'} = t'_u$.

C) $t'_{s\sigma'} = s$ pour tout $s \in STermes^+(t_u)$ tel que $t'_{s\sigma'} = t_{s\sigma'}$.

D) $t_u = Exp(\cdot,\cdot)$ si $\ulcorner u \urcorner = Exp(\cdot,\cdot)$ et u est standard.

On remarque aisément que B) implique C) quand $s = t_u$, et que si $s \in V_F$ alors naturellement $t'_{s\sigma'} = s$. En conséquence, pour montrer C) il nous suffira de ne considérer que le cas $s \in STermes^+(t_u) \setminus (\{t_u\} \cup V_F)$ si l'on a déjà prouvé B). On définit à présent t_u de manière inductive sur la structure de u :

1. Si $\ulcorner u \urcorner \in F$ et $\ulcorner u \urcorner$ est un atome, une paire, ou une encryption $\{..\}^{s \text{ ou } p}_{..}$, alors on pose $t_u = x_{\ulcorner u \urcorner}$.
2. Si $u \in Atomes$, on pose $t_u = u$.
3. Si $u = \langle u_1, u_2 \rangle$ (resp. $u = \{u_1\}^{s \text{ ou } p}_{u_2}$), on pose $t_u = \langle t'_{u_1}, t'_{u_2} \rangle$ (resp. $t_u = \{t'_{u_1}\}^{s \text{ ou } p}_{t'_{u_2}}$).
4. Si $u = u_1^{z_1} \cdot .. \cdot u_n^{z_n}$, avec u_i nécessairement standards, posons $C_1, \ldots, C_k \subseteq \{u_1, \ldots, u_n\}$ une classe d'équivalence sur $\{u_1, \ldots, u_n\}$ telle que $\ulcorner v \urcorner = \ulcorner v' \urcorner$ pour tous $v, v' \in C_i$, $i \in \{1, .., n\}$, et $\ulcorner v \urcorner \neq \ulcorner v' \urcorner$ pour tous $v \in C_i$, $v' \in C_j$, $i \neq j$. On suppose sans perte de généralité que $v\sigma' = 1$ pour tout $v \in C_1$ (Au pire, cette classe sera vide). On remarque que les messages $\ulcorner u_i \urcorner$ sont standards puisque les produits ne peuvent apparaître que dans les exponentielles, ce qui reste vrai lorsque l'on normalise. Soit $s_{C_i} \in C_i$, $i \geq 2$, un ensemble de représentants des classes C_i, et soit $z_{C_i} = \Sigma_{u_i \in C_i} z_i$. Enfin, soit $J = \{i \mid z_{C_i} = 0\}$. On pose alors, avec 1 comme valeur d'un produit vide :
$$t_u = \Pi_{i \notin J \cup \{1\}} (t'_{s_{C_i}})^{z_{C_i}}$$
5. Si $u = Exp(v, M)$, on distingue deux cas :
   (a) Si $\ulcorner v \urcorner \neq Exp(\cdot,\cdot)$, on pose $t_u = t'_v$ si $\ulcorner M \urcorner = 1$, et $t_u = Exp(t'_v, t'_M)$ sinon.
   (b) Si $\ulcorner v \urcorner = Exp(v', M')$ pour un terme v' et un produit M', alors grâce à l'induction et à D), on sait que $t_v = Exp(v'', M'')$ pour un terme v'' et un produit M'' tels que $v''\sigma' = v'$ et $M''\sigma' = M'$ (On remarque que v est un message standard, par définition des messages). Si $\ulcorner M' \bullet M \urcorner = 1$, alors on pose $t_u = v''$. Sinon, on pose $t_u = Exp(v'', t'_{M' \cdot M})$.

Cette définition n'a d'intérêt pour nous que si l'on peut prouver que les quatre propriétés A) à D) sont vérifiées. Nous allons faire cela par induction sur la structure de u, en suivant pas à pas la définition de t_u. Nous donnons donc quatre lemmes, un pour chaque propriété.

Lemme 5.6.0.2 *La propriété A) est vraie.*

PREUVE. Considérons les différents cas la définition de t_u :

1. Si $\ulcorner u \urcorner \in F$ et $\ulcorner u \urcorner \neq Exp(\cdot,\cdot)$, on a $t_u = x_{\ulcorner u \urcorner}$ et donc naturellement $t_u\sigma' = \ulcorner u \urcorner$.
2. Si $u \in Atomes$, on a $t_u = u$ et donc naturellement $t_u\sigma' = \ulcorner u \urcorner$.
3. Si $u = \langle u_1, u_2 \rangle$, alors $t_u = \langle t'_{u_1}, t'_{u_2} \rangle$, et par induction sur u_1 et u_2, on a naturellement $t_u\sigma' = \ulcorner u \urcorner$.
   De même pour $u = \{u_1\}^s_{u_2}$ ou $u = \{u_1\}^p_{u_2}$.
4. Si $u = u_1^{z_1} \cdot .. \cdot u_n^{z_n}$, alors $t_u = \Pi_{i \notin J \cup \{1\}} (t'_{s_{C_i}})^{z_{C_i}}$ selon la définition ci-dessus. Comme les $\ulcorner u_i \urcorner$ sont standards, on se rend compte aisément que $\ulcorner u \urcorner = \Pi_{i \notin J \cup \{1\}} \ulcorner s_{C_i} \urcorner^{z_{C_i}}$. De plus, grâce à l'induction on obtient $\ulcorner s_{C_i} \urcorner = t'_{s_{C_i}}\sigma'$, et donc $t_u\sigma' = \ulcorner u \urcorner$.
5. Si $u = Exp(v, M)$ et $\ulcorner v \urcorner \neq Exp(\cdot,\cdot)$, alors $t_u = t'_v$ si $\ulcorner M \urcorner = 1$ et $t_u = Exp(t'_v, t'_M)$ sinon. On peut appliquer l'induction dans les deux cas suivants :
   (a) Si $\ulcorner M \urcorner = 1$, on obtient $t_u\sigma' = t'_v\sigma' = \ulcorner v \urcorner = \ulcorner u \urcorner$, ce qui prouve le lemme.
   (b) Sinon, i.e. $\ulcorner M \urcorner \neq 1$, obtient $t_u\sigma' = Exp(t'_v\sigma', t'_M\sigma') = Exp(\ulcorner v \urcorner, \ulcorner M \urcorner) = \ulcorner u \urcorner$.
6. Si $u = Exp(v, M)$ et $\ulcorner v \urcorner = Exp(v', M')$, alors on a $t_u = v''$ si $\ulcorner M' \bullet M \urcorner = 1$, et $t_u = Exp(v'', t'_{M' \cdot M})$ sinon, selon la définition ci-dessus.
   (a) Si $\ulcorner M' \bullet M \urcorner = 1$, alors $\ulcorner u \urcorner = v' = v''\sigma' = t_u\sigma'$, ce qui prouve le lemme.
   (b) Si $\ulcorner M' \bullet M \urcorner \neq 1$, grâce à l'induction, on a :
   $$\ulcorner u \urcorner = Exp(v', \ulcorner M' \bullet M \urcorner) = Exp(v''\sigma', t'_{M' \bullet M}\sigma') = t_u\sigma'$$

La propriété A) est donc vrai dans tous les cas de la définition de t_u. □

Lemme 5.6.0.3 *La propriété B) est vraie.*

PREUVE. Considérons les différents cas la définition de t_u :

1. Si $\ulcorner u \urcorner \in F$ et $\ulcorner u \urcorner \neq Exp(\cdot,\cdot)$, on a $t_u = x_{\ulcorner u \urcorner}$ et donc naturellement $t_u = t_{\ulcorner u \urcorner}$.
2. Si $u \in Atomes$, on a $t_u = u$ et donc naturellement $t_u = t_{\ulcorner u \urcorner}$.
3. Si $u = \langle u_1, u_2 \rangle$, alors $t_u = \langle t'_{u_1}, t'_{u_2} \rangle$, et par induction sur u_1 et u_2, on a naturellement $t_u\sigma' = \ulcorner u \urcorner$.
   De même pour $u = \{u_1\}^s_{u_2}$ ou $u = \{u_1\}^p_{u_2}$.
4. Si $u = u_1^{z_1} \cdot .. \cdot u_n^{z_n}$, alors $t_u = \Pi_{i \notin J \cup \{1\}} (t'_{s_{C_i}})^{z_{C_i}}$ selon la définition ce dessus. On a $\ulcorner u \urcorner = \Pi_{i \notin J \cup \{1\}} \ulcorner s_{C_i} \urcorner^{z_{C_i}}$. De plus, les facteurs de $\ulcorner u \urcorner$ sont normalisés, ils appartiennent à des classes d'équivalence différentes, aucun facteur n'est réduit à 1. On obtient donc, grâce à l'induction :
   $$t_{\ulcorner u \urcorner} = \Pi_{i \notin J \cup \{1\}} (t'_{\ulcorner s_{C_i} \urcorner})^{z_{C_i}} = \Pi_{i \notin J \cup \{1\}} (t'_{s_{C_i}})^{z_{C_i}} = t_u.$$
5. Si $u = Exp(v, M)$ et $\ulcorner v \urcorner \neq Exp(\cdot,\cdot)$, alors $t_u = t'_v$ si $\ulcorner M \urcorner = 1$ et $t_u = Exp(t'_v, t'_M)$ sinon. On a deux cas :

(a) Si $\ulcorner M \urcorner = 1$, on remarque que $\ulcorner v \urcorner$ est soit un atome, une paire, ou une encryption. Ainsi, on a $t_u = t'_v$, d'où deux cas : si $t'_v \in V_F$, alors $\ulcorner v \urcorner \in F$, et donc $t_v = t'_v \in V_F$. Sinon, $\ulcorner v \urcorner \notin F$, d'où $t'_v = t_v$. Dans ces deux cas, on obtient grâce à l'induction que $t_u = t'_v = t_v = t_{\ulcorner \urcorner_v} = t_{\ulcorner \urcorner_u}$, ce qui prouve B).

(b) Sinon, i.e. $\ulcorner M \urcorner \neq 1$, on a grâce à l'induction $t_{\ulcorner \urcorner_u} = Exp(t'_{\ulcorner \urcorner_v}, t'_{\ulcorner M \urcorner}) = Exp(t'_v, t'_M) = t_u$, ce qui prouve B).

6. Si $u = Exp(v, M)$ et $\ulcorner v \urcorner = Exp(v', M')$, alors on a $t_u = v''$ si $\ulcorner M' \bullet M \urcorner = 1$, et $t_u = Exp(v'', t'_{M' \cdot M})$ sinon, selon la définition ci-dessus.

(a) Si $\ulcorner M' \bullet M \urcorner = 1$, d'une part on a $t_u = v''$, et d'autre part l'induction nous donne $Exp(v'', M'') = t_v = t_{\ulcorner \urcorner_v} = Exp(t'_{v'}, t'_{M'})$ (On remarque que v' est soit un atome, soit une paire, soit une encryption, puisque $v' \neq Exp(\cdot, \cdot)$). Ainsi, $v'' = t'_{v'}$, et l'on doit prouver que $t'_{v'} = t_{v'}$. Si $t'_{v'} \in V_F$, alors $v' \in F$, et donc $t_{v'} = t'_{v'} \in V_F$. Sinon, $v' \notin F$, d'où $t'_{v'} = t_{v'}$. Dans ces deux cas, on obtient $t_u = v'' = t'_{v'} = t_{v'} = t_{\ulcorner \urcorner_u}$, ce qui prouve B).

(b) Si $\ulcorner M' \bullet M \urcorner \neq 1$, on commence par montrer, comme ci-dessus, que $t'_{v'} = v''$. Ainsi, on a $\ulcorner u \urcorner = Exp(v', \ulcorner M' \bullet M \urcorner)$, et donc $t_{\ulcorner \urcorner_u} = Exp(t'_{v'}, t'_{\ulcorner M' \bullet M \urcorner}) = Exp(v'', t'_{M' \bullet M}) = t_u$, ce qui prouve B).

Ainsi, La propriété B) est donc vrai dans tous les cas de la définition de t_u. □

Lemme 5.6.0.4 *La propriété C) est vraie.*

Preuve. Considérons les différents cas la définition de t_u :

1. Si $\ulcorner u \urcorner \in F$ et $\ulcorner u \urcorner \neq Exp(\cdot, \cdot)$, on a $t_u = x_{\ulcorner \urcorner_u}$ et donc et donc naturellement $t'_{s\sigma'} = s$ pour tout $s \in STermes^+(t_u)$ tel que $t'_{s\sigma'} = t_{s\sigma'}$.
2. Si $u \in Atomes$, on a $t_u = u$ et donc C) est naturellement vérifiée.
3. Si $u = \langle u_1, u_2 \rangle$, alors $t_u = \langle t'_{u_1}, t'_{u_2} \rangle$. Soit $s \in STermes^+(t_u) \setminus (\{t_u\} \cup V_F)$ avec $t'_{s\sigma'} = t_{s\sigma}$. On a donc $s \in STermes^+(t'_{u_i})$. De plus, comme $s \notin V_F$, on obtient $t'_{u_i} \notin V_F$ et donc $t'_{u_i} = t_{u_i}$. Grâce à l'induction et à $s \in STermes^+(t_{u_i})$, on a alors $t'_{s\sigma'} = s$, ce qui prouve C).
4. Si $u = u_1^{z_1} \cdot .. \cdot u_n^{z_n}$, alors $t_u = \Pi_{i \notin J \cup \{1\}} (t'_{s_{C_i}})^{z_{C_i}}$ selon la définition ce dessus. Soit $s \in STermes^+(t_u) \setminus (\{t_u\} \cup V_F)$ avec $t'_{s\sigma'} = t_{s\sigma}$. On a alors $s \in STermes^+(t'_{s_{C_i}})$. Puisque $s \notin V_F$, on a $t'_{s_{C_i}} \notin V_F$, et donc $t'_{s_{C_i}} = t_{s_{C_i}}$. De plus, on sait que $s_{C_i} = u_j$ pour un $j \in \{1, .., n\}$. Ainsi, $s \in STermes^+(t_{u_j})$, d'où $t'_{s\sigma'} = s$ grâce à l'induction, ce qui prouve C).
5. Si $u = Exp(v, M)$ et $\ulcorner v \urcorner \neq Exp(\cdot, \cdot)$, alors $t_u = t'_v$ si $\ulcorner M \urcorner = 1$ et $t_u = Exp(t'_v, t'_M)$ sinon. On a deux cas :

(a) Si $\ulcorner M \urcorner = 1$, posons $s \in STermes^+(t_u) \setminus (\{t_u\} \cup V_F)$ avec $t'_{s\sigma'} = t_{s\sigma}$. On a $t_u = t'_v$, et donc $s \in STermes^+(t'_v)$. De plus, $s \notin V_F$, d'où $t'_v = t_v$. Ainsi, on obtient $s \in STermes^+(t_v)$, et l'induction fournit $t'_{s\sigma'} = s$.

(b) Sinon, $\ulcorner M \urcorner \neq 1$. Soit $s \in STermes^+(t_u) \setminus (\{t_u\} \cup V_F)$ avec $t'_{s\sigma'} = t_{s\sigma}$. On a $t_u = Exp(t'_v, t'_M)$, et donc $s \in STermes^+(t'_v)$ (ou $s \in STermes^+(t'_M)$). De plus, comme $s \notin V_F$, on obtient $t'_v = t_v$ (ou $t'_M = t_M$). Ainsi, $s \in STermes^+(t_v)$ (ou $s \in STermes^+(t_M)$), d'où $t'_{s\sigma'} = s$ grâce à l'induction. Il est important de remarquer ici que M est un sous terme propre étendu de u.

6. Si $u = Exp(v, M)$ et $\ulcorner v \urcorner = Exp(v', M')$, on a $t_u = v''$ si $\ulcorner M' \bullet M \urcorner = 1$, et $t_u = Exp(v'', t'_{M' \cdot M})$ sinon, selon la définition ci-dessus.

   (a) Si $\ulcorner M' \bullet M \urcorner = 1$, alors C) découle simplement du fait que $s \in STermes^+(v'') \subseteq STermes^+(t_v)$.

   (b) Si $\ulcorner M' \bullet M \urcorner \neq 1$, posons $s \in STermes^+(t_u) \backslash (\{t_u\} \cup V_F)$ avec $t'_{s\sigma'} = t_{s\sigma}$. on a $t_u = Exp(v'', t'_{M' \bullet M}) = Exp(v'', t'_{M''\sigma' \bullet M})$. Ainsi, soit $s \in STermes^+(v'')$ soit $s \in STermes^+(t'_{M''\sigma' \cdot M})$. Dans le premier cas, on prouve C) comme ci-dessus. Pour le second cas, on a deux possibilités. Soit $s = t'_{M''\sigma' \cdot M}$, et donc grâce à B) on a $t'_{s\sigma'} = s$, ce qui prouve C). Soit $s \neq t'_{M''\sigma' \cdot M}$, et comme $s \notin V_F$, on a $t'_{M''\sigma' \cdot M} \notin V_F$, d'où $t'_{M''\sigma' \bullet M} = t_{M''\sigma' \bullet M}$. Notons $M = t_1^{z_1} \cdots t_n^{z_n}$ et $M'' = {t''}_1^{z''_1} \cdots {t''}_{n''}^{z''_{n''}}$. On sait déjà que $t_{M''\sigma' \bullet M}$ est de la forme $\Pi_{i \notin J \cup \{1\}} (t'_{s_i})^{z_{C_i}}$ avec soit $s_i = t''_j \sigma'$ soit $s_i = t_j$ pour un certain j. Ainsi, $s \in STermes^+(t'_{s_i})$. Considérons les deux ces suivants, selon la nature de s_i :
   - Si $s_i = t_j$, alors $s \in STermes^+(t'_{t_j}) = STermes^+(t_{t_j})$ puisque $s \notin V_F$. Grâce à l'induction et comme t_j est un sous terme propre de u, on obtient $t'_{s\sigma'} = s$.
   - Si $s_i = t''_j \sigma'$, alors $s \in STermes^+(t'_{t''_j \sigma'})$. De plus, $s \notin V_F$, et l'on sait donc que $t'_{t''_j \sigma'} \notin V_F$. En particulier, $s \in STermes^+(t_{t''_j \sigma'})$. Or on sait déjà que $t''_j \in STermes^+(t_v)$, et comme $t'_{t''_j \sigma'} \notin V_F$, on a également $t'_{t''_j \sigma'} = t_{t''_j \sigma'}$. Ainsi, grâce à l'induction on obtient $t_{t''_j \sigma'} = t''_j$, et donc $s \in STermes^+(t''_j) \subseteq STermes^+(t_v)$, ainsi que $t'_{s\sigma'} = s$ puisque v est un sous terme propre de u.

   On a donc bien la propriété C) dans tous les cas.

La propriété C) est donc vrai dans tous les cas de la définition de t_u. □

Lemme 5.6.0.5 *La propriété D) est vraie.*

Preuve. Il suffit d'examiner les différents cas de la définition de t_u pour voir que la propriété D) est naturellement vérifiée dans tous les cas. En particulier, il suffit de voir que u est non standard pour le cas $u = u_1^{z_1} \cdot .. \cdot u_n^{z_n}$, et que $\ulcorner u \urcorner = v' \neq Exp(\cdot, \cdot)$ pour le cas $u = Exp(v, M)$ avec $\ulcorner v \urcorner = Exp(v', M')$ et $\ulcorner M' \bullet M \urcorner = 1$. □

En résumé, on vient de prouver le lemme suivant :

Lemme 5.6.0.6 *Propriétés de t_u et t'_u.*

Pour tout ensemble (fini) F de messages standards et normalisés, et pour tout message u générique, on a :

1. $t_u \sigma' = \ulcorner u \urcorner$ *et* $t'_u \sigma' = \ulcorner u \urcorner$.
2. $t_u = t_{\ulcorner u \urcorner}$, $t'_u = t'_{\ulcorner u \urcorner}$, $t_{t_u \sigma'} = t_u$, *et* $t'_{t'_u \sigma'} = t'_u$.
3. $t'_{s\sigma'} = s$ *pour tout* $s \in STermes^+(t_u)$ *tel que* $t'_{s\sigma'} = t_{s\sigma'}$.
4. *Si* $\ulcorner u \urcorner = Exp(\cdot, \cdot)$ *et si* u *est standard, alors* $t_u = Exp(\cdot, \cdot)$.

avec t_v la forme normale de v selon F, et $t'_u = t_{\ulcorner v \urcorner}$ pour tout message générique v.

Dans le but de borner les tailles DAG des substitutions d'attaques minimales, nous avons également besoin du lemme suivant :

Lemme 5.6.0.7 *Sous termes de formes normales.*

Soient u un terme générique normalisé, et σ une substitution normalisée. Posons $\mathcal{F} = \bigcup_{x\in Var(u)} Facteur(\sigma(x))$ l'ensemble des facteurs de σ, $V_{\mathcal{F}} = \{x_s \,|\, s \in \mathcal{F}\}$ un ensemble de nouvelles variables, et $\sigma'(x_s) = s$ la substitution correspondante. De plus, notons t_s la forme normale de s selon $\mathcal{F}$. On a alors :

$$\begin{array}{rl} & STermes(t_{u\sigma}) \subseteq \{\, t_{s\sigma} \,|\, s \in STermes^+(u)\,\} \cup V_{\mathcal{F}} \cup \{1\} \\ \textit{d'où} & STermes(t'_{u\sigma}) \subseteq \{\, t_{s\sigma} \,|\, s \in STermes^+(u)\,\} \cup V_{\mathcal{F}} \cup \{1\} \end{array}$$

Preuve. Nous allons (à nouveau) procéder par induction sur la structure de u.

Si $\ulcorner u\sigma \urcorner \in \mathcal{F}$ et $\ulcorner u\sigma \urcorner$ est un atome, une paire, ou une encryption, alors $t_{u\sigma} \in V_{\mathcal{F}}$ ce qui prouve les inclusions précédentes immédiatement. Sinon, i.e. si $\ulcorner u\sigma \urcorner \notin \mathcal{F}$, on distingue plusieurs cas :

Supposons tout d'abord que u soit une variable. Si $u\sigma$ est un atome, une paire, ou une encryption, alors $u\sigma \in \mathcal{F}$, et donc $t_{u\sigma} \in V_{\mathcal{F}}$. Si $u\sigma = Exp(v, t_1^{z_1} \cdot .. \cdot t_n^{z_n})$, alors $v, t_1, \ldots, t_n \in \mathcal{F}$, et comme $u\sigma$ est normalisé, la définition de $t_{u\sigma}$ nous donne $t_{u\sigma} = Exp(x_v, x_{t_1}^{z_1} \cdot .. \cdot x_{t_n}^{z_n})$. Dans les deux cas, on obtient que $STermes(t_{u\sigma}) \subseteq \{t_{s\sigma} \mid s \in STermes^+(u)\} \cup V_{\mathcal{F}} \cup \{1\}$ce qui prouve le lemme dans ce cas. On suppose donc à partir d'ici que u n'est pas une variable.

Si $u\sigma \in Atomes$ (i.e. u est un atome), alors les inclusions sont naturellement vraies.

Si $u = \langle u_1, u_2\rangle$, alors $t_{u\sigma} = \langle t_{u_1\sigma}, t_{u_2\sigma}\rangle$. Ainsi, par induction et comme $STermes^+(u_1, u_2) \subseteq STermes^+(u)$, on obtient

$$\begin{array}{rl} STermes(t_{u\sigma}) & \subseteq \{t_{u\sigma}\} \cup STermes(t_{u_1\sigma}) \cup STermes(t_{u_2\sigma}) \\ & \subseteq \{\, t_{s\sigma} \mid s \in STermes^+(u)\,\} \cup V_{\mathcal{F}} \cup \{1\} \end{array}$$

Si $u = u_1^{z_1} \cdot .. \cdot u_n^{z_n}$ et $\ulcorner u\sigma \urcorner \neq 1$, avec les notations de la définition de forme normale selon $\mathcal{F}$, on a $t_{u\sigma} = \Pi_{i\notin J\cup\{1\}}(t'_{s_{C_i}\sigma})^{z_{C_i}}$. En particulier, on remarque que pour chaque i, il existe j tel que $s_{C_i} = u_j$. Ainsi, par induction et comme $STermes^+(u_i) \subseteq STermes^+(u)$, on obtient

$$\begin{array}{rl} STermes(t_{u\sigma}) & \subseteq \{t_{u\sigma}\} \cup \bigcup_{i=1..n} STermes(t'_{u_i\sigma}) \\ & \subseteq \{\, t_{s\sigma} \mid s \in STermes^+(u)\,\} \cup V_{\mathcal{F}} \cup \{1\} \end{array}$$

Enfin, si $u = Exp(v, M)$ on distingue plusieurs cas :

- $\ulcorner v\sigma \urcorner \neq Exp(\cdot,\cdot)$: On a soit $t_{u\sigma} = t'_v$, soit $t_{u\sigma} = Exp(t'_{v\sigma}, t'_{M\sigma})$, et donc $STermes(t_{u\sigma}) \subseteq \{t_{u\sigma}\} \cup STermes(t'_{v\sigma}) \cup STermes(t'_{M\sigma})$. Par induction et comme tous les sous termes étendus de v et M sont des sous termes étendus de u ($STermes^+(v, M) \subseteq STermes^+(u)$), on obtient :

$$STermes(t_{u\sigma}) \subseteq \{t_{s\sigma} \mid s \in STermes^+(u)\} \cup V_{\mathcal{F}} \cup \{1\}$$

- $\ulcorner v\sigma \urcorner = Exp(v', M')$: Grâce au lemme 5.6.0.6, point 4, on sait que $t_{v\sigma} = Exp(v'', M'')$ avec $v''\sigma' = v'$ et $M''\sigma' = M'$. Si $t_u = v''$, alors par induction et comme tous les sous termes étendus de v ($STermes^+(v)$) sont aussi des sous termes étendus de u, on obtient $STermes(t_{u\sigma}) \subseteq \{t_{s\sigma} \mid s \in STermes^+(u)\} \cup V_{\mathcal{F}} \cup \{1\}$. Sinon, on a $t_u = Exp(v'', t'_{M'\bullet M\sigma}) = Exp(v'', t'_{M''\sigma'\bullet M\sigma})$. Si $t'_{M''\sigma'\bullet M\sigma} \in V_{\mathcal{F}}$, alors par induction et comme $STermes^+(v) \subseteq STermes^+(u)$, on obtient $STermes(t_{u\sigma}) \subseteq \{t_{s\sigma} \mid s \in STermes^+(u)\} \cup V_{\mathcal{F}} \cup \{1\}$. Sinon, $t'_{M''\sigma'\cdot M\sigma} \notin V_{\mathcal{F}}$, et donc $t'_{M''\sigma'\bullet M\sigma} = t_{M''\sigma'\bullet M\sigma}$. Notons $M = t_1^{z_1} \cdots t_n^{z_n}$ et $M'' = {t''}_1^{z''_1} \cdots {t''}_{n''}^{z''_{n''}}$. On sait que $t_{M''\sigma'\bullet M\sigma}$

est de la forme $\Pi_{i\notin J\cup\{1\}}(t'_{s_i})^{z_{C_i}}$ où soit $s_i = t_j\sigma$, soit $s_j = t''_j\sigma'$ pour un j quelconque (On rappelle que par convention, si le produit est vide alors sa valeur est 1). Ainsi,

$$\begin{aligned} STermes(t_{u\sigma}) \subseteq\ & \{t_{u\sigma}\} \cup STermes(t_{v\sigma}) \cup STermes(\{t'_{t_j\sigma} \mid j = 1,\dots,n\}) \\ & \cup STermes(\{t'_{t''_j\sigma'} \mid j = 1,\dots,n''\}) \end{aligned}$$

Grâce au lemme 5.6.0.6, point 3, on sait que $t'_{t''_j\sigma'} = t''_j$ si $t'_{t''_j\sigma'} \notin V_{\mathcal{F}}$, et donc $t'_{t''_j\sigma'} = t_{t''_j\sigma'}$ puisque $t''_j \in STermes(t_{v\sigma})$. On a donc :

$$STermes(\{t'_{t''_j\sigma'} \mid j = 1,\dots,n''\}) \subseteq STermes(t_{v\sigma}) \cup V_{\mathcal{F}}$$

Par induction, on obtient que $STermes(t_{v\sigma}) \subseteq \{t_{s\sigma} \mid s \in STermes^+(u)\} \cup V_{\mathcal{F}} \cup \{1\}$ puisque v est un sous terme propre de $STermes^+(u)$. De plus, $STermes(t'_{t_j\sigma}) \subseteq \{t_{s\sigma} \mid s \in STermes^+(u)\} \cup V_{\mathcal{F}} \cup \{1\}$ puisque t_j est un sous terme propre de u. Au final, on obtient

$$STermes(t_{u\sigma}) \subseteq \{t_{s\sigma} \mid s \in STermes^+(u)\} \cup V_{\mathcal{F}} \cup \{1\}$$

On a donc prouvé les inclusions dans tous les cas, et le lemme est donc démontré. □

Nous avons maintenant tous les outils nécessaires pour borner les tailles DAG des substitutions d'attaques minimales, ce qui est le but de cette section.

Théorème 5.6.0.8 *Les attaques minimales sont bornées en taille DAG.*

Pour toute attaque minimale (π,σ), on a $|\{\sigma(x) \mid x \in Var\}|_{dag} \leq 8.|P|_{dag} + 1$, où $|P|_{dag} = |\mathcal{SP}|_{dag}$.

Preuve. Posons $|P|^+_{dag} = \#STermes^+(\{R_i, S_i \mid i \in \{1,..,\#\mathcal{I}\}\})$. Grâce à la définition de sous termes étendus ($STermes^+(..)$), on a $|P|^+_{dag} \leq 2.|P|_{dag}$. On va donc borner la taille de σ par $4.|P|^+_{dag} + 1$. Pour commencer, nous avons besoin de quelques notations. Étant donnés un ensemble E de termes génériques et un terme générique t, on pose $E_{<t} = \{t' \in E \mid t' \text{ sous terme de } t\}$ l'ensemble de tous les termes de E sous termes de t.

Soit (π,σ) une attaque minimale (de P). On pose F' l'ensemble des termes génériques s de la forme $Exp(\cdot,\cdot)$ tels qu'il existe deux variables $x, y \in Var$ vérifiant $s \in Facteur(\sigma(x))$ et $s = \sigma(y)$. De plus, on associe à chaque $s \in F'$ une variable y telle que $s = \sigma(y)$. On note $V_{F'} \subseteq Var$ l'ensemble des variables de Var associées à un message de F'. Posons :

$$F = \{s \mid x \in Var,\ s \in Facteur(\sigma(x))\} \backslash F'$$

Pour tout $s \in F$, on introduit une nouvelle variable x_s (différente pour chaque élément de F). On note Var_F l'ensemble de ces variables. Grâce au lemme 5.5.3.1, on sait que pour tout $s \in F$, il existe $\hat{s} \in \mathcal{SP}$ tel que $\hat{s} \sqsubseteq_\sigma s$. On en déduit immédiatement que :

$$\#(F) \leq |P|$$

Posons $\mathcal{F} = F \cup F'$ l'ensemble de tous les facteurs de σ. Pour $s \in F$, on définit $\sigma'(x_s) = s$. Et pour $s \in F'$, on définit $\sigma'(x) = s$ avec x la variable associée à s. De plus, pour tout message générique m (produit clos ou message), on note t_m la forme normale de m selon $\mathcal{F}$. Enfin, on définit :

$$G = \{t_{u\sigma} \mid u \in STermes(P)\}$$

Grâce au lemme 5.6.0.6, on sait que :

$$t_{u\sigma}\sigma' = \ulcorner u\sigma \urcorner \qquad (1)$$

Nous voulons borner polynomialement la taille DAG de G par la taille DAG du protocole. Pour cela, il suffit d'appliquer le lemme 5.6.0.7. En effet, on obtient $STermes(G) \subseteq \{t_{s\sigma} \mid s \in STermes^+(P)\} \cup V_F \cup V_{F'} \cup \{1\}$, et donc :

$$|G| = \#(STermes(G)) \leq 3.|P|^+_{dag} + 1$$

Posons à présent $S = \{\sigma(x) \mid x \in Var\}$ et $S' = \{\sigma'(x) \mid x \in V_F \cup V_{F'}\}$. De manière à borner le nombre de sous termes de S, nous commençons par examiner les sous termes de F. Grâce au lemme 5.5.3.1 et à (1), on a :

$$F \subseteq \{t_{u\sigma}\sigma' \mid u \in STermes(P)\} \subseteq \{t\sigma' \mid t \in STermes(G)\}$$

Ainsi, pour tout $s \in F$:

$$\begin{array}{rcl} STermes(s) & \subseteq & \{t\sigma' \mid t \in STermes(G)\} \cup STermes(S'_{<s}) \\ & \subseteq & \{t\sigma' \mid t \in STermes(G)\} \cup STermes((F \cup F')_{<s}) \\ & \subseteq & \{t\sigma' \mid t \in STermes(G)\} \cup STermes(F_{<s}) \cup STermes(F'_{<s}) \end{array}$$

On en déduit que, pour tout $x \in Var$:

$$\begin{array}{rcll} STermes(F_{<\sigma(x)}) & = & \bigcup_{s \in F_{<\sigma(x)}} STermes(s) & \\ & \subseteq & \{t\sigma' \mid t \in STermes(G)\} \cup STermes(F'_{<\sigma(x)}) & \\ & \subseteq & \{t\sigma' \mid t \in STermes(G)\} \cup STermes(S_{<\sigma(x)}) & (2) \end{array}$$

Et donc :

$$\begin{array}{rcll} STermes(\sigma(x)) & = & \{\sigma(x)\} \cup STermes(Facteur(\sigma(x))) & \\ & = & \{\sigma(x)\} \cup STermes(Facteur(\sigma(x)) \cap (F \cup F')) & \\ & \subseteq & \{\sigma(x)\} \cup STermes(F_{<\sigma(x)}) \cup STermes(S_{<\sigma(x)}) & \\ & \subseteq & \{\sigma(x)\} \cup \{t\sigma' \mid t \in STermes(G)\} \cup STermes(S_{<\sigma(x)}) & \text{grâce à (2)} \end{array}$$

On peut alors caractériser S :

$$\begin{array}{rcl} STermes(S) & = & \bigcup_{x \in Var} STermes(\sigma(x)) \\ & \subseteq & S \cup \{t\sigma' \mid t \in STermes(G)\}. \end{array}$$

Ainsi, $|S|_{dag} = \#(STermes(S)) \leq \#(S) + |G|_{dag} \leq \#Var + 3.|P|^+_{dag} + 1 \leq 4.|P|^+_{dag} + 1$ ce qui prouve le théorème. □

On en déduit immédiatement :

Corollaire 5.6.0.9 *Borner la taille DAG des messages transmis dans une attaque minimale.*
Soit (π, σ) une attaque minimale sur P. Pour tout $i \in \{1, .., \#\pi\}$, on a

$$\begin{array}{rcl} & |R_i\sigma, S_0\sigma, .., S_{i-1}\sigma|_{dag} & \leq 10.|P|_{dag} + 1 \\ et & |Secret, S_0\sigma, .., S_{i-1}\sigma|_{dag} & \leq 10.|P|_{dag} + 1 \end{array}$$

Nous avons donc réussi à borner les tailles DAG des substitutions minimales. Cependant, à la différence de l'intrus xor, cette borne seule ne suffit plus : Il nous faut aussi borner les coefficients de ces substitutions. C'est ce que nous allons faire dans la section suivante.

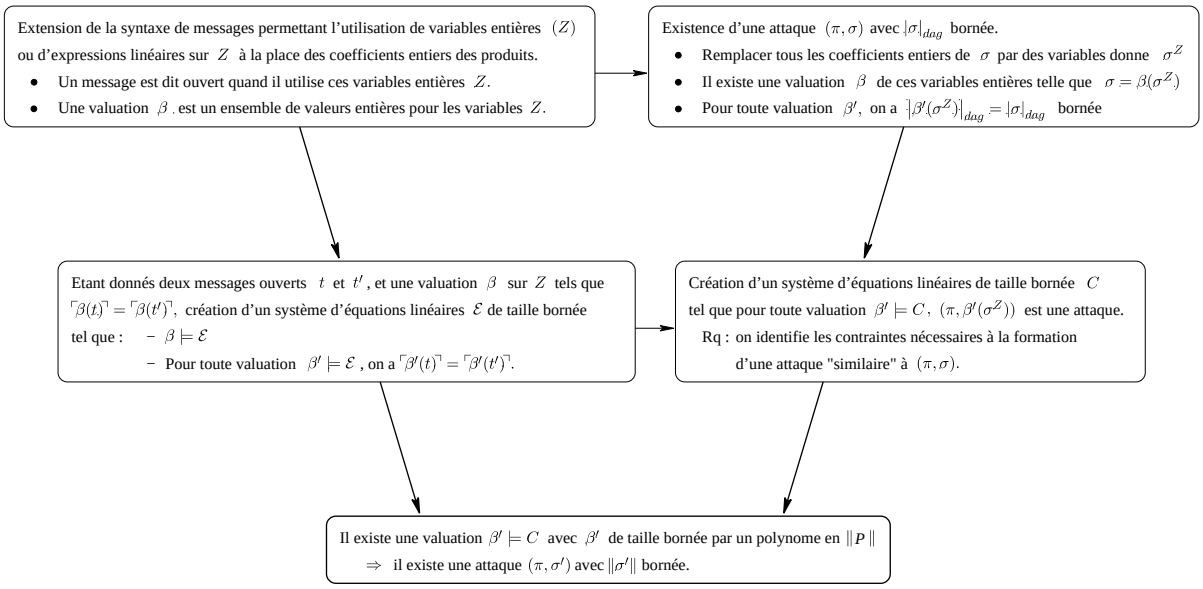


FIG. 5.2: Idée générale de la section 5.7

5.7 Existence d'attaques à coefficients bornés

La notion d'attaque minimale que nous avons utilisée jusqu'ici nous a été très utile pour borner les tailles DAG des substitutions à considérer lors d'une recherche d'attaque. Cependant, cette notion ne tient absolument pas compte des coefficients entiers présents dans les produits de ces substitutions. Le but de cette section est de prouver l'existence d'attaques dont les coefficients sont bornés polynomialement en la taille du protocole (et dont la taille DAG est également bornée), dès lors que le protocole admet au moins une attaque minimale. Si c'est le cas, nous dirons que le protocole P admet une (ou plusieurs) attaque à coefficients bornés. Il s'agir bien sûr ici des coefficients des produits présents dans la substitution définissant cette attaque. Pour prouver cela, nous allons associer à toute attaque minimale (π, σ) une substitution σ^Z et un système d'équations linéaires tels que σ et σ^Z coïncident sauf pour les exposants entiers de leurs produits, et tels que pour toute instance σ' de σ^Z induite (ou créée à partir) par une solution du système d'équations, (π, σ') est une attaque. En bornant polynomialement la taille d'un tel système d'équations en fonction de la taille du protocole, on obtiendra une borne polynomiale sur ses solutions minimales (c.f. [13]), i.e. une borne polynomiale sur les coefficients entiers de certaines "variantes" d'attaques minimales.

L'idée générale de cette section est présentée à la figure 5.2. Les parties les plus difficiles seront (évidement) les preuves des deux propriétés de la seconde ligne. En particulier, l'existence (et la création) du système d'équations linéaires $\mathcal{E}$ de la propriété de gauche, ligne 2, correspondant à la proposition 5.7.4.3, est le noyau de cette section. Nous allons tout d'abord présenter les différentes notions particulières à cette section dont nous aurons besoin. En particulier, nous définirons des tuples nous permettant de représenter symboliquement un ensemble de solutions alternatives à une solution donnée pour le problème de l'insécurité (i.e. attaque avec d'autres coefficients entiers). Nous pourrons alors montrer que les tuples existent et sont de taille bornée, ce qui nous permettra de trouver des solutions à coefficients bornés au problème de l'insécurité de protocoles face à l'intrus DH.

5.7.1 Définitions de systèmes d'équations et de tuples.

Dans cette section, nous allons essentiellement travailler sur les coefficients entiers des produits présents dans une attaque (sa substitution, en fait) ou dans un message transmis. En particulier, nous aurons besoin de placer des variables entières comme coefficients d'un produit (dans un message). Pour cela, nous donnons la définition suivante de messages (ou produits) ouverts, i.e. des messages (ou produits) dont les coefficients entiers peuvent être des variables entières. De plus, nous aurons besoin de gérer des contraintes sur ces variables entières, ce que nous ferons grâce à des expressions linéaires.

Définition 5.7.1.1 *Messages et produits ouverts, expressions linéaires.*

Soit Z un ensemble de variables. On définit l'ensemble $\ln(Z)$ *des messages ouverts sur Z, noté simplement $\mathcal{M}$, l'ensemble $\mathcal{P}(Z)$ des produits ouverts sur Z, noté simplement $\mathcal{P}$, et l'ensemble $\mathcal{L}_{exp}(Z)$ des expressions linéaires sur Z, noté simplement $\mathcal{L}_{exp}$, de la manière suivante :*

$$\begin{array}{rcl} \mathcal{M} & ::= & Atomes \mid \langle \mathcal{M}, \mathcal{M} \rangle \mid \{\mathcal{M}\}^s_{\mathcal{M}} \mid \{\mathcal{M}\}^p_{\mathcal{M}} \mid Exp(\mathcal{M}, \mathcal{P}) \\ \mathcal{P} & ::= & \mathcal{M}^{\mathcal{L}_{exp}} \mid \mathcal{M}^{\mathcal{L}_{exp}} \cdot \mathcal{P} \\ \mathcal{L}_{exp} & ::= & \boldsymbol{Z} \mid Z \mid \mathcal{L}_{exp} + \mathcal{L}_{exp} \mid \boldsymbol{Z}.\mathcal{L}_{exp} \end{array}$$

De plus, on appelle message générique ouvert un message ou produit ouvert.

Comme pour les produits du chapitre 2, on considère les produits ouverts modulo l'associativité et la commutativité de l'opérateur $\cdot$, i.e. $AC.$. De la même manière, on considérera toujours les expressions linéaires modulo l'associativité et la commutativité de l'opérateur $+$, i.e. AC_+. En particulier, on dira qu'une expression linéaire e appartient à un ensemble S quand il existe un élément de s équivalent à e modulo AC_+. De la même manière, on définit l'inclusion d'ensembles d'expressions linéaires.

On étend de manière naturelle les notions de sous terme et de sous terme étendu, $STermes(t)$ et $STermes^+(t)$, à tout message ou produit ouvert. On en déduit les notions correspondantes de taille DAG et taille DAG étendue, $|t|_{dag}$ et $|t|^+_{dag}$, i.e. le nombre de sous termes ou de sous termes étendus de t. On rappelle que les sous termes étendus d'un message (ouvert) t est $STermes(t) \cup \{M \mid Exp(u, M) \in STermes(t)\}$. De plus, on définit naturellement la taille des exposants de t, notée $|t|_{exp}$, comme étant :

$$\text{Pour } t \text{ message générique ouvert,} \quad |t|_{exp} = \sum_{t_1^{e_1} \cdot \ldots \cdot t_n^{e_n} \in STermes(t)} \|e_1\| + .. + \|e_n\|$$

avec $\|e\|$ le nombre de caractères nécessaires pour représenter une expression linéaire e quand les entiers sont codés en binaire. La taille d'un message générique ouvert est alors $\|t\| = |t|_{dag} + |t|_{exp}$, et sa taille étendue $\|t\|^+ = |t|^+_{dag} + |t|_{exp}$. Toutes ces notions de taille sont naturellement étendues aux ensembles de messages génériques. De même que pour les messages ou produits habituels, on a :

Remarque : Pour tout message ou produit ouvert t, on a $|t|^+_{dag} \leq 2.|t|_{dag}$ et donc $\|t\|^+ \leq 2.\|t\|$.

A la manière des substitutions sur les termes classiques du chapitre 2, on a besoin d'instancier les variables entières présentes dans message ou un produit ouvert (intuitivement, pour le "refermer") et ainsi obtenir un message ou un produit classique. Pour cela, on appellera *valuation* une fonction $\beta : Z \to \boldsymbol{Z}$. L'application d'une valuation β à une expression linéaire e donne,

par calcul, un entier $\beta(e) \in \boldsymbol{Z}$. Ainsi, l'application d'une valuation à un message ou un produit, définit récursivement de manière naturelle, donne un message ou un produit (classique). Dans toute cette section, β désignera toujours une valuation de Z sur $\boldsymbol{Z}$.

Nous avons décrit un moyen de paramétrer les exposants des produits. Mais pour assurer certaines propriétés des messages (comme par exemple la description d'une attaque), nous avons besoin de décrire des ensembles de contraintes sur les variables entières. Pour cela, on définit un *système d'équations linéaires* $\mathcal{E}$ comme étant un ensemble fini d'équations de la forme $e = e'$, avec e et e' deux expressions linéaires (sur Z). La taille d'un tel système, notée $\|\mathcal{E}\|$, est la somme des tailles des expressions linéaires le composant :

$$\|\mathcal{E}\| = \sum_{e=e' \in \mathcal{E}} \|e\| + \|e'\|$$

De plus, on peut appliquer une valuation à un système d'équations linéaires. Ainsi, on dira qu'une valuation β satisfait un système d'équations linéaires $\mathcal{E}$, noté $\beta \models \mathcal{E}$, quand $\beta(e) = \beta(e')$ pour tout $e = e' \in \mathcal{E}$. Pour tout message ou produit t, on note $\mathcal{L}_{exp}(t)$ l'ensemble des expressions linéaires présentes dans t. De la même manière, pour tout système d'équations linéaires $\mathcal{E}$, on note $\mathcal{L}_{exp}(\mathcal{E}) = \{e \,|\, e = e' \in \mathcal{E} \text{ ou } e' = e \in \mathcal{E}\}$ l'ensemble des expressions linéaires présentes dans $\mathcal{E}$.

Pour faciliter l'utilisation de ces systèmes d'équations linéaires, on définit $R_{\mathcal{E}} = \{\, (e, e') \,|\, e = e' \in \mathcal{E} \,\} \subseteq \mathcal{L}_{exp} \times \mathcal{L}_{exp}$ la relation déduite de $\mathcal{E}$. De plus, on note $R^*_{\mathcal{E}}$ la clôture réflexive et transitive de $R_{\mathcal{E}}$ (Attention, on considère les expressions linéaires modulo AC_+ i.e. l'associativité et la commutativité de l'opérateur $+$). On obtient ainsi une relation d'équivalence entre deux systèmes d'équations linéaires $\mathcal{E}$ et $\mathcal{E}'$, quand $R^*_{\mathcal{E}} = R^*_{\mathcal{E}'}$. Dans toute la suite, nous considérerons toujours les systèmes d'équations linéaires selon $R^*_{\mathcal{E}}$, i.e. modulo la réflexivité et la transitivité de l'égalité. On pose donc $\mathcal{E} = \mathcal{E}'$ ssi $R^*_{\mathcal{E}} = R^*_{\mathcal{E}'}$, et $\mathcal{E} \subseteq \mathcal{E}'$ ssi $R^*_{\mathcal{E}} \subseteq R^*_{\mathcal{E}'}$.

Nous pouvons à présent poser quelques définitions importantes pour la suite sur les notations précédentes. Tout d'abord, on a besoin d'une notion d'équivalence sur les messages ou produits ouverts :

Définition 5.7.1.2 *Égalité et équivalence modulo une valuation.*

Soient $\mathcal{E}$ un système d'équations linéaires, β une valuation et t, t' deux messages ou produits ouverts. On dit que :

- *t et t' sont égaux* selon β, noté $t =_\beta t'$, *ssi* $\beta(t) = \beta(t')$.[14]
- *t et t' sont dits équivalents selon β*, noté $t \approx_\beta t'$, *ssi* $\lceil \beta(t) \rceil = \lceil \beta(t') \rceil$.
- *t et t' sont dits égaux selon $\mathcal{E}$, noté $t =_{\mathcal{E}} t'$, ssi pour tout $\beta \models \mathcal{E}$, on a $t =_\beta t'$.*
- *t et t' sont dits équivalents selon $\mathcal{E}$, noté $t \approx_{\mathcal{E}} t'$, ssi pour tout $\beta \models \mathcal{E}$, on a $t \approx_\beta t'$.*

De plus, on peut caractériser tous les systèmes d'équations linéaires validant une valuation et une égalité de messages génériques :

Définition 5.7.1.3 *Système d'équations linéaires basé sur une valuation et une égalité.*

Pour toute valuation β et tous messages ou produits ouverts t et t', on dit qu'un système *(d'équations linéaires) $\mathcal{E}$* est basé sur $t =_\beta t'$ *quand :*

- $\beta \models \mathcal{E}$
- *Pour toute valuation β' telle que $\beta' \models \mathcal{E}$, on a $t =_{\beta'} t'$.*

[14] On rappelle que l'égalité est modulo l'associativité et la commutativité de $\cdot$. Par exemple, $a^2 \cdot b^3 \cdot c^{-2} = c^{-2} \cdot a^2 \cdot b^3$.

Nous allons commencer par montrer qu'il existe de "petits" systèmes (d'équations linéaires) basés sur une relation $t =_\beta t'$ donnée. Pour cela :

Lemme 5.7.1.4 *Existence de systèmes d'équations linéaires bornés.*

Soit β une valuation, et soient t et t' deux messages ou produits ouverts tels que $t =_\beta t'$. Alors il existe un système d'équations linéaires $\mathcal{E}^{=_\beta}_{t,t'}$ basé sur $t =_\beta t'$ de taille bornée par $2.(|t,t'|^+_{dag})^3$.

Preuve. Posons $R \subseteq STermes^+(t,t') \times STermes^+(t,t')$ tel que $s =_\beta s'$ pour tout $(s,s') \in R$. De cette manière, R est la plus petite relation sur $STermes^+(t,t')$ telle que :

- $(t,t') \in R$,
- si $(s,s') \in R$, et si $s = \langle t_1, t_2 \rangle$, alors nécessairement $s' = \langle t'_1, t'_2 \rangle$ avec t'_1 et t'_2 deux messages ouverts, puisque par construction $s =_\beta s'$. On obtient donc $\langle t_1, t'_1 \rangle \in R$ et $\langle t_2, t'_2 \rangle \in R$. De même pour les deux encryptions, symétrique et asymétrique.
- si $(s,s') \in R$, et si s est un produit $t_1^{e_1} \cdots t_n^{e_n}$, avec $n \geq 1$, alors s' est un produit de la forme $t'^{e'_1}_1 \cdots t'^{e'_n}_n$, avec $n \geq 1$. Alors comme $s =_\beta s'$, pour tout i il existe au moins un j tel que $t_i =_\beta t'_j$, et donc tel que $(t_i, t'_j) \in R$.
- si $t = Exp(u, M)$ avec u un message ouvert et M un produit ouvert, alors $t' = Exp(u', M')$ avec u' un message ouvert et M' un produit ouvert. On a donc $(u,u') \in R$ et $(M,M') \in R$.

On définit alors, pour tout $(s,s') \in R$, un système d'équations linéaires $\mathcal{E}_{(s,s')}$ comme suit : Si s n'est pas un produit (et donc s' non plus), alors $\mathcal{E}_{(s,s')} = \emptyset$. Sinon, s étant de la forme $t_1^{e_1} \cdots t_n^{e_n}$ avec $n \geq 1$, et s' étant de la forme $t'^{e'_1}_1 \cdots t'^{e'_n}_n$, on pose $\mathcal{E}_{(s,s')} = \{e_i = e'_j \mid (t_i, t'_j) \in R\}$. On voit alors aisément, par induction structurelle, que $\mathcal{E}_R = \bigcup_{(s,s') \in R} \mathcal{E}_{(s,s')}$ est un système d'équations linéaires basé sur $t =_\beta t'$. De plus, la taille de $\mathcal{E}_R$ est clairement inférieure à $2.|t,t'|^{+\ 3}_{dag}$. □

Dans toute la suite, $\mathcal{E}^{=_\beta}_{t,t'}$ sera toujours le système d'équations linéaires basé sur $t =_\beta t'$ défini dans la preuve ci-dessus (lemme 5.7.1.4). En effet :

Remarque : Le système d'équations linéaires $\mathcal{E}^{=_\beta}_{t,t'}$ (basé sur $t =_\beta t'$) comme construit au lemme 5.7.1.4 est déterminé de manière unique.

En outre, nous aurons aussi besoin de baser un système d'équations linéaires sur la notion d'équivalence modulo une valuation ($\approx_\beta$), en plus de la notion déjà définie de système d'équations linéaires basé sur l'équivalence modulo une valuation ($=_\beta$). Ainsi :

Définition 5.7.1.5 *Système d'équations linéaires basé sur une valuation et une équivalence.*

Pour toute valuation β et tous messages ou produits ouverts t et t', on dit qu'un système *(d'équations linéaires) $\mathcal{E}$* est basé sur $t \approx_\beta t'$ *quand :*

- $\beta \models \mathcal{E}$
- *Pour toute valuation β' telle que $\beta' \models \mathcal{E}$, on a $t \approx_{\beta'} t'$.*

De manière à construire de tels systèmes d'équation linéaires à t et β donnés, on introduit une notion de tuple. L'idée derrière cette notion est, étant donné une solution (à coefficients "trop importants") du problème de l'insécurité de protocole, de construire une autre solution équivalente mais n'utilisant que des coefficients de taille bornée polynomialement par la taille du protocole.

Définition 5.7.1.6 *Tuple sur une valuation et un message générique ouvert.*

Soient β une valuation et t un message ou un produit ouvert. Pour tout message ou produit ouvert t' et tout système d'équations linéaires $\mathcal{E}$, on dit que $(t', \mathcal{E})$ est un tuple sur β et t quand :

– $\beta(t') = \ulcorner\beta(t)\urcorner$, et

– $\mathcal{E}$ est basé sur $t \approx_\beta t'$, i.e. $\beta \models \mathcal{E}$ et pour tout β' tel que $\beta' \models \mathcal{E}$, $t \approx_{\beta'} t'$.

De plus, t' est appelé un message générique basé sur (β, t).

En utilisant la notion de système d'équations linéaires basé sur $=_\mathcal{E}$, nous avons la possibilité de construire un système d'équations linéaires basé sur $\approx_\beta$ à parti d'un tuple sur β (et un message générique ouvert t). Formellement :

Lemme 5.7.1.7 *Système d'équations linéaires créé sur un tuple.*

Soit β une valuation, et soient t et t' deux messages ou produits ouverts tels que $t \approx_\beta t'$. De plus, supposons qu'il existe un tuple $(s, \mathcal{E})$ sur β et t et un tuple $(s', \mathcal{E}')$ sur β et t'. Alors il existe un système d'équations linéaires basé sur $s =_\beta s'$, que l'on nome $\mathcal{E}^{=_\beta}_{s,s'}$, et :

$$\mathcal{E}^{\approx_\beta}_{t,t'} = \mathcal{E} \cup \mathcal{E}' \cup \mathcal{E}^{=_\beta}_{s,s'}$$

est un système d'équations linéaires basé sur $t \approx_\beta t'$.

Preuve. Commençons par montrer qu'il existe un système d'équations linéaires basé sur $s =_\beta s'$. En effet, on a $\beta(s) = \ulcorner\beta(t)\urcorner = \ulcorner\beta(t')\urcorner = \beta(s')$, ce qui prouve $s =_\beta s'$. On peut donc utiliser le lemme 5.7.1.4, d'où l'existence d'un système d'équations linéaires basé sur $s =_\beta s'$ que l'on note $\mathcal{E}^{=_\beta}_{s,s'}$.

A présent, on doit montrer que $\mathcal{E}^{\approx_\beta}_{t,t'}$ est un système d'équations linéaires basé sur $t \approx_\beta t'$. Par construction, on a $\beta \models \mathcal{E}^{\approx_\beta}_{t,t'}$. Posons β' tel que $\beta' \models \mathcal{E}^{\approx_\beta}_{t,t'}$. On doit prouver que $\ulcorner\beta'(t)\urcorner = \ulcorner\beta'(t')\urcorner$. Comme $\beta' \models \mathcal{E}^{=_\beta}_{s,s'}$, on a $\beta(s) = \beta(s')$. De plus, comme $\beta' \models \mathcal{E}$ (resp. $\beta' \models \mathcal{E}'$) on obtient $\ulcorner\beta'(s)\urcorner = \ulcorner\beta'(t)\urcorner$ (resp. $\ulcorner\beta'(s')\urcorner = \ulcorner\beta'(t')\urcorner$). Ainsi, $\ulcorner\beta'(t)\urcorner = \ulcorner\beta'(s)\urcorner = \ulcorner\beta'(s')\urcorner = \ulcorner\beta'(t')\urcorner$ ce qui prouve le lemme. □

Pour tenter de donner une idée de la structure de cette section, un schéma simplifié de preuve est donné à la Figure 5.3. Les flèches d'une propriété à une autre représentent une propriété utilisée pour en prouver une autre. De plus, les propriétés de la Figure correspondent aux lemmes, propositions et théorème suivants (on vient de prouver A et B) :

A : Proposition 5.7.2.1 B : Lemme 5.7.3.2 C : Lemme 5.7.3.5
D : Proposition 5.7.4.2 E : Proposition 5.7.4.3 F : Lemme 5.7.5.2
G : Proposition 5.7.5.3 H : Théorème 5.7.5.4

5.7.2 Existence des tuples.

Le but de cette sous section est de montrer qu'il existe toujours au moins un tuple basé sur un message générique ouvert et une valuation donnés. Nous bornerons les tailles des tuples dans la sous section suivante, ce qui nous permettra finalement de prouver l'existence de systèmes d'équations linéaires de tailles bornées. Mais pour le moment, nous nous concentrons sur la propriété d'existence des tuples, i.e. :

Proposition 5.7.2.1 *Existence des tuples.*

Soient une valuation β et un message ou un produit ouvert t. Alors il existe (au moins) un tuple sur β et t.

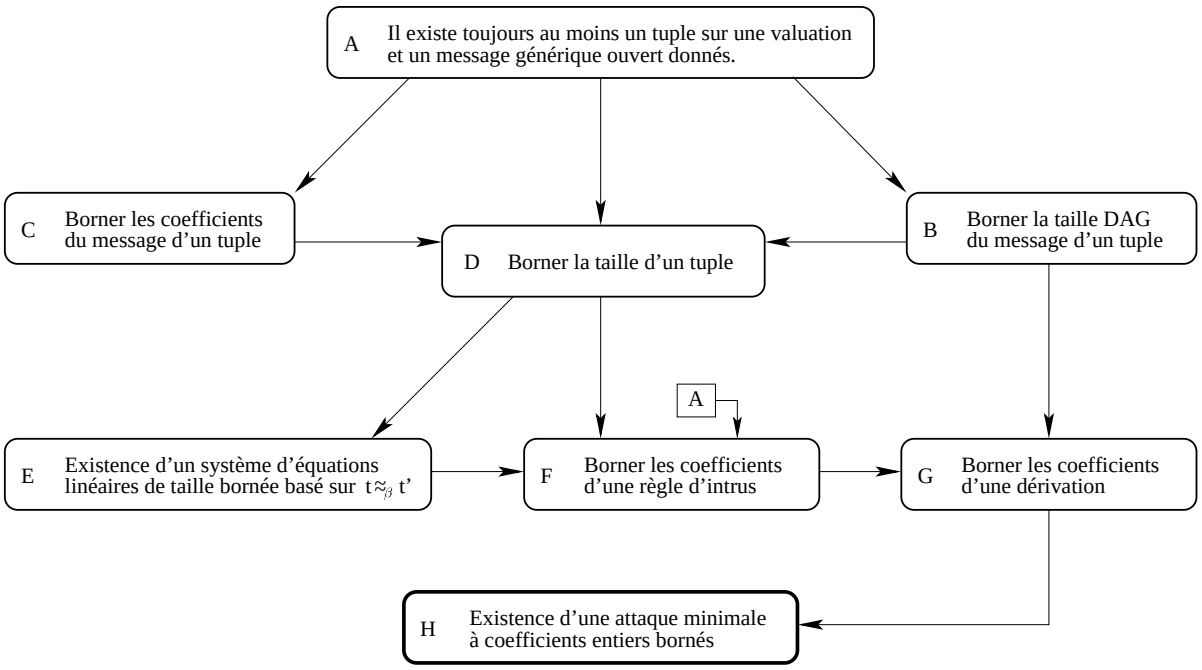


FIG. 5.3: Schéma de preuve de la section 5.7

PREUVE. Nous allons construire un tuple $(t^\beta, \mathcal{E}_t^\beta)$ sur β et t de manière inductive sur la structure de t. Au point de départ de l'induction, i.e. pour $t \in Atomes$, on pose $t^\beta = t$ et $\mathcal{E}_t^\beta = \emptyset$. Ainsi, $(t^\beta, \mathcal{E}_t^\beta)$ est bien un tuple sur β et t. Pour chaque pas le l'itération, i.e. si t n'est pas atomique et si l'on a déjà construit un tuple $(t'^\beta, \mathcal{E}_{t'}^\beta)$ sur β et t' pour chaque message ou produit ouvert t' de taille DAG inférieure à (celle de) t, alors :

Si $t = \langle t_1, t_2 \rangle$, on a deux tuples $(t_1^\beta, \mathcal{E}_{t_1}^\beta)$ et $(t_2^\beta, \mathcal{E}_{t_2}^\beta)$ sur β et t_1 et sur β et t_2 respectivement. Posons $t^\beta = \left\langle t_1^\beta, t_2^\beta \right\rangle$ et $\mathcal{E}_t^\beta = \mathcal{E}_{t_1}^\beta \cup \mathcal{E}_{t_2}^\beta$. On a ainsi construit un tuple $(t^\beta, \mathcal{E}_t^\beta)$ sur β et t. De la même manière, on construit un tuple $(t^\beta, \mathcal{E}_t^\beta)$ sur β et t quand t est une encryption symétrique ou asymétrique.

Si $t = t_1^{e_1} \cdot .. \cdot t_n^{e_n}$, on a un tuple $(t_i^\beta, \mathcal{E}_{t_i}^\beta)$ sur β et t_i pour chaque i. Soient $C_1, \ldots, C_l \subseteq \{1, t_1, \ldots, t_n\}$ des classes d'équivalence de $1, t_1, \ldots, t_n$ modulo $\approx_\beta$. Posons, pour chaque $j \in \{1, .., l\}$, $e_{C_j} = \Sigma_{t_i \in C_j} e_i$ et choisissons un $s_{C_j} \in C_j$ comme représentant de la classe C_j. On suppose, sans perte de généralité, que $1 \in C_1$, i.e. que pour tout $s \in C_1$, $s \approx_\beta 1$. Grâce à l'induction, on sait que $s^\beta = 1$ pour tout $s \in C_1$, puisque $\beta(s^\beta) = \ulcorner \beta(s) \urcorner = 1$ (définition de tuple). Posons alors $J = \{j \in \{2, \ldots l\} \mid \beta(e_{C_j}) = 0\}$. Pour tout ensemble $C = \{s_1, \ldots, s_k\}$ dont tous les s_j sont équivalents deux à deux selon β, et avec pour chaque j un message (générique) s_j^β basé sur (β, s_j), on définit :

$$\mathcal{E}_C^\beta = \mathcal{E}_{\{s_1,\ldots,s_k\}}^\beta = \bigcup_{i \neq j} \mathcal{E}^{=_\beta}_{s_i^\beta, s_j^\beta}.$$

Il est important de remarquer ici que $\mathcal{E}^{=_\beta}_{s_i^\beta, s_j^\beta}$. En effet, on a $\beta(s_i^\beta) = \ulcorner \beta(s_i) \urcorner = \ulcorner \beta(s_j) \urcorner = \beta(s_j^\beta)$, et donc $s_i^\beta =_\beta s_j^\beta$ pour tous i et j. Ainsi, on peut appliquer le lemme 5.7.1.4, d'où l'existence de

$\mathcal{E}^{=_\beta}_{s_i^\beta, s_j^\beta}$ pour tous i et j. On peut à présent définit t^β :

$$\begin{array}{rl} \text{Si } \forall j \in \{2,..,l\},\ \beta(e_{C_j}) = 0, \text{ alors :} & t^\beta = 1 \\ \text{Sinon, on pose } J = \{j \in \{2,\dots l\} \mid \beta(e_{C_j}) = 0\}, \text{ et :} & t^\beta = \Pi_{j \notin J \cup \{1\}} (s^\beta_{C_j})^{e_{C_j}}. \end{array}$$

De plus, on définit un système d'équations linéaires correspondant (pour former un tuple) :

$$\mathcal{E}^\beta_t = \bigcup_{i=1}^{n} \mathcal{E}^\beta_{t_i} \cup \bigcup_{j=2}^{l} \mathcal{E}^\beta_{C_j} \cup \bigcup_{j \in J} \{e_{C_j} = 0\}.$$

Il nous reste à montrer que $(t^\beta, \mathcal{E}^\beta_t)$ est bien un tuple sur β et t. Grâce à l'induction, on sait que $\beta \models \mathcal{E}^\beta_t$. De plus, par définition de la fonction de normalisation, on vérifie aisément que si $\beta(e_{C_j}) = 0$ pour tout $j \in \{2,..,l\}$, alors $\ulcorner \beta(t) \urcorner = 1$, et donc $t^\beta = \ulcorner \beta(t) \urcorner$. Dans le cas contraire, on a $\ulcorner \beta(t) \urcorner = \Pi_{j \notin J \cup \{1\}} \ulcorner \beta(s_{C_j}) \urcorner^{\beta(e_{C_j})}$, et par induction $\beta(s^\beta_{C_j}) = \ulcorner \beta(s_{C_j}) \urcorner$. Ainsi, $\beta(t^\beta) = \ulcorner \beta(t) \urcorner$ dans les deux cas, ce qui prouve le premier point de la définition de tuple. A présent, soit β' une valuation telle que $\beta' \models \mathcal{E}^\beta_t$, et soient $s, s' \in C_j$ tels que $s \neq s'$ $(j > 1)$. Par définition de $\mathcal{E}^\beta_t$, on a $\beta' \models \mathcal{E}^\beta_s \cup \mathcal{E}^\beta_{s'} \cup \mathcal{E}^{=_\beta}_{s,s'} (= \mathcal{E}^{\approx_\beta}_{s,s'})$. Ainsi, grâce au lemme 5.7.1.7, on obtient $\ulcorner \beta'(s) \urcorner = \ulcorner \beta'(s') \urcorner$. De plus, pour tout $s \in C_1$, on sait que $s^\beta = 1$, d'où $\beta'(s^\beta) = 1$. Au final, on a $\beta'(e_{C_j}) = 0$ pour tout $j \in J$. Ainsi, on a bien $\ulcorner \beta'(t) \urcorner = \ulcorner \beta'(t^\beta) \urcorner$, et $(t^\beta, \mathcal{E}^\beta_t)$ est un tuple sur β et t.

Si $t = Exp(u,\, M)$ et $\ulcorner \beta(u) \urcorner \neq Exp(\cdot, \cdot)$, alors par induction il existe un tuple $(u^\beta, \mathcal{E}^\beta_u)$ sur β et u, et un tuple $(M^\beta, \mathcal{E}^\beta_M)$ sur β et M. On pose :

$$\begin{array}{rl} \text{Si } \ulcorner \beta(M) \urcorner = 1, \text{ alors :} & t^\beta = u^\beta, \\ \text{Sinon :} & t^\beta = Exp(u^\beta,\, M^\beta). \end{array}$$

On pose dans les deux cas :

$$\mathcal{E}^\beta_t \quad = \quad \mathcal{E}^\beta_u \cup \mathcal{E}^\beta_M.$$

Il nous faut à présent montrer (par induction) que $(t^\beta, \mathcal{E}^\beta_t)$ est bien un tuple sur β et t. On remarque tout d'abord que par construction, $\beta \models \mathcal{E}^\beta_t$. On veut donc prouver que $\beta(t^\beta) = \ulcorner \beta(t) \urcorner$ et que $\ulcorner \beta'(t) \urcorner = \ulcorner \beta'(t^\beta) \urcorner$ pour toute valuation β' telle que $\beta' \models \mathcal{E}^\beta_t$. Pour cela, on examine les deux cas de la définition de t^β :

(a) Supposons que $\ulcorner \beta(M) \urcorner = 1$, et donc que $\ulcorner \beta(t) \urcorner = \ulcorner \beta(u) \urcorner$. On a (par induction) : $\ulcorner \beta(t) \urcorner = \ulcorner \beta(u) \urcorner = \beta(u^\beta) = \beta(t^\beta)$. On a également $\beta(M^\beta) = \ulcorner \beta(M) \urcorner = 1$, d'où $M^\beta = 1$. A nouveau grâce à l'induction, on obtient $1 = \ulcorner \beta'(M^\beta) \urcorner = \ulcorner \beta'(M) \urcorner$. En conséquence,

$$\ulcorner \beta'(t) \urcorner = \ulcorner \beta'(u) \urcorner \overset{(*)}{=} \ulcorner \beta'(u^\beta) \urcorner \overset{(**)}{=} \ulcorner \beta'(t^\beta) \urcorner$$

avec $(*)$ obtenu par induction et par définition de $\mathcal{E}^\beta_t$, et $(**)$ obtenu par définition de t, ce qui prouve de $(t^\beta, \mathcal{E}^\beta_t)$ est bien un tuple sur β et t dans ce cas.

(b) Supposons maintenant que $\ulcorner \beta(M) \urcorner \neq 1$. On a :

$$\ulcorner \beta(t) \urcorner = Exp(\ulcorner \beta(u) \urcorner, \ulcorner \beta(M) \urcorner) \overset{(*)}{=} Exp(\beta(u^\beta), \beta(M^\beta)) \overset{(**)}{=} \beta(t^\beta)$$

avec $(*)$ obtenu par induction et $(**)$ par définition de t. De plus, soit β' une valuation telle que $\beta' \models \mathcal{E}_t^\beta$. On a :

$$\begin{array}{lll} \ulcorner \beta'(t) \urcorner & = \ulcorner Exp(\ulcorner \beta'(u) \urcorner, \ulcorner \beta'(M) \urcorner) \urcorner = \ulcorner Exp(\ulcorner \beta'(u^\beta) \urcorner, \ulcorner \beta'(M^\beta) \urcorner) \urcorner & (*) \\ & = \ulcorner Exp(\beta'(u^\beta), \beta'(M^\beta)) \urcorner = \ulcorner \beta'(t^\beta) \urcorner & (**) \end{array}$$

avec $(*)$ obtenu par induction et définition de $\mathcal{E}_t^\beta$, et $(**)$ obtenu par définition de t^β. Ainsi, $(t^\beta, \mathcal{E}_t^\beta)$ est bien un tuple sur β et t ici aussi.

Si $t = Exp(u, M)$ et $\ulcorner \beta(u) \urcorner = Exp(u', M')$, alors par induction on sait qu'il existe un tuple $(u^\beta, \mathcal{E}_u^\beta)$ sur β et u. En particulier, on a $\beta(u^\beta) = \ulcorner \beta(u) \urcorner$, et donc u^β est nécessairement de la forme $Exp(u'', M'')$ avec $\beta(u'') = u'$ et $\beta(M'') = M'$. De plus, (encore) par induction, on sait qu'il existe un tuple $((M'' \bullet M)^\beta, \mathcal{E}^\beta_{(M'' \bullet M)})$ sur β et $(M'' \bullet M)$. Posons :

$$\begin{array}{rl} \text{Si } \ulcorner \beta(M'' \bullet M) \urcorner = 1, \text{ i.e. } \ulcorner M' \bullet \beta(M) \urcorner = 1, \text{ alors :} & t^\beta = u'' \\ \text{Sinon :} & t^\beta = Exp(u'', (M'' \cdot M)^\beta) \end{array}$$

Dans les deux cas, on pose également :

$$\mathcal{E}_t^\beta = \mathcal{E}_u^\beta \cup \mathcal{E}^\beta_{(M'' \bullet M)}$$

Il nous reste à montrer que $(t^\beta, \mathcal{E}_t^\beta)$ est bien un tuple pour β et t. Tout d'abord, on a naturellement $\beta \models \mathcal{E}_t^\beta$. On doit donc montrer que $\beta(t^\beta) = \ulcorner \beta(t) \urcorner$ et $\ulcorner \beta'(t) \urcorner = \ulcorner \beta'(t^\beta) \urcorner$ pour toute valuation β' telle que $\beta' \models \mathcal{E}_t^\beta$. Pour cela, posons $(u^\beta, \mathcal{E}_u^\beta) = Exp(u'', M'')$ défini comme comme ci-dessus. On a alors $\ulcorner \beta(t) \urcorner = \ulcorner Exp(\ulcorner \beta(u) \urcorner, \beta(M)) \urcorner = \ulcorner Exp(u', \ulcorner M' \cdot \beta(M) \urcorner) \urcorner$. De plus, si $\ulcorner M' \bullet \beta(M) \urcorner = 1$, alors $\ulcorner \beta(t) \urcorner = u' = \beta(u'') = \beta(t^\beta)$. Et sinon, on a :

$$\begin{array}{lll} \ulcorner \beta(t) \urcorner & = Exp(u', \ulcorner M' \bullet \beta(M) \urcorner) & \\ & = Exp(\beta(u''), \ulcorner \beta(M'') \bullet \beta(M) \urcorner) & \\ & = Exp(\beta(u''), \ulcorner \beta(M'' \bullet M) \urcorner) & \\ & = Exp(\beta(u''), \beta((M'' \bullet M)^\beta)) & (*) \\ & = \beta(t^\beta) & (**) \end{array}$$

avec $(*)$ obtenu par induction, et $(**)$ par définition de t^β. A présent, soit β' une valuations telle que $\beta' \models \mathcal{E}_t^\beta$. On sait déjà que $\ulcorner \beta'(t) \urcorner = \ulcorner Exp(\beta'(u), \beta'(M)) \urcorner$. Comme $\beta' \models \mathcal{E}_u^\beta$, on obtient grâce à l'induction que $\ulcorner \beta'(u) \urcorner = \ulcorner \beta'(u^\beta) \urcorner = \ulcorner \beta'(Exp(u'', M'')) \urcorner$. Ainsi :

$$\ulcorner \beta'(t) \urcorner = \ulcorner Exp(\beta'(u^\beta), \beta'(M)) \urcorner = \ulcorner Exp(\beta'(u''), \beta'(M'') \cdot \beta'(M)) \urcorner = \ulcorner Exp(\beta'(u''), \beta'(M'' \cdot M)) \urcorner$$

En outre, comme $\beta' \models \mathcal{E}^\beta_{(M'' \bullet M)}$, on obtient par induction que $\ulcorner \beta'((M'' \cdot \bullet M)^\beta) \urcorner = \ulcorner \beta'(M'' \bullet M) \urcorner$, d'où $\ulcorner \beta'(t) \urcorner = \ulcorner Exp(\beta'(u''), \beta'((M'' \cdot M)^\beta)) \urcorner$. Enfin, si $t^\beta = Exp(u'', (M'' \bullet M)^\beta)$, on obtient $\ulcorner \beta'(t) \urcorner = \ulcorner t^\beta \urcorner$. Sinon, on a $t^\beta = u''$ et $\ulcorner \beta(M'' \bullet M) \urcorner = 1$, d'où $(M'' \bullet M)^\beta = 1$ et $\ulcorner \beta'(t) \urcorner = \ulcorner t^\beta \urcorner$. Ainsi, $(t^\beta, \mathcal{E}_t^\beta)$ est bien un tuple sur β et t.

En résumé, on vient d'examiner une liste exhaustive de structures possibles pour t, en construisant à chaque fois un tuple sur β et t. Ainsi, on a bien au moins un tuple pour toute valuation β et tout message ou produit ouvert t, par induction structurelle sur t, ce qui prouve la proposition. □

5.7.3 Borner la taille du message dans un tuple.

A partir de maintenant, nous noterons toujours t^β le message générique ouvert basé sur (β, t) tel qu'il a été construit dans la preuve de la proposition 5.7.2.1 ci-dessus. Notre but à présent est de montrer que pour tous β et t, il existe toujours un tuple sur β et t de taille bornée polynomialement par $\|t\|$. Dans ce but, nous allons borner dans cette sous section la taille de t^β. Nous bornerons la taille du système d'équations linéaires correspondant dans la section suivante.

Tout d'abord, nous allons montrer que les messages ou produits ouverts basés sur (β, t) sont déterminés de manière unique au moins quand $\beta(t)$ est normalisé. En effet :

Lemme 5.7.3.1 *Unicité des messages basés dans le cas normalisé.*
Pour tout message ou produit ouvert t tel que $\beta(t) = \ulcorner\beta(t)\urcorner$, on a $t^\beta = t$.

Preuve. Nous allons prouver ce résultat par induction structurelle sur t, i.e. en suivant les différents points de la preuve de la proposition 5.7.2.1 définissant t^β. Tout d'abord, si t est atomique, si t est une paire, ou si t est une encryption, alors la propriété est évidente. De plus :

Si $t = t_1^{e_1} \cdot .. \cdot t_n^{e_n}$, on sait que $\beta(t_i) = \ulcorner\beta(t_i)\urcorner \neq 1$ pour tout i, que $\ulcorner\beta(t_i)\urcorner \neq \ulcorner\beta(t_j)\urcorner$ pour tous $i \neq j$, et que $\beta(e_i) \neq 0$ pour tout i. On en conclut donc naturellement que $t^\beta = t$.

Supposons à présent que $t = Exp(u, M)$. On commence par remarquer que $\ulcorner\beta(u)\urcorner \neq Exp(u', M')$ pour tous u' et M' normalisés. En effet, on aurait sinon $\ulcorner\beta(t)\urcorner = Exp(u', M'') = \beta(t)$, et donc $\beta(u) = u'$, ce qui nous conduit à $Exp(u', M') = \ulcorner\beta(u)\urcorner = \ulcorner u' \urcorner = u'$ et donc à une contradiction (u' ne peut pas être l'un de ses sous termes propres). En outre, on a $\ulcorner\beta(M)\urcorner \neq 1$ car sinon $Exp(\beta(u), \beta(M)) = \beta(t) = \ulcorner\beta(t)\urcorner = \ulcorner\beta(u)\urcorner$ alors que l'on a déjà $\ulcorner\beta(u)\urcorner \neq Exp(\cdot, \cdot)$. Ainsi, $Exp(\beta(u), \beta(M)) = \beta(t) = \ulcorner\beta(t)\urcorner = Exp(\ulcorner\beta(u)\urcorner, \ulcorner\beta(M)\urcorner)$, d'où $\beta(u) = \ulcorner\beta(u)\urcorner$ et $\beta(M) = \ulcorner\beta(M)\urcorner$. On a alors par induction, que $u^\beta = u$ et $M^\beta = M$, ce qui nous donne $t^\beta = Exp(u^\beta, M^\beta) = Exp(u, M) = t$ par définition de t^β. □

Pour tout ensemble E de messages ou produits ouverts, on pose $E^\beta = \{t^\beta \mid t \in E\}$. Le lemme suivant va nous permettre de borner $\left|t^\beta\right|^+_{dag}$, i.e. le nombre de sous termes étendus de t^β. Ceci nous permettra, quand nous aurons borné $\left|t^\beta\right|_{exp}$, de borner $\|t\|^+$.

Lemme 5.7.3.2 *Borne sur $\left|t^\beta\right|^+_{dag}$, avec t^β un message générique ouvert basé sur β et t.*
Pour toute valuation β et tous messages ou produits ouverts $t, t_1, .., t_n$, on a :

1. $STermes(t^\beta) \subseteq STermes(t)^\beta$
2. $\left|\ulcorner\beta(t)\urcorner\right|_{dag} \leq |t|_{dag}$ *et* $\left|\ulcorner\beta(t_1)\urcorner, .., \ulcorner\beta(t_1)\urcorner\right|_{dag} \leq |t_1, .., t_n|_{dag}$
3. $\left|t^\beta\right|^+_{dag} \leq 2.|t|_{dag}$.

Preuve. Nous allons examiner ces trois points un par un.

Point 1 : Nous allons procéder (vu la définition, on n'a pas beaucoup d'autres choix) par induction structurelle sur t.

Si $t \in Atomes$, on a $STermes(t^\beta) = \{t\} = STermes(t)^\beta$.

Si $t = \langle t_1, t_2 \rangle$, alors par induction :

$$\begin{aligned} STermes(t^\beta) &= \{t^\beta\} \cup STermes(t_1^\beta) \cup STermes(t_2^\beta) \\ &\subseteq \{t^\beta\} \cup STermes(t_1)^\beta \cup STermes(t_2)^\beta = STermes(t)^\beta \end{aligned}$$

De même si t est une encryption.

Si $t = t_1^{e_1} \cdots t_n^{e_n}$, alors on a deux (sous) cas. D'une part, si $t^\beta = 1$, on a naturellement $STermes(t^\beta) \subseteq STermes(t)^\beta$. D'autre part, si $t^\beta \neq 1$, alors $t^\beta = \Pi_{j \notin J \cup \{1\}} (s^\beta_{C_j})^{e_{C_j}}$ (avec les notations de la preuve de la proposition 5.7.2.1), et pour chaque $s^\beta_{C_j}$ il existe t_i tel que $t_i^\beta = s^\beta_{C_j}$. Ainsi, on obtient par induction :

$$\begin{array}{rl} STermes(t^\beta) & \subseteq \{t^\beta\} \cup \bigcup_{j \notin J \cup \{1\}} STermes(s^\beta_{C_j}) \\ & \subseteq \{t^\beta\} \cup \bigcup_{i=1}^n STermes(t_i^\beta) \\ & \subseteq \{t^\beta\} \cup \bigcup_{i=1}^n STermes(t_i)^\beta = STermes(t)^\beta \end{array}$$

Si $t = Exp(u, M)$ et $t^\beta = u^\beta$, alors par induction on a immédiatement $STermes(t^\beta) \subseteq STermes(t)^\beta$.

Si $t = Exp(u, M)$, et $t^\beta = Exp(u^\beta, M^\beta)$ avec $M = t_1^{e_1} \cdots t_n^{e_n}$, alors $M^\beta \neq 1$ et, par induction,

$$\begin{array}{rl} STermes(t^\beta) & \subseteq \{t^\beta\} \cup STermes(u^\beta) \cup \bigcup_i STermes(t_i^\beta) \\ & \subseteq \{t^\beta\} \cup STermes(u)^\beta \cup \bigcup_i STermes(t_i)^\beta = STermes(t)^\beta \end{array}$$

Si $t = Exp(u, M)$, $u^\beta = Exp(u'', M'')$, et $t^\beta = u''$. Alors $STermes(t^\beta) \subseteq STermes(u^\beta) \subseteq STermes(u)^\beta \subseteq STermes(t)^\beta$.

Sinon, on a $t = Exp(u, M)$, $u^\beta = Exp(u'', M'')$, et $t^\beta = Exp(u'', (M'' \bullet M)^\beta)$ avec $(M'' \bullet M)^\beta \neq 1$. Ainsi, $(M'' \bullet M)^\beta$ est de la forme $\bigcup_{j \notin J \cup \{1\}} (s^\beta_{C_j})^{e'_j}$ avec $M'' = {t''}_1^{e''_1} \cdot .. \cdot {t''}_{n''}^{e''_{n''}}$, $M = t_1^{e_1} \cdot .. \cdot t_n^{e_n}$, et pour chaque j, il existe i tel que $s^\beta_{C_j} = t''_i$ ou $s^\beta_{C_j} = t_i^\beta$ (On remarque que $t''_i = {t''}_i^\beta$ grâce au lemme 5.7.3.1). De plus, on a $t''_i \in STermes(u^\beta) \subseteq STermes(u)^\beta$ par induction. Ainsi, on peut conclure que :

$$\begin{array}{rl} STermes(t^\beta) & = \{t^\beta\} \cup STermes(u'') \cup \bigcup_{j \notin J \cup \{1\}} STermes(s^\beta_{C_j}) \\ & \subseteq \{t^\beta\} \cup STermes(u'') \cup \bigcup_{i=1}^n STermes(t_i^\beta) \cup \bigcup_{i=1}^{n''} STermes({t''}_i) \\ & \subseteq \{t^\beta\} \cup \bigcup_{i=1}^n STermes(t_i)^\beta \cup STermes(u^\beta) \\ & \subseteq \{t^\beta\} \cup \bigcup_{i=1}^n STermes(t_i)^\beta \cup STermes(u)^\beta = STermes(t)^\beta \end{array}$$

Point 2 : Remarquons tout d'abord que $\ulcorner \beta(t) \urcorner = \beta(t^\beta)$, et que $|\beta(s)|_{dag} \leq |s|_{dag}$ pour tout message ou produit ouvert s. Ainsi, on a :

$$\begin{array}{rll} \left|\ulcorner \beta(t) \urcorner\right|_{dag} & = \left|\beta(t^\beta)\right|_{dag} \leq \left|t^\beta\right|_{dag} & \\ & \leq \#(STermes(t)^\beta) & (*) \\ & \leq \#(STermes(t)) = |t|_{dag} & \end{array}$$

avec $(*)$ obtenu grâce au point 1. On peut utiliser le même argument pour

$$\left|\ulcorner \beta(t_1) \urcorner, .., \ulcorner \beta(t_1) \urcorner\right|_{dag} \leq |t_1, .., t_n|_{dag}$$

Point 3 : C'est une conséquence immédiate du point 1 et du fait que $|u|^+_{dag} \leq 2.|u|_{dag}$ pour tout u. En effet : $\left|t^\beta\right|^+_{dag} \leq 2.\left|t^\beta\right|_{dag} \leq 2.\#(STermes(t)^\beta) \leq 2.|t|_{dag}$. □

Ainsi, nous avons pu donner une borne polynomiale sur la taille DAG étendue de t^β en fonction de la taille DAG de t. Pour pouvoir borner la taille (totale) de t^β, nous avons aussi besoin de borner la taille de ses coefficients, i.e. $\left|t^\beta\right|_{exp}$. Nous allons prouver cela en deux temps. Tout d'abord, nous examinons les coefficients d'un seul produit. Pour cela, pour tout produit ouvert $M = m_1^{e_1} \cdot .. \cdot m_n^{e_n}$, on pose $|M|_{coef} = \sum_{i=1..n} \|e_i\|$. C'est l'un des éléments formant la taille $|t|_{exp}$ d'un message ou produit ouvert t.

Lemme 5.7.3.3 *Borne sur les coefficients d'un produit.*

Soient β une valuation et t un message ou un produit ouvert tel que $t^\beta = Exp(u, M)$. Alors $|M|_{coef} \leq |t|_{dag} \times |t|_{exp} \leq \|t\|^{+2}$.

Preuve. Il nous suffit de montrer par induction structurelle sur t que $|M|_{coef} \leq |t|_{dag} \times |t|_{exp}$. En effet, avec cette inégalité et le fait que $|t|_{dag} \leq \|t\|^{+2}$ et $|t|_{exp} \leq \|t\|^{+2}$, le reste du lemme suit naturellement.

- Supposons tout d'abord que $t = Exp(u', M')$ et $\ulcorner\beta(u')\urcorner \neq Exp(\cdot, \cdot)$. On a directement $|M|_{coef} \leq |M'|_{coef} \leq |t|_{exp}$.

- Supposons à présent que $t = Exp(u', M')$ et $\ulcorner\beta(u')\urcorner = Exp(\cdot, \cdot)$. On a $u'^\beta = Exp(u'', M'')$, avec u'' un message ouvert et M'' un produit ouvert. En conséquence, on a nécessairement $M = (M'' \bullet M')^\beta$, d'où par induction $|M|_{coef} \leq |M''|_{coef} + |M'|_{coef} \leq |u'|_{dag} \times |u'|_{exp} + |t|_{exp} \leq |t|_{dag} \times |t|_{exp}$ puisque $|u'|_{exp} \leq |t|_{exp}$ et $|u'|_{dag} < |t|_{dag}$.

- Dans tous les autres cas, avec $t^\beta = Exp(u, M)$, il existe un sous terme v de t tel que $v^\beta = t^\beta$. Ainsi, on obtient simplement la borne par induction.

Le pas de l'induction structurelle sur t est donc vérifié dans tous les cas, ce qui prouve le lemme. □

Afin de borner tous les coefficients apparaissant dans un message générique ouvert donné, nous avons besoin du lemme technique suivant. L'idée sera d'itérer ce lemme, pour un ensemble E de départ bien choisi, de manière à obtenir une somme finie connue (et bornée) de taille $\|..\|^{+2}$ de sous termes de t. Concrètement :

Lemme 5.7.3.4 *Lemme technique.*

Soit E un ensemble fini de messages ou produits ouverts tel que $STermes^+(E) = E$, et soit $t \in E$ maximal dans E pour l'ordre sous terme. On a :

$$\left|\bigcup_{s\in E} STermes^+(s^\beta)\right|_{coef} \leq \left|\bigcup_{s\in E\setminus\{t\}} STermes^+(s^\beta)\right|_{coef} + \|t\|^{+2}$$

Preuve. Nous allons effectuer (comme d'habitude) une induction structurelle sur t. Pour le cas de base, i.e. quand t est atomique, on a naturellement :

$$\left|\bigcup_{s\in E} STermes^+(s^\beta)\right|_{coef} = \left|\bigcup_{s\in E\setminus\{t\}} STermes^+(s^\beta)\right|_{coef}$$

Pour le pas de l'induction, on examine la structure de t :

- Si $t = \langle t_1, t_2\rangle$, alors $t^\beta = \left\langle t_1^\beta, t_2^\beta \right\rangle$ et donc $STermes^+(t^\beta) \subseteq \{t^\beta\} \cup STermes^+(t_1^\beta) \cup STermes^+(t_2^\beta)$. On sait que $t_1, t_2 \in E\setminus t$ puisque $STermes^+(E) = E$. Et comme $\left|t^\beta\right|_{coef} = 0$, on

obtient $\left|\bigcup_{s\in E} STermes^+(s^\beta)\right|_{coef} = \left|\bigcup_{s\in E\setminus\{t\}} STermes^+(s^\beta)\right|_{coef}$. En outre, on peut appliquer les mêmes arguments pour les deux types d'encryption.

- Si $t = t_1^{e_1}\cdots t_n^{e_n}$, alors $STermes^+(t^\beta) \subseteq \{t^\beta\} \cup \bigcup_i STermes^+(t_i^\beta)$. On sait déjà que $t_1, \ldots, t_n \subseteq E\setminus\{t\}$, et donc que $STermes^+(t^\beta) \subseteq \bigcup_{s\in E\setminus\{t\}} STermes^+(s^\beta) \cup \{t^\beta\}$. En outre, on a $\left|t^\beta\right|_{coef} \leq |t|_{coef} \leq \|t\|^{+2}$, ce qui prouve le lemme naturellement.

- A présent, supposons que $t = Exp(u, M)$ et soit $t^\beta = u^\beta$ (1), soit $t^\beta = u''$ (2), soit $t^\beta = Exp(u^\beta, M^\beta)$ (3). On suit ici les différents cas de la proposition 5.7.2.1. Pour (1) et (2), on a :

$$\begin{array}{rl} STermes^+(t^\beta) & \subseteq STermes^+(u^\beta) \cup STermes^+(M^\beta) \\ & \subseteq \bigcup_{s\in E\setminus\{t\}} STermes^+(s^\beta). \end{array}$$

De plus, pour (3) on a :

$$\begin{array}{rl} STermes^+(t^\beta) & \subseteq \{t^\beta\} \cup STermes^+(u^\beta) \cup STermes^+(M^\beta) \\ & \subseteq \{t^\beta\} \cup \bigcup_{s\in E\setminus\{t\}} STermes^+(s^\beta) \end{array}$$

et $\left|t^\beta\right|_{coef} = 0$. Ainsi, on a toujours $\left|\bigcup_{s\in E} STermes^+(s^\beta)\right|_{coef} = \left|\bigcup_{s\in E\setminus\{t\}} STermes^+(s^\beta)\right|_{coef}$.

- Enfin, supposons que $t = Exp(u, M)$, avec $\ulcorner\beta(u)\urcorner = Exp(u', M')$ et $u^\beta = Exp(u'', M'')$. On a $t^\beta = Exp(u'', (M''\bullet M)^\beta)$, et donc :

$$\begin{array}{rl} STermes^+(t^\beta) & \subseteq \{t^\beta\} \cup STermes^+(u^\beta) \cup STermes^+((M''\bullet M)^\beta) \\ & \subseteq \{t^\beta\} \cup STermes^+(u^\beta) \cup \{(M''\bullet M)^\beta\} \cup STermes^+(M^\beta). \end{array}$$

En effet, grâce au lemme 5.7.3.1 on a $STermes^+(M''^\beta) = STermes^+(M'')$, et $STermes^+(M'') \subseteq STermes^+(u^\beta)$. En conséquence, on obtient :

$$STermes^+(t^\beta) \subseteq \{t^\beta\} \cup \{(M''\bullet M)^\beta\} \cup \bigcup_{s\in E\setminus\{t\}} STermes^+(s^\beta)$$

De plus, on sait que $\left|t^\beta\right|_{coef} = 0$, et grâce au lemme 5.7.3.3 on a $\left|(M''\bullet M)^\beta\right|_{coef} \leq \|t\|^{+2}$. Le lemme est donc prouvé dans tous les cas. □

Nous avons à présent tous les outils nécessaires pour donner assez simplement une borne sur la totalité des coefficients de t^β. Formellement :

Lemme 5.7.3.5 *Borne sur tous les coefficients entiers dans un message.*
Soient β une valuation et t un message ou un produit ouvert. On a $\left|t^\beta\right|_{exp} \leq \|t\|^{+3}$.

Preuve. Nous allons itérer le lemme technique 5.7.3.4 précédent. En effet, avec $STermes^+(t) = E$ on a $\left|t^\beta\right|_{exp} = \left|STermes^+(t^\beta)\right|_{coef} \leq \left|\bigcup_{s\in E} STermes^+(s^\beta)\right|_{coef}$. Le lemme 5.7.3.4 nous permet d'extraire itérativement tous les éléments de E (par ordre sous terme décroissant). On obtient alors $\left|\bigcup_{s\in E} STermes^+(s^\beta)\right|_{coef} \leq |t|^+_{dag} \times \|t\|^{+2}$ (On remarque que $\#E = |t|^+_{dag}$). Ainsi, on obtient $\left|t^\beta\right|_{exp} \leq \|t\|^{+3}$ et le lemme est prouvé. □

Nous venons de borner, d'un côté le nombre de sous termes étendus de t^β ($\left|t^\beta\right|^+_{dag}$, lemme 5.7.3.2), et d'un autre côté les taille de tous les coefficients entiers de t^β ($\left|t^\beta\right|_{exp}$, lemme 5.7.3.5), en fonction de la taille de t. Or la taille (étendue) de t^β n'est autre que la somme de ces deux valeurs, ce qui nous donne la proposition suivante (but de cette sous section) :

Proposition 5.7.3.6 *Borne sur la taille (totale) de t^β.*

Pour toute valuation β et tout message ou produit ouvert t, on a $\left\|t^\beta\right\|^+ \leq 3.\|t\|^3$.

5.7.4 Borner la taille du système d'équations dans un tuple.

Nous avons vu, dans la section précédente, comment borner la taille (totale) d'un message générique t' ouvert formant la première composante d'un tuple sur β et t donnés. Cependant, un tuple a pour seconde composante un système d'équations linéaires basé sur $t \approx_\beta t'$. Dans cette sous section, nous allons prouver l'existence de tels systèmes d'équations linéaires dont la taille est bornée par un polynôme en la taille (étendue) de t, i.e. $\|t\|^+$. Pour cela, étant donnés une valuation β et un message ou produit ouvert t, on définit le système d'équations linéaires ${\mathcal{E}'}_t^\beta$ suivant, de manière inductive sur la structure de t :

- Si t est un atome, une paire, ou une encryption, on pose ${\mathcal{E}'}_t^\beta = \emptyset$. (Pas de contrainte dans ce cas).
- Si $t = t_1^{e_1} \cdots t_n^{e_n}$, en reprenant les notations du lemme 5.7.2.1, on pose ${\mathcal{E}'}_t^\beta = \bigcup_{j=2}^{l} \mathcal{E}_{C_j}^\beta \cup \bigcup_{j \in J} \{e_{C_j} = 0\}$.
- Si $t = Exp(u, M)$, on pose :

$$\begin{array}{ll} \text{Si } \ulcorner \beta(u) \urcorner \neq Exp(\cdot,\cdot) \text{ alors :} & {\mathcal{E}'}_t^\beta = \emptyset \\ \text{Si } \ulcorner \beta(u) \urcorner = Exp(\cdot,\cdot) \text{ alors :} & {\mathcal{E}'}_t^\beta = {\mathcal{E}'}_{M'' \bullet M}^\beta \text{ avec } u^\beta = Exp(u'', M'') \end{array}$$

On remarque aisément que le système d'équations linéaires ${\mathcal{E}'}_t^\beta$ est déterminé de manière unique (modulo AC_+, i.e. l'associativité et la commutativité de l'opérateur d'addition dans les expressions linéaires). Cependant, ce système d'équations n'est pas suffisant pour décrire toutes les contraintes liées à un message (ou un produit) t donné, puisqu'il ne donne que les contraintes "en tête" du message. Nous définissons donc le système suivant, pour toute valuation β et tout message ou produit t ouvert :

$$\mathcal{E}_t^\beta = \bigcup_{s \in STermes^+(t)} {\mathcal{E}'}_s^\beta$$

De cette manière, on obtient un ensemble regroupant toutes les contraintes liées à t (i.e. le système d'équations linéaires devant être satisfait pour que l'on puisse espérer, par la suite, construire de nouvelles attaques à coefficients bornés à partir d'attaques existantes). Nous avons défini ce système d'équations linéaires dans un but bien précis : montrer que $(t^\beta, \mathcal{E}_t^\beta)$ est un tuple sur β et t, et que la taille de $\mathcal{E}_t^\beta$ est bornée (polynomialement) par la taille de t. Dans ce but, nous commençons par prouver le lemme suivant :

Lemme 5.7.4.1 *Lemme technique.*

Soit β une valuation, et soient $M = t_1^{e_1} \cdot .. \cdot t_n^{e_n}$ et $M' = {t'}_1^{e'_1} \cdot .. \cdot {t'}_{n'}^{e'_{n'}}$ deux produits ouverts tels que $\beta(t'_i) = \ulcorner \beta(t'_i) \urcorner$ pour tout i.De plus, soit $(t_i^\beta, \mathcal{E}_i)$ un tuple pour β et t_i, pour tout i. Alors $((M' \cdot M)^\beta, {\mathcal{E}'}_{M' \bullet M}^\beta \cup \bigcup_i \mathcal{E}_i)$ est un tuple pour β et $M' \bullet M$.

Preuve. Posons $t = M' \bullet M$. De manière évidente, on a $\beta \models {\mathcal{E}'}_t^\beta \cup \bigcup_i \mathcal{E}_i$. De plus, on sait que $\beta(t^\beta) = \ulcorner \beta(t) \urcorner$. Soit β' une valuation telle que $\beta' \models {\mathcal{E}'}_t^\beta \cup \bigcup_i \mathcal{E}_i$. On doit prouver que $\ulcorner \beta'(t^\beta) \urcorner = \ulcorner \beta'(t) \urcorner$.

Tout d'abord, montrons que $\ulcorner\beta'(t_i)\urcorner = \ulcorner\beta'(t_i^\beta)\urcorner$ et $\ulcorner\beta'(t'_j)\urcorner = \ulcorner\beta'(t'^\beta_j)\urcorner$ pour tous i et j. En effet, comme $\beta' \models \mathcal{E}_i$, on a immédiatement $\ulcorner\beta'(t_i)\urcorner = \ulcorner\beta'(t_i^\beta)\urcorner$. De plus, le lemme 5.7.3.1 implique que $t'^\beta_j = t'_j$. On a donc bien :

$$\ulcorner\beta'(t'_j)\urcorner = \ulcorner\beta'(t'^\beta_j)\urcorner \qquad (1)$$

Soient $C_1, \ldots, C_l$ les classes d'équivalence définies comme lors de la preuve du lemme 5.7.2.1. On veut à présent montrer que :

$$\ulcorner\beta'(s)\urcorner = \ulcorner\beta'(s')\urcorner \text{ pour tous } k \text{ et } s, s' \in C_k \qquad (2)$$

En effet, on a plusieurs cas : Supposons en premier qu'il existe i et j tels que $s = t'_i$ et $s' = t'_j$. Grâce au lemme 5.7.3.1, on a $s^\beta = s$ et $s'^\beta = s'$. Ainsi, par définition de $\mathcal{E}'^\beta_t$, on obtient $\beta'(s) = \beta'(s')$ comme annoncé. Maintenant, supposons qu'il existe i et j tels que $s = t_i$ et $s' = t'_j$. A nouveau, on a $s'^\beta = s'$, et la définition de $\mathcal{E}'^\beta_t$ nous fournit $\beta'(s') = \beta'(s^\beta)$. De plus, comme $\beta' \models \mathcal{E}_i$, on a $\ulcorner\beta'(s)\urcorner = \ulcorner\beta'(s^\beta)\urcorner$, ce qui nous donne bien $\ulcorner\beta'(s')\urcorner = \ulcorner\beta'(s)\urcorner$. On traite de la même manière le cas $s = t_i$ et $s' = t_j$, ce qui finit de montrer la propriété annoncée.

On peut alors démontrer le lemme assez simplement. En effet, par définition de $\mathcal{E}'^\beta_t$, on sait que $\beta'(e_{C_j}) = 0$ pour tout $j \in J$. Ainsi, en utilisant les deux propriétés (1) et (2) ci-dessus, on obtient $\ulcorner\beta'(t^\beta)\urcorner = \ulcorner\beta'(t)\urcorner$. □

Nous pouvons à présent montrer que non seulement $(t^\beta, \mathcal{E}^\beta_t)$ est un tuple sur β et t, mais en plus la taille du système d'équations linéaires $\mathcal{E}^\beta_t$ est bornée polynomialement par la taille de t. Formellement :

Proposition 5.7.4.2 *Validité de $\mathcal{E}^\beta_t$ et borne sur sa taille.*
Pour toute valuation β et tout message ou produit ouvert t, on a :

1) *$(t^\beta, \mathcal{E}^\beta_t)$ est un tuple sur β et t.*
2) *La taille de $\mathcal{E}'^\beta_t$ est bornée polynomialement par $\|t\|^+$.*
3) *La taille de $(t^\beta, \mathcal{E}^\beta_t)$, i.e. $\left\|t^\beta\right\|^+ + \left\|\mathcal{E}^\beta_t\right\|^+$, est bornée polynomialement par $\|t\|^+$.*

Preuve. Nous allons montrer ces trois points les uns après les autres.

Point 1 : Nous allons le prouver par induction structurelle sur t, en suivant la construction développée à la proposition 5.7.2.1. Tout d'abord, le cas $t \in Atomes$ est évident. Ensuite, examinons la nature de t :

- Si $t = \langle t_1, t_2 \rangle$, alors $\mathcal{E}^\beta_t = \mathcal{E}'^\beta_t \cup \bigcup_{s \in STermes^+(t_1)} \mathcal{E}'^\beta_s \cup \bigcup_{s \in STermes^+(t_2)} \mathcal{E}'^\beta_s$. Par définition, on a $\mathcal{E}^\beta_t = \mathcal{E}^\beta_{t_1} \cup \mathcal{E}^\beta_{t_2}$ (puisque le système $\mathcal{E}'^\beta_{..}$ est déterminé de manière unique). On en conclut alors, comme dans la preuve du lemme 5.7.2.1, que $(t^\beta, \mathcal{E}^\beta_t)$ est un tuple sur β et t. On traite le cas où t est une encryption (symétrique ou asymétrique) de la même manière.
- Si $t = t_1^{e_1} \cdot .. \cdot t_n^{e_n}$, en utilisant les notations de la preuve du lemme 5.7.2.1, on a grâce à la définition de $\mathcal{E}'^\beta_t$ (et puisque $\mathcal{E}'^\beta_t$ est déterminé de manière unique), que $\mathcal{E}^\beta_t = \bigcup_i \mathcal{E}^\beta_{t_i} \cup \bigcup_{j=2}^{l} \mathcal{E}^\beta_{C_j} \cup \bigcup_{j \in J} \{e_{C_j} = 0\}$. A nouveau comme dans la preuve du lemme 5.7.2.1, on en déduit que $(t^\beta, \mathcal{E}^\beta_t)$ est un tuple sur β et t.

- Si $t = Exp(u, M)$ et $\ulcorner\beta(u)\urcorner \neq Exp(\cdot,\cdot)$, alors par définition de $\mathcal{E}_t^\beta$ (et toujours puisque $\mathcal{E}'^\beta_t$ est déterminé de manière unique), on a $\mathcal{E}_t^\beta = \mathcal{E}_u^\beta \cup \mathcal{E}_M^\beta$. Comme dans la preuve du lemme 5.7.2.1, on en déduit que $(t^\beta, \mathcal{E}_t^\beta)$ est un tuple sur β et t.
- Enfin, si $t = Exp(u, M)$ et $\ulcorner\beta(u)\urcorner \neq Exp(\cdot,\cdot)$, avec $u^\beta = Exp(u'', M'')$ et $M = t_1^{e_1} \cdots t_n^{e_n}$, alors le lemme 5.7.4.1 implique que $((M''\cdot M)^\beta, \mathcal{E}'^\beta_{M''\bullet M} \cup \bigcup_i \mathcal{E}_{t_i}^\beta)$ est un tuple sur β $M''\bullet M$. Par définition de $\mathcal{E}_t^\beta$, on a alors que $\mathcal{E}_u^\beta \cup \bigcup_i \mathcal{E}_{t_i}^\beta \cup \mathcal{E}'^\beta_{M''\bullet M} \subseteq \mathcal{E}_t^\beta$. Ainsi, comme dans la preuve du lemme 5.7.2.1, on en déduit que $(t^\beta, \mathcal{E}_t^\beta)$ est un tuple sur β et t.

Point 2 : Si t est un atome, une paire, ou une encryption, la propriété est évidente. Si t est un produit, alors grâce au lemme 5.7.1.4 et à la proposition 5.7.3.6, on obtient que la taille de $\mathcal{E}'^\beta_t$ peut être bornée par un polynôme en $\|t\|^+$. Si $t = Exp(\cdot,\cdot)$, alors on obtient le polynôme désiré grâce à la proposition 5.7.3.6 et au cas où t est un produit.

Point 3 : Si p est le polynôme bornant la taille de $\mathcal{E}'^\beta_t$, alors $p(\|t\|^+) \times \|t\|^+$ borne la taille de $\mathcal{E}_t^\beta$. De plus, grâce au lemme 5.7.3.6 on sait que la taille de t^β peut être bornée par un polynôme en $\|t\|^+$. □

Nous avons donc pu borner la taille des systèmes d'équations linéaires formant un tuple, comme annoncé au début de la section. De plus, on peut pousser ce résultat un petit peu plus loin, en bornant la taille d'un système d'équations linéaires basé sur une relation $t \approx_\beta t'$ donnée. En effet, il nous suffit de rassembler les lemmes 5.7.1.4 et 5.7.1.7, et la proposition 5.7.4.2 pour avoir la proposition suivante :

Proposition 5.7.4.3 *Existence d'un système d'équations linéaires borné basé sur une relation $t \approx_\beta t'$.*

Soit β une valuation, et soient t et t' deux messages ou produits ouverts tels que $t \approx_\beta t'$. Alors il existe un système d'équations linéaires basé sur $t \approx_\beta t'$ de taille bornée par un polynôme en $\|t, t'\|^+$.

Pour toute la suite, nous noterons $\mathcal{E}_{t,t'}^{t\approx_\beta t'}$ l'un de ces systèmes d'équations linéaires basé sur $t \approx_\beta t'$ et de taille bornée.

5.7.5 Borner les coefficients de certaines attaques.

Dans cette sous section, nous allons nous appuyer sur l'existence de tuples sur β et t de taille bornée par $\|t\|^+$ et de systèmes d'équations linéaires basés sur $t \approx_\beta t'$ de taille bornée par $\|t, t'\|^+$, pour construire des attaques à coefficients (entiers) bornés polynomialement par la taille du protocole. Nous montrerons ainsi que les règles le l'intrus DH admettent effectivement des attaques à coefficients polynomiaux, ce qui nous permettra, dans la section suivante, de donner un algorithme NP pour le problème de l'insécurité face à cet intrus. Pour cela, nous commencerons par borner les coefficients entiers présents dans une seule application d'une règle d'intrus (lemme 5.7.5.2), puis nous itérerons ce résultat pour borner les coefficients entiers présents dans une dérivation (proposition 5.7.5.3). Ceci nous permettra enfin de montrer l'existence d'attaques à coefficients bornes (théorème 5.7.5.4) dès lors, bien sûr, que le protocole admet au moins une attaque (non bornée).

Cependant, pour faciliter les preuves, nous avons tout d'abord besoin d'un petit lemme technique.

Lemme 5.7.5.1 *Lemme technique.*

Soit β une valuation, et soient $t_0, .., t_n$ des messages ou produits ouverts tels que pour tout i, $\beta(t_i) = \ulcorner\beta(t_i)\urcorner$. De plus, soient $z_1, .., z_n \in Z$ des variables entières n'apparaissant pas dans les $\{t_i\}_{i=1..n}$. Enfin, posons $t = Exp(t_0, t_1^{z_1} \cdot .. \cdot t_n^{z_n})$. Alors la taille de chaque expression linéaire apparaissant dans t est bornée par $\max\{|e|_{dag} \,|\, e \in \mathcal{L}_{exp}(t_0, .., t_n)\} + n$.

Preuve. Remarquons tout d'abord que grâce au lemme 5.7.3.1, on a $t_i^\beta = t_i$ pour tout $i \in \{0, .., n\}$. Posons $M = t_1^{z_1} \cdots t_n^{z_n}$. On distingue alors plusieurs cas :

- Si $\beta(t_0) = \ulcorner\beta(t_0)\urcorner \neq Exp(\cdot, \cdot)$, alors la borne est assez évidente.

- Si $\beta(t_0) = \ulcorner\beta(t_0)\urcorner = Exp(u', M')$, alors nécessairement $t_0 = Exp(u'', M'')$ avec $\beta(u'') = u'$ et $\beta(M'') = M'$. Si de plus, $t = u''$, alors on obtient la borne directement. Sinon, on a $t = Exp(u'', (M'' \bullet M)^\beta)$ et $(M'' \bullet M)^\beta \neq 1$. Notons $M'' = t''^{e''_1}_1 \cdots t''^{e''_{n''}}_{n''}$. on sait déjà que $\ulcorner\beta(t''_i)\urcorner = \beta(t''_i)$ pour chaque i, et que $\beta(t''_i) \neq \beta(t''_j)$ pour tous $i$et j tels que $i \neq j$. Soient $C_1, .., C_l$ les classes d'équivalence définies comme lors de la preuve de la proposition 5.7.2.1. On a alors $t = \Pi_{j \notin J \cup \{1\}} (s^\beta_{C_j})^{e_{C_j}}$. Ainsi, comme $\ulcorner\beta(t''_i)\urcorner = \beta(t''_i)$, et donc comme $t''^\beta_i = t''_i$ et $t_i^\beta = t_i$, on sait que pour tout j, il existe i tel que soit $s^\beta_{C_j} = t''_i$ soit $s^\beta_{C_j} = t_i$. De plus, il existe au plus un t''_i dans chaque classe C_j. En conséquence, la taille de e_{C_j} est nécessairement bornée comme annoncé dans le lemme. Enfin, comme tous les sous termes de t sont aussi sous terme d'un t_i, la taille des expressions linéaires présentes dans t est bien bornée comme annoncé. □

Dans les preuves suivantes, nous aurons besoin de considérer des extensions de valuations. On dit qu'une valuation $\beta' : Z' \to \mathbb{Z}$ est une *extension* de la valuation $\beta : Z \to \mathbb{Z}$ quand $Z \subseteq Z'$ et $\beta'(z) = \beta(z)$ pour tout $z \in Z$. De plus, comme β et β' coïncident sur Z, nous appellerons souvent β une extension de β, par abus de notation.

Le lemme suivant va nous permettre de réduire la taille des coefficients entiers nécessaires à l'application d'une règle d'intrus.

Lemme 5.7.5.2 *Taille des coefficients nécessaires à l'application d'une règle d'intrus.*

Soit β une valuation, soient $t_1, .., t_n$ des messages ouverts tels que $\beta(t_i) = \ulcorner\beta(t_i\urcorner$ pour tout i, et soit s un message normalisé (non ouvert !). De plus, soit $L \in L_{DY} \cup L_{DHd} \cup L_{DHc}$ une règle d'oracle telle que $L = \ulcorner\beta(t_1)\urcorner, .., \ulcorner\beta(t_1)\urcorner \to s$. Alors il existe un message ouvert t, un système d'équations linéaires $\mathcal{E}$, et une extension β de β, tels que :

1. $\beta(t) = s$,
2. $\beta \models \mathcal{E}$,
3. $\ulcorner\beta'(t_1)\urcorner, .., \ulcorner\beta'(t_1)\urcorner \to s$ *pour tout* β' *tel que* $\beta' \models \mathcal{E}$,
4. $\max\{|e|_{dag} \,|\, e \in \mathcal{L}_{exp}(t)\} \leq \max\{|e|_{dag} \,|\, e \in \mathcal{L}_{exp}(t_1, .., t_n)\} + n$,
5. *et la taille de* $\mathcal{E}$ *est bornée par un polynôme en* $\|t_1, .., t_n\|^+$.

Preuve. Nous allons examiner les différentes types de règles.

- Si $L \in L_{p1}$, on a nécessairement $n = 1$, $\ulcorner\beta(t_1)\urcorner = \langle a, b\rangle$, $s = a$, $t_1 = \langle a', b'\rangle$ avec a, b deux messages normalisés, a', b' deux messages ouverts, et $\beta(a') = a$ et $\beta(b') = b$. On peut alors poser $t = a'$, $\mathcal{E} = \emptyset$, et β non étendue, ce qui vérifie naturellement les points 1 à 5. De la même manière, on construit t, $\mathcal{E}$, et on étend β pour satisfaire ces cinq points quand $L \in L_{p2}$, $L \in L_{ad}$, ou $L \in L_c$.

- Si $L \in L_{sd}$, on a nécessairement $n = 2$, $\ulcorner\beta(t_1)\urcorner = \{a\}^s_b$ et $\ulcorner\beta(t_2)\urcorner = b$ avec a et b deux messages normalisés. Ainsi, on a $t_1 = \{a'\}^s_{b'}$ et $t_2 = b''$ avec a', b' et b'' trois messages ouverts tels

que $\beta(a') = a$ et $\beta(b') = \beta(b'') = b$. On peut alors poser $t = a'$, $\mathcal{E} = \mathcal{E}^{=_\beta}_{b',b''}$, et β n'est pas étendue. Grâce à lemme 5.7.1.4, on se rend alors aisément compte que les points 1 à 5 sont vérifiés.

- Si $L \in L_{DHd} \cup L_{DHc}$, on a nécessairement $s = \ulcorner Exp(\beta(t_1), \beta(t_2)^{a_2} \cdot .. \cdot \beta(t_n)^{a_n} \urcorner$ avec $a_i \in \mathbb{Z}$ pour tout i. Posons $t = Exp(t_1, t_2^{z_2} \cdot .. \cdot t_n^{z_n})^\beta$, avec z_i de nouvelles variables entières, et $\mathcal{E} = \mathcal{E}^\beta_t$. De plus, on étend β de façon à avoir $\beta(z_i) = a_i$ pour tout i. Alors grâce au lemme 5.7.2.1, on obtient $s = \ulcorner \beta(Exp(t_1, t_2^{z_2} \cdot .. \cdot t_n^{z_n})) \urcorner = \beta(t)$, $\beta \models \mathcal{E}$, et $\ulcorner \beta'(t) \urcorner = \ulcorner \beta'(Exp(t_1, t_2^{z_2} \cdot .. \cdot t_n^{z_n})) \urcorner = \ulcorner Exp(\ulcorner \beta'(t_1) \urcorner, \ulcorner \beta'(t_2) \urcorner^{\beta'(z_2)} \cdot .. \cdot \ulcorner \beta'(t_2) \urcorner^{\beta'(z_2)}) \urcorner$ pour toute valuation β' telle que $\beta' \models \mathcal{E}$, ce qui prouve les points 1 à 3. De plus, le lemme 5.7.5.1 prouve le point 4, et la proposition 5.7.4.2 prouve le point 5. □

Nous allons maintenant étendre ce lemme au dérivations tout entières, de manière à suivre les messages créés par l'intrus lors d'une attaque. En effet, étant donnée une dérivation (bien formée) de $\beta(E)$ vers $\beta(t)$, nous allons construire un système d'équations représentant toutes les contraintes devant être satisfaites par une valuation β' pour que $\beta'(t)$ soit toujours dérivable par l'intrus à partir de $\beta'(E)$. Ce système d'équations devra être de taille (polynomialement) bornée, car ainsi il admettra au moins une solution de taille (polynomialement) bornée.

Proposition 5.7.5.3 *Taille des coefficients dans une dérivation.*

Soient $t, t_1, .., t_n$ des messages ouverts tels $\ulcorner \beta(t) \urcorner \in forge(\ulcorner \beta(t_1) \urcorner, .., \ulcorner \beta(t_n) \urcorner)$, avec β une valuation. Alors il existe une extension de β et un système d'équations linéaires $\mathcal{E}$ tels que :

1. *$\beta \models \mathcal{E}$,*
2. *$\ulcorner \beta'(t) \urcorner \in forge(\ulcorner \beta'(t_1) \urcorner, .., \ulcorner \beta'(t_n) \urcorner)$ pour toute valuation β' telle que $\beta' \models \mathcal{E}$,*
3. *et la taille de $\mathcal{E}$ est bornée par un polynôme en $\|t, t_1, .., t_n\|^+$.*

Preuve. Soient β, t, et $t_1, .., t_n$ définis comme ci-dessus. Posons $E = \{\ulcorner \beta(t_1) \urcorner, \ldots, \ulcorner \beta(t_n) \urcorner\}$, et soit :

$$D \;=\; E \rightarrow_{L_1} E, s_1 \rightarrow_{L_2} \ldots \rightarrow_{L_l} E, s_1, .., s_l$$

une dérivation bien formée partant de E et de but $\ulcorner \beta(t) \urcorner$. Nous savons déjà que la longueur de D (i.e. l) est bornée par un polynôme en $\left|\ulcorner \beta(t) \urcorner, \ulcorner \beta(t_1) \urcorner, \ldots, \ulcorner \beta(t_n) \urcorner\right|_{dag}$. De plus, grâce au lemme 5.7.3.2 (point 2), on peut borner $\left|\ulcorner \beta(t) \urcorner, \ulcorner \beta(t_1) \urcorner, \ldots, \ulcorner \beta(t_n) \urcorner\right|_{dag}$ par un polynôme en $|t, t_1, \ldots, t_n|_{dag}$, et donc également par un polynôme en $\|t, t_1, \ldots, t_n\|^+$. Par construction, on a $s_l = \ulcorner \beta(t) \urcorner$ et pour tout i, s_i est un message normalisé. De plus, comme D est une dérivation bien formée, on a $s_i \in STermes(\ulcorner \beta(t) \urcorner, \ulcorner \beta(t_1) \urcorner, .., \ulcorner \beta(t_n) \urcorner)$ pour tout i.

Soit $(t^\beta, \mathcal{E}^\beta_t)$ un tuple sur β et t, et soit t_i^β un message générique basé sur (β, t_i), pour tout i. On a $\beta(t_i^\beta) = \ulcorner \beta(t_i) \urcorner$ pour tout i, et donc $E = \{\beta(t_1^\beta), .., \beta(t_n^\beta)\}$. Ainsi, on peut appliquer le lemme 5.7.5.2 à la première règle de D. On obtient un message ouvert r_1, un système d'équations linéaires $\mathcal{E}_1$, et une extension de β tels que $\beta(r_1) = s_1$, $\beta \models \mathcal{E}_1$, et :

$$\ulcorner \beta'(t_1^\beta) \urcorner, .., \ulcorner \beta'(t_n^\beta) \urcorner \rightarrow_{L_1} \ulcorner \beta'(t_1^\beta) \urcorner, .., \ulcorner \beta'(t_n^\beta) \urcorner, \ulcorner \beta'(r_1) \urcorner \text{ pour tout } \beta' \models \mathcal{E}_1$$

Il est important de remarquer que $\beta(t_i^\beta) = \ulcorner \beta(t_i^\beta) \urcorner$ pour tout i, et que $\beta(r_1) = \ulcorner \beta(r_1) \urcorner = s_1$. On peut ainsi appliquer le lemme 5.7.5.2 inductivement sur toutes les règles de D. Pour tout $j \in \{1, .., l\}$, on obtient alors s'_j, $\mathcal{E}_j$, et une extension de β tels que $\beta(r_j) = s_j$, $\beta \models \mathcal{E}_j$, et

$$E', \ulcorner \beta'(r_1) \urcorner, \ldots, \ulcorner \beta'(r_{j-1}) \urcorner \rightarrow_{L_j} E', \ulcorner \beta'(r_1) \urcorner, \ldots, \ulcorner \beta'(r_j) \urcorner \quad \text{pour tout } \beta' \models \mathcal{E}_j$$
$$\text{avec } E' = \ulcorner \beta'(t_1^\beta) \urcorner, \ldots, \ulcorner \beta'(t_n^\beta) \urcorner$$

En conséquence, on a :

$$\beta \models \bigcup_{j=1}^{l} \mathcal{E}_j$$
$$\text{et } \ulcorner\beta'(r_l)\urcorner \in forge(\ulcorner\beta'(t_1^\beta)\urcorner, .., \ulcorner\beta'(t_n^\beta)\urcorner) \text{ pour tout } \beta' \models \bigcup_{j=1}^{l} \mathcal{E}_j$$

Nous allons maintenant construire $\mathcal{E}$ vérifiant les deux premiers points du lemme. Nous vérifierons ensuite qu'il est bien de taille bornée. Pour tout β' tel que $\beta' \models \bigcup_{i=1}^{n} \mathcal{E}_{t_i}^{\beta}$, on a $\ulcorner\beta'(t_i)\urcorner = \ulcorner\beta'(t_i^\beta)\urcorner$. De plus, on sait que $\beta(t^\beta) = \ulcorner\beta(t)\urcorner = s_l = \beta(r_l)$. Ainsi, $t^\beta =_\beta r_l$. On peut alors appliquer le lemme 5.7.1.4 : soit $\mathcal{E}_{t^\beta, r_l}^{=_\beta}$ le système d'équations linéaires basé sur $t^\beta =_\beta r_l$ déterminé (de manière unique) par ce lemme. A présent, pour toute valuation β' telle que $\beta' \models \mathcal{E}_t^\beta \cup \mathcal{E}_{t^\beta, r_l}^{=_\beta}$, on obtient $\ulcorner\beta'(t)\urcorner = \ulcorner\beta'(t^\beta)\urcorner = \ulcorner\beta'(r_l)\urcorner$. On pose donc :

$$\mathcal{E} = \bigcup_{i=1}^{n} \mathcal{E}_{t_i}^{\beta} \cup \mathcal{E}_t^\beta \cup \bigcup_{j=1}^{l} \mathcal{E}_j \cup \mathcal{E}_{t^\beta, r_l}^{=_\beta}.$$

On a naturellement $\beta \models \mathcal{E}$ et $\ulcorner\beta'(t)\urcorner \in forge(\ulcorner\beta'(t_1)\urcorner, .., \ulcorner\beta'(t_n)\urcorner)$ pour toute valuation β' telle que $\beta' \models \mathcal{E}$.

Il ne nous reste donc plus qu'à montrer que la taille du système d'équations linéaires $\mathcal{E}$ est bornée par un polynôme en $\|t_1, \ldots, t_n, t\|^+$. Grâce au lemme 5.7.5.2, on sait déjà chaque $\mathcal{E}_j$ est de taille bornée par un polynôme en $\|t_1, \ldots, t_n, r_1, \ldots, r_{j-1}\|^+$. De plus, on a $\beta(r_j) = s_j \in STermes(\ulcorner\beta(t_1)\urcorner, \ldots, \ulcorner\beta(t_n)\urcorner, \ulcorner\beta(t)\urcorner)$. Ainsi, le lemme 5.7.3.2 nous permet de borner polynomialement $|r_j|_{dag}$ en fonction de $|t_1, \ldots, t_n, t|_{dag}$. On obtient donc que pour tout j, $|r_j|_{dag}^+$ est borné par un polynôme en $|t_1, \ldots, t_n, t|_{dag}$, puisque $|..|_{dag}^+ \leq 2.|..|_{dag}$. Grâce au lemme 5.7.5.2, on en déduit que $\max\{|e|_{dag} \mid e \in \mathcal{L}_{exp}(r_j)\} \leq \max\{|e|_{dag} \mid e \in \mathcal{L}_{exp}(t_1^\beta, .., t_n^\beta)\} + n \times (j-1)$. De plus, on a $j \leq l$ et on sait déjà que l est borné par un polynôme en $\|t_1, \ldots, t_n, t\|^+$. En outre, grâce au lemme 5.7.3.5 il existe i tel que :

$$\max\{|e|_{dag} \mid e \in \mathcal{L}_{exp}(t_1^\beta, .., t_n^\beta)\} \leq \left|t_i^\beta\right|_{exp} \leq \|t_i\|^{+3}$$

Ainsi, il existe un polynôme p tel que $\|r_j\|^+$ soit borné par $p(\|t_1, \ldots, t_n, t\|^+)$. En conséquence, $\|t_1, \ldots, t_n, t, r_1, \ldots, r_j\|^+$ est borné par $p(\|t_1, \ldots, t_n, t\|^+) \times (p'(\|t_1, \ldots, t_n, t\|^+) + 1)$ avec p' le polynôme bornant l en fonction de $\|t_1, \ldots, t_n, t\|^+$. Grâce au lemme 5.7.5.2, ceci implique que $\mathcal{E}_j$ est de taille bornée polynomialement par $\|t_1, \ldots, t_n, t\|^+$. On peut alors utiliser la proposition 5.7.4.2 et le lemme 5.7.1.4 pour en déduire que la taille de $\mathcal{E}$ est bornée par un polynôme en $\|t_1, \ldots, t_n, t\|^+$. □

Il ne nous reste plus qu'à utiliser la proposition 5.7.5.3 précédente et le théorème 5.6.0.8 pour prouver le théorème suivant, but de cette section et des sections précédentes : tout protocole non sûr face à l'intrus DH admet au moins une attaque de taille (totale, i.e. DAG + coefs) bornée par un polynôme en la taille du protocole (face à l'intrus DH).

Théorème 5.7.5.4 *Attaques minimales à coefficients bornés.*

Pour tout protocole $P = (\{R_\iota \Rightarrow S_\iota, \iota \in \mathcal{I}\}, <_\mathcal{I}, S_0)$ admettant au moins une attaque face à l'intrus DH, il existe une attaque (π, σ) sur P avec $\|\sigma\|^+$ borné par un polynôme p en $\|P\|$, face à ce même intrus.

PREUVE. Soit (π, σ) une attaque minimale sur P. De plus, soit σ^Z la substitution issue de σ où l'on a remplacé tous les coefficients entiers par de nouvelles variables entières (une variable par position dans σ), et soit β la valuation attribuant à chaque variable le coefficient entier d'origine de σ. On a donc, par construction, $\sigma(x) = \beta(\sigma^Z(x))$ pour tout $x \in Var(P)$. On remarque que grâce au théorème 5.6.0.8, $\left|\sigma^Z\right|_{dag} = |\sigma|_{dag}$ est bornée par un polynôme en $|P|_{dag}$. De plus, comme tous les coefficients produits de σ^Z sont des variables, on a $\left|\sigma^Z\right|_{exp} \leq \left|\sigma^Z\right|_{dag}^2$. Ainsi, $\left\|\sigma^Z\right\|^+$ est bornée par un polynôme en $\|P\|^+$.

Soit $n = \#\pi$ le nombre de pas de cette attaque, et soient $R_i \Rightarrow S_i$, $i \in \{1, .., n\}$ les pas de cette attaque, en voyant π comme une bijection de $\mathcal{J} \subseteq \mathcal{I}$ sur $\{1, .., n\}$. On suppose sans perte de généralité que S_0 est un seul message au lieu d'un ensemble de messages. Ce n'est pas restrictif, car on peut au pire représenter $S_0 = \{a_1, .., a_p\}$ par $\langle a_1, \langle ..., a_n\rangle\rangle$. Pour respecter l'harmonie de ces définitions, on pose $R_{n+1} = Secret$. Comme (π, σ) est une attaque, on a :

$$\text{Pour tout } i \in \{1, .., n+1\},\ \ulcorner\beta(R_i\sigma^Z)\urcorner \in forge(\ulcorner\beta(S_0\sigma^Z)\urcorner, .., \ulcorner\beta(S_{i-1}\sigma^Z)\urcorner$$

On peut alors appliquer la proposition 5.7.5.3 pour chaque $i \in \{1, .., n+1\}$, en prenant soin de définir à chaque fois des extensions indépendantes de β, i.e. en choisissant à chaque i de nouvelles variables non choisies aux autres $i' \neq i$. On obtient ainsi une extension de β et un système d'équations linéaires $\mathcal{E}_i$ tels que :

- $\beta \models \mathcal{E}_i$,
- $\ulcorner\beta'(R_i\sigma^Z)\urcorner \in forge(\ulcorner\beta'(S_0\sigma^Z)\urcorner, .., \ulcorner\beta'(S_{i-1}\sigma^Z)\urcorner$ pour toute valuation β' telle que $\beta' \models \mathcal{E}_i$,
- et la taille de $\mathcal{E}_i$ est bornée par un polynôme en $\left\|S_0\sigma^Z, \ldots, S_k\sigma^Z, R_1\sigma^Z, \ldots, R_{k+1}\sigma^Z\right\|^+$, elle-même bornée par un polynôme en $\|P\|^+$.

En rassemblant tout ceci, on obtient que $\beta \models \mathcal{E}$ avec $\mathcal{E} = \bigcup_{i\in\{1,..,n\}} \mathcal{E}_i$, et donc $\mathcal{E}$ admet au moins une solution. On obtient également que pour toute valuation β' telle que $\beta' \models \mathcal{E}$, $(\pi, \beta'(\sigma^Z))$ est une attaque sur P. Or un tel système d'équations linéaires solvable admet nécessairement au moins une solution dont la représentation binaire peut être bornée par un polynôme en la taille du système (voir [13]). On sait donc qu'il existe β', solution de $\mathcal{E}$ i.e. $\beta' \models \mathcal{E}$ et $(\pi, \beta'(\sigma^Z))$ est une attaque sur P, dont la représentation (binaire) est bornée par $\|P\|^+$. En posant la substitution $\sigma' = \beta'(\sigma^Z)$, on sait que (π, σ') est une attaque sur P. De plus, σ et σ' ne diffèrent que par leurs coefficients entiers (des produits), ce qui nous donne (avec la borne sur la taille de β') une borne polynomiale sur $|\sigma'|_{exp}$ en fonction de $\|P\|^+$, et donc une borne polynomiale sur $\|P\|$. Il nous reste à remarquer que $|\sigma'|_{dag} = |\sigma|_{dag}$, ce qui termine de prouver le théorème : $\|\sigma'\|^+$ est bornée par un polynôme en $\|P\|$. □

5.8 Algorithme NP pour le problème de l'insécurité

Nous avons maintenant à notre disposition deux types de bornes : d'une part, il existe (à partir du moment où le protocole n'est pas sûr) au moins une attaque (π, σ) où tous les messages transmis sont bornés linéairement par la taille de la spécification de protocole, et d'autre part dans une attaque il existe au moins une dérivation permettant de créer le message $R_i\sigma$ à partir de $S_0\sigma, .., S_{i-1}\sigma$ qui soit linéairement bornée par la taille de $R_i\sigma, S_0\sigma, .., S_{i-1}\sigma$ et donc par la taille de la spécification de protocole. Il nous suffit donc de décrire un algorithme (NP) cherchant une attaque vérifiant ces bornes. Ainsi, si le protocole est n'est pas sûr on trouvera nécessairement une attaque, et s'il est sûr on pourra certifier qu'il n'existe pas d'attaque. Le détail de l'algorithme est donné dans la Figure 5.4.

1. Choisir un ordre d'exécution π sur P.

2. Choisir une substitution close σ telle que pour tout $x \in Var$,
   - $|\sigma(x)|_{dag} \leq 8.\|P\| + 1$ (c.f. thm. 5.6.0.8) et
   - $|\sigma|_{exp} \leq p(\|P\|)$ (avec p obtenu au théorème 5.7.5.4)

5. Vérifier que pour tout $i \in \{1, .., \#\pi\}$, $\ulcorner R_i\sigma \urcorner \in forge_{DH}(\ulcorner S_0\sigma, .., S_{i-1}\sigma \urcorner)$ et $Secret \in forge_{DH}(\ulcorner S_0\sigma, .., S_{\#\pi}\sigma \urcorner))$.

6. Si toutes ces vérifications sont réussies, alors répondre VRAI.

FIG. 5.4: Algorithme NP pour l'insécurité de protocoles avec oracle (exponentielle).

Nous savons (par construction et grâce aux bornes données aux sections précédentes) que dans le cas de l'intrus DH, cet algorithme est correct et complet. En outre, toujours pour cet intrus, il est NP. En effet :

- L'ordre d'exécution π peut être choisi en temps polynomial, car ce n'est qu'une fonction injective partielle de $\mathcal{J} \subset \mathcal{I}$ dans $\{1, .., \#\mathcal{J}\}$ avec $\#\mathcal{I} \leq n$, et $n = \|P\|$.
- Une représentation DAG de la substitution σ peut être choisie en temps polynomial en n, puisque pour tout $x \in Var$, $|\sigma(x)|_{dag} \leq 8.n + 1$.

De plus, on a besoin de savoir que le problème d'appartenance à $Derive(DH) = \{ (t, E) \,|\, t \in forge_{DH}(E) \}$ est polynomial. En effet :

Proposition 5.8.0.5 *Le problème $Derive(DH)$ est polynomial.*
Pour tout ensemble de messages E et tout message t, on peut décider si $t \in forge_{DH}(E)$ en temps polynomial en $|E, t|_{dag}$.

PREUVE. Il nous suffit de réutiliser la preuve correspondante pour les intrus xor ou préfixe, i.e. la preuve de la proposition 4.3.0.3. En effet, cette preuve ne nécessite que l'existence de dérivation bien formée de taille bornée (ici par $10.n + 1$, c.f. corollaire 5.6.0.9) et la possibilité de décider (en temps polynomial) si $F \rightarrow_{DH} F, u$ pour F un ensemble messages normalisés et u un message normalisé (c'est l'appartenance à $Application(DH)$, c.f. sous section 5.4.3). Comme l'intrus DH définit un ensemble de règles d'oracle, on a l'existence de dérivations bien formées. □

Ainsi, on a bien un algorithme NP pour décider de l'insécurité de protocoles cryptographiques face à l'intrus de DH. De plus, en examinant cet algorithme on se rend compte que l'on n'a besoin "que" de deux propriétés de d'intrus pour qu'il fonctionne avec n'importe quel ensemble de règles d'oracle. En effet, on doit d'une part assurer qu'il existe un polynôme p (utilisé au point 1 de l'algorithme) bornant les coefficients entiers nécessaires à la création d'une attaque, et d'autre part on doit être capable de vérifier en temps polynomial qu'une règle d'intrus (ou d'oracle) donnée peut être appliquée à un ensemble de messages donnés. Modulo ces deux propriétés assez fortes, on peut utiliser cet algorithme sur n'importe quel ensemble de règles d'oracle, et il sera toujours NP. Une propriété plus faible sur la vérification des applications de règles d'oracle donnerait certes un algorithme valable, mais d'une complexité supérieure. En résumé, on a :

Théorème 5.8.0.6 *L'insécurité modulo des règles d'oracle est NP.*

Soit un intrus disposant de règles d'oracle $\mathcal{L}$ (selon la définition 5.2.0.7), basé sur les opérateurs standards, le produit, et l'exponentielle, et tel que :

- *Application($\mathcal{L}$) est décidable en temps polynomial en $|E,t|_{dag}$.*
- *Pour toute attaque minimale (π,σ) sur un protocole P, il existe une attaque (π,σ') telle que $|\sigma'|_{exp}$ (resp. $|\sigma'|_{dag}$) soit bornée polynomialement par $\|P\|$ (resp. $|\sigma|_{dag}$).*

Alors le problème de l'insécurité de protocoles cryptographiques, basé sur ces mêmes opérateurs et face à cet intrus, est NP.

De plus, pour l'intrus DH on a déjà montré que ces conditions sont remplies. Ainsi :

Corollaire 5.8.0.7 *L'insécurité modulo DH est NP-complète.*

Le problème de l'insécurité de protocoles cryptographiques (basé sur les opérateurs standards, produit, et exponentielle) face à l'intrus DH est NP-complet.

En effet, les codages de problèmes 3-SAT du chapitre 3 sont toujours valables.

En outre, on peut remarquer que la notion de dérivation bien formée utilisée ici permet de résoudre assez simplement un problème plus simple que celui de l'insécurité : le problème de dérivation, i.e. décider si $t \in forge(E)$ étant donnés E et t. En effet, supposons que E et t soient normalisés (s'ils ne le sont pas, on peut les normaliser en temps $\|E,t\|$. On sait que si $t \in forge(E)$, alors il existe une dérivation bien formée partant de E et de but t. De plus, une telle dérivation n'utilise que des termes intermédiaires sous termes de E ou t. Ainsi, même si $forge(E)$ est infini, on peut construire la clôture (en un pas) de E par la relation $\succ$ suivante : $E \succ E, a$ ssi a est un sous terme de E ou t, et il existe une règle L telle que $E \rightarrow E, a$. En effet, on n'a qu'un nombre polynomial en $|E,t|_{dag}$ de termes a à tester (puisque a est sous terme de E ou t), et grâce à la proposition 5.4.3.1 on peut décider s'il existe une telle règle L en temps polynomial en $\|E,t\|$. Ainsi, on calcule $\bar{E} = \bigcup_{E \succ E'} E'$ en temps polynomial en $\|E,t\|$. Comme les dérivations bien formées sont de longueur bornée par $|E,t|_{dag}$, on calculer l'ensemble E' des messages normalisés accessibles à partir de E par une dérivation bien formée en temps polynomial en $\|E,t\|$. Ceci résout le problème de dérivation, puisqu'il ne nous reste plus qu'à tester si $t \in E \cup E'$. On a donc la proposition suivante :

Proposition 5.8.0.8 *Le problème de dérivation (décider si $t \in forge(E)$) est polynomial en $\|E,t\|$.*

5.9 Conclusion

Nous avons présenté ici un ensemble de bornes sur les tailles de certaines attaques de manière à construire un algorithme NP résolvant le problème de l'insécurité face à l'intrus DH. La structure de preuve suit toujours l'idée relativement générale initiée pour l'intrus DY (caractériser les pré termes des substitutions pour en borner la taille DAG). Cependant, elle est devenue notablement plus complexe que pour les intrus DY ou xor en raison notamment des propriétés particulières de l'opérateur produit et des règles d'intrus adaptées à cet opérateur et à l'exponentielle. En effet, d'une part à cause de la présence de coefficients entiers, la normalisation peut énormément changer la structure des messages. Et d'autre part, l'intrus peut choisir n'importe quels coefficients entiers à la création d'une exponentielle, mais l'attaque qu'il construit doit satisfaire un ensemble de contraintes sur les coefficients choisis. Ce sont ces contraintes que l'on a représenté (et géré) par un ensemble d'équations linéaires. Malgré tout cela, on a encore une

fois des bornes polynomiales sur les tailles des attaques à considérer (modulo une représentation binaires des coefficients entiers, i.e. les coefficients eux-mêmes peuvent être exponentiels), d'où l'algorithme NP.

6

Insécurité avec commutation de clefs

Sommaire

Dans ce chapitre, nous allons ajouter des propriétés de commutation de clefs à certains types de protocoles cryptographiques. Ceci permet par exemple de représenter une encryption RSA utilisant un module commun, ou toute autre type d'encryption vérifiant cette propriété, que ce soit voulu ou non.

Nous commencerons par regarder le cas des protocoles Ping-Pong (avec sessions infinies) en donnant un résultat de NP-complétude du problème de l'insécurité dans ces protocoles face à la commutation de clefs. Pour mémoire, sans commutation de clefs ce problème est connu pour être polynomial. Ensuite, nous expliquerons comment modifier les preuves du chapitre 5 précédent pour obtenir une procédure de décision en temps NP de ce problème (avec commutation de clefs) dans le modèle de protocoles par rôles.

La différence principale entre l'intrus EC que l'on veut étudier ici, et l'intrus DH du chapitre précédent, est que dans notre cas (encryption publique/privée) l'intrus ne doit pas être capable de calculer l'inverse d'un exposant. Typiquement, étant donnée une clef publique concrète (n, e) et un message encrypté $c = m^e \bmod n$, l'intrus ne peut pas calculer la clef privée (concrète) d permettant de calculer $m = c^d \bmod n$.

6.1 Sessions infinies : Protocoles Ping-Pong

Mais qu'est-ce qu'un protocole Ping-Pong ? Le modèle de protocole Ping-Pong, présenté initialement par Dolev-Yao en 1983, modélise deux principaux (honnêtes) se renvoyant un message

en appliquant à chaque fois différents opérateurs d'encryption ou de décryptage. Il existe plusieurs versions de ces protocoles. Nous nous appuierons sur la version de D. Dolev, S. Even, et R.M. Karp (c.f. [42]), car elle est un peu plus générale que les deux versions d'origine de Dolev-Yao. Pour lier cette définition de protocoles Ping-Pong au modèle de protocoles par rôles que nous utilisons ici, nous pouvons dire qu'un protocole est un protocole Ping-Pong quand :

- il n'utilise aucun couple $\langle .., .. \rangle$ (ni aucun opérateur "spécial" xor, exponentielle, ou produit, ou encryption commutative, du moins dans la définition d'origine) ;
- toutes les clefs d'encryption sont atomiques et connues lors de la spécification du protocole. On ne peut donc pas utiliser de variable en position clef.
- les principaux n'effectuent aucune vérification sur le message reçu. Ils se contentent d'ajouter des opérateurs d'encryption et de décryptage (en fait, de l'encryption à clef inverse) et de renvoyer le résultat même s'il n'a aucun sens.

Le modèle de protocole Ping-Pong utilise ainsi un ensemble d'opérateurs $E_X(..)$ (encryption avec la clef publique du principal X) et $D_X(..)$ (décryptage avec la clef privée du principal X). Dans notre modèle, nous pouvons modéliser ceci avec $\{..\}^p_{K_X}$ et $\{..\}^p_{K_X{}^*}$. Néanmoins, les protocoles Ping-Pong utilisent implicitement la simplification des opérateurs E_X et D_X, ce que nous n'avons pas directement dans le modèle par rôles : pour tout message M et tout principal X, $E_X(D_X(M)) = D_X(E_X(M)) = M$. En outre, pour tenter de compenser la perte de l'opérateur $\langle .., .. \rangle$, deux nouveaux types d'opérateurs sont introduits. Il s'agit de i_X et d_X, pour tout principal X. Informellement, $i_X(M)$ représente l'ajout de l'atome X à la fin du message M. On peut modéliser ceci par $\langle M, X \rangle$, en remarquant que X est un atome fixé par la spécification du protocole (on ne peut pas concaténer deux messages). Inversement, $d_X(M)$ représente l'élimination de l'atome X présent à la fin du message M (et la vérification qu'il y est). On a donc $d_X(i_X(M)) = M$, pour tout message M et principal X, mais $i_X(d_X(..))$ ne se simplifie pas. Par extension, on dispose d'un opérateur d générique, éliminant n'importe quel atome : pour tout message M et principal X, $d(i_X(M)) = M$.

6.1.1 Syntaxe

Nous avons vu assez informellement que les protocoles Ping-Pong sont structurellement plus simples que les protocoles du modèle par rôles. Il sont d'ailleurs également plus simples à vérifier (procédure de décision en temps polynomial). Cependant, ils utilisent un nombre infini de sessions, ce qui nous interdit de les spécifier directement comme restriction du modèle par rôles.

Nous allons définir un nouveau modèle de protocole, dit modèle Ping-Pong. L'idée générale suit le modèle donné par [42], et utilise les notions classiques d'alphabet et de mot.

Définition 6.1.1.1 *Protocole Ping-Pong.*

Soit $A = \{E_1, D_1, .., E_n, D_n, i_1, d_1, .., i_k, d_k, d\}$ un alphabet avec $n, k \in \mathbb{N}^*$ quelconques. Alors un protocole Ping-Pong est un ensemble fini $\{\alpha_i \mid i \in \{0, .., p\}\} \subseteq A^+$, i.e. un ensemble de $p+1$ mots non vides sur A, avec $p \in \mathbb{N}$.

Exit donc les notions de principal, de secret, de variable et de message que l'on a utilisé jusqu'à présent. Nous ne définissons aucun principal, et en fait aucun rôle, car un protocole Ping-Pong peut être attaqué avec n'importe quel ordre sur ses "pas de protocole" α_i. Un protocole Ping-Pong peut être vu comme un protocole disposant de p principaux exécutant chacun un seul pas. Un "pas de protocole" α_i, pour $i \in \{1, ..p\}$, peut être vu (intuitivement) comme un pas $x_i \Rightarrow \alpha_i(x)$, en remplaçant les opérateurs E_j, D_j, i_k, d_k par leurs équivalents $\{..\}^p_{K_j}$, $\{..\}^p_{K_j{}^*}$, $\langle .., A_k \rangle$ et un opérateur de décomposition de couple respectivement. Modulo quelques

réécritures de ce pas, on peut se passer de l'opérateur de décomposition de couple (par exemple, $x_i \longrightarrow d_2(x_i)$ peut être vu comme $\langle x'_i, A_2\rangle \Rightarrow x'_i$). Le mot α_0 a ici un rôle particulier. Il représente d'une certaine manière les connaissances initiales de l'intrus (ou le service au ping-pong). Du point de vue du modèle par rôles, ce serait avoir $\alpha_0(Secret)$ dans les connaissances initiales de l'intrus. Le but de l'intrus est donc d'empiler suffisamment de mots α_i sur $\alpha_0(Secret)$ pour que cela se simplifie entièrement et donne $Secret$.

6.1.2 Notion d'attaque

Pour que l'intrus puisse calculer $Secret$ à partir de $\alpha_0(Secret)$ en utilisant autant de pas α_i que nécessaire, nous devons considérer les mots sur A modulo une équivalence $\equiv$ permettant la simplification des encryptions/décryptages E_i/D_i et des i_k/d_k. On note ϵ le mot vide.

Définition 6.1.2.1 *Équivalence de mots sur A.*

Soit $A = \{E_1, D_1, .., E_n, D_n, i_1, d_1, .., i_k, d_k, d\}$ *un alphabet. On pose* $\equiv$ *la plus petite relation* symétrique, réflexive *et* transitive *sur les mots de A telle que*

- *pour tout* $i \in \{1, .., n\}$ *ou* $j \in \{1, .., k\}$, $E_iD_i \equiv D_iE_i \equiv d_ji_j \equiv di_j \equiv \epsilon$, *et*
- *pour tous mots* β_1, β_2, γ *et* γ' *sur A, si* $\gamma \equiv \gamma'$ *alors* $\beta_1\gamma\beta_2 \equiv \beta_1\gamma'\beta_2$.

On peut à présent spécifier formellement ce qu'est une attaque sur un protocole Ping-Pong :

Définition 6.1.2.2 *Attaque sur un protocole Ping-Pong.*

Une attaque sur un protocole Ping-Pong $\{\, \alpha_i \,|\, i \in \{0, .., p\}\,\}$ *est une liste d'indices* $\{I_i\}_{i\in\{1..n\}}$, *avec* $I_i \in \{1, .., p\}$ *pour tout i, telle que* $\alpha_{I_1}\alpha_{I_2}...\alpha_{I_n}\alpha_0 \equiv \epsilon$.

Ainsi, le but de l'intrus est bien de trouver une combinaison de mots α_i permettant de "simplifier" α_0, et donc de découvrir $Secret$ si l'on voit α_0 comme la connaissance initiale $\alpha_0(Secret)$.

6.1.3 Exemple

Nous aurons à la section suivante un exemple d'utilisation des protocoles Ping-Pong avec commutation de l'encryption (codage du problème Integer Programming). En attendant, nous allons simplement illustrer les définitions précédentes de protocole Ping-Pong et d'attaque avec un exemple assez simple et sa correspondance dans le modèle par rôles. Considérons le protocole suivant, dans le modèle par rôles et avec quatre principaux A, B, C et D, et la connaissance initiale S_0 :

$$S_0 = \left\{ \left\langle \left\{ \{Secret\}^p_{K_1} \right\}^p_{K_2}, A_2 \right\rangle \right\}^p_{K_3}$$

$$\begin{array}{lcl} A & : & \{\langle x, A_2\rangle\}^p_{K_3} \longrightarrow x \\ B & : & y \longrightarrow \left\langle \{y\}^p_{K_2{}^*}, A_1 \right\rangle \\ C & : & \langle z, A_1\rangle \longrightarrow \left\{ \{z\}^p_{K_1{}^*} \right\}^p_{K_2} \\ D & : & \langle x', y'\rangle \longrightarrow x' \end{array}$$

Ce protocole se traduit assez facilement en un protocole Ping-Pong. Il suffit d'identifier les opérateurs E_1, E_2 et E_3 pour les clefs K_1, K_2 et K_3, les opérateurs D_1, D_2 et D_3 pour les clefs

${K_1}^*$, ${K_2}^*$ et ${K_3}^*$, et les opérateurs i_1, d_1, i_2, et d_2 pour les atomes A_1 et A_2. On obtient :

$$\begin{aligned}\alpha_0 &= E_3\, i_2\, E_2\, E_1\\ \alpha_1 &= d_2\, D_3\\ \alpha_2 &= i_1\, D_2\\ \alpha_3 &= E_2\, D_1\, d_1\\ \alpha_4 &= d\end{aligned}$$

Ce protocole admet naturellement une attaque selon la définition précédente. En effet, on a :

$$\alpha_4\, \alpha_2\, \alpha_3\, \alpha_2\, \alpha_1\, \alpha_0 \;=\; d\, E_2\, D_1\, d_1\, i_1\, D_2\, d_2\, D_3\, E_3\, i_2\, E_2\, E_1 \;\equiv\; \epsilon$$

6.1.4 Résultat connu : l'insécurité des Ping-Pong est polynomiale.

La complexité du problème de l'insécurité de protocoles Ping-Pong a été étudiée dès l'introduction de la modèle restreint de protocoles par D. Dolev et A.C. Yao en 1982-83. Ceux-ci ont donné une procédure de décision en temps polynomial dans [43] pour un modèle de protocoles Ping-Pong plus restreint que celui présenté ici, et avec un polynôme de degré relativement élevé. Puis ce résultat a été étendu dans [42], montrant que le problème de l'insécurité de protocoles Ping-Pong (tels qu'ils sont présentés ici) est décidable en temps $O(n^3)$, avec n la taille du protocole. Cette borne polynomiale est nettement meilleure que celle du premier papier ([43]). L'idée de la preuve est de construire un automate reconnaissant tous les mots $\alpha_{I_1}..\alpha_{I_n}\alpha_0$, pour toute liste l'indices $\{I_i\}_{i\in\{1,..,n\}}$, puis de le modifier pour qu'il reconnaisse tous les mots t obtenus à partir de $\alpha_{I_1}..\alpha_{I_n}\alpha_0$ en appliquant les simplifications $E_iD_i \longrightarrow \epsilon$, $D_iE_i \longrightarrow \epsilon$, $d_ji_j \longrightarrow \epsilon$ et $di_j \longrightarrow \epsilon$. Cette construction se fait en temps polynomial. Il ne reste alors plus qu'à tester si cet automate reconnaît ϵ.

6.2 Extension des Ping-Pong avec commutation de clefs

Dans cette section, nous allons étendre la définition précédente d'attaques sur les protocoles Ping-Pong pour permettre la commutation des opérateurs E_i et D_i. Du point de vue du modèle par rôles, cela revient à utiliser l'opérateur d'encryption commutative $\{..\}^{pc}_{K_i}$ à la place de $\{..\}^{p}_{K_i}$. Nous allons montrer que le problème de l'insécurité de protocoles Ping-Pong face à la commutation des E_i, D_i, est décidable en temps NP (en la taille du protocole). Ceci sera fait en deux temps. Un codage du problème Integer Programming montrera tout d'abord que ce problème est au moins NP difficile. Puis nous donnerons un algorithme de décision NP pour ce problème.

6.2.1 Présentation du modèle

Pour permettre la commutation des E_i et D_i, il nous suffit de modifier l'équivalence de mots $\equiv$ définie précédemment. On a la remplace par la suivante :

Définition 6.2.1.1 *Équivalence de mots sur A.*

Soit $A = \{E_1, D_1, .., E_n, D_n, i_1, d_1, .., i_k, d_k, d\}$ un alphabet. On pose $\equiv$ la plus petite relation symétrique, réflexive *et* transitive *sur les mots de A telle que*

- *pour tout $i \in \{1, .., n\}$ ou $j \in \{1, .., k\}$, $E_iD_i \equiv D_iE_i \equiv d_ji_j \equiv di_j \equiv \epsilon$,*
- *pour tous $i, j \in \{1, .., n\}$, $E_iE_j \equiv E_jE_i$, $D_iD_j \equiv D_jD_i$, et $E_iD_j \equiv D_jE_i$.*
- *pour tous mots β_1, β_2, γ et γ' sur A, si $\gamma \equiv \gamma'$ alors $\beta_1\gamma\beta_2 \equiv \beta_1\gamma'\beta_2$.*

La définition d'attaque sur un protocole Ping-Pong quant à elle ne change pas, excepté qu'elle utilise maintenant la définition précédente d'équivalence $\equiv$ sur les mots de A. Le symbole $\equiv$ désigne à partir de maintenant exclusivement cette (nouvelle) notion d'équivalence. En résumé, le problème que l'on souhaite résoudre est le suivant :

Problème : *Insécurité des Protocoles Ping-Pong avec Commutation (IPPC).*
Soit $A = \{E_1, D_1, .., E_n, D_n, i_1, d_1, .., i_k, d_k, d\}$ *un alphabet.*
Données : *un ensemble fini* $\{\alpha_i \,|\, i \in \{0, .., p\}\}$ *de mots sur* A.
Question : *Existe-t-il une liste d'indices* $I = \{I_i \in \{1, .., p\}\}_{i \in \{1,..,m\}}$ *tels que* $\alpha_{I_1}..\alpha_{I_m}\alpha_0 \equiv \epsilon$.

Suivant la notation habituelle des mots, on note A^* l'ensemble des mots sur l'alphabet A. Cette formulation du problème de l'insécurité avec commutation est très proche de la formulation sans commutation. Cependant, les propriétés de commutation des opérateurs E_x et D_x permettent de représenter les instances de ce problème de manière beaucoup plus compacte que dans le cas sans commutation. Pour formaliser ceci, nous avons besoin de quelques notations. Étant donné $\delta \in \{E_1, D_1, .., E_n, D_n\}^*$ et un entier $k \in \mathbb{N}$, on note :

$$\delta^{(k)} \;=\; \delta ... \delta \qquad k \text{ fois}$$

De plus, pour tout mot $\delta \in A^*$, on remarque qu'il existe $v_0, .., v_k \in (ED)^*$ et $w_1, .., w_{k-1} \in ID$, avec $k \in \mathbb{N}$, $ED = \{E_1, D_1, .., E_n, D_n\}$ et $ID = \{i_1, d_1, .., i_k, d_k, d\}$, tels que $\delta = v_0 w_1 v_1 .. w_{k-1} v_k$. En outre, pour tout $i \in \{0, .., k\}$, on a $v_i \equiv E_1^{(a_1^i)} D_1^{(b_1^i)} ... E_n^{(a_n^i)} D_n^{(b_n^i)}$ avec $a_j^i, b_j^i \in \mathbb{N}$. On peut alors définit la taille d'une représentation de δ :

$$\|\delta\| \;=\; \sum_{i \in \{1,..,k-1\}} \#w_i \;+\; \sum_{i \in \{0,..,k\}} \sum_{j \in \{1,..,n\}} \ln(\text{abs}\left(a_j^i - b_j^i\right) + 1)$$

Avec $\|w_i\|$ le nombre de lettres du mot w_i, ln la partie entière du logarithme en base deux, et abs la valeur absolue. Avec cette notion de taille, on exprime le fait que chaque bloc commutatif $v_i \in (ED)^*$ peut être simplement représenté par le nombre d'occurrences de chaque lettre E_j ou D_j écrit en base deux. Par opposition, les opérateurs de ID ne commutent pas et doivent donc être énumérés un par un. Par exemple, le mot $i_1 i_1 E_1 E_1 E_1 E_1 D_2 D_2 d_1$ sera vu comme $i_1 i_1 E_1^{(4)} D_2^{(2)} d_1$ et aura donc une taille de $2 + \ln(4) + \ln(2) + 1 = 6$. Par comparaison, ce même mot sans commutation aurait une taille de 9. Dans le pire des cas, i.e. pour $\delta = E_1 D_1 E_1 D_1 ... E_1 D_1$, on remarque que la taille de δ est $2.\ln(n)$ au lieu de $2.n$ s'il n'y avait pas de commutation. On peut à présent définir la taille d'une instance d'IPPC :

Définition 6.2.1.2 *Taille d'une instance d'IPPC.*
La taille d'une instance $\{\alpha_i \,|\, i \in \{0, .., p\}\}$ *d'IPPC sur un alphabet* A *est* $\sum_{i=0..p} \|\alpha_i\|$.

6.2.2 L'insécurité est NP-difficile.

Pour montrer que le problème IPPC est NP-difficile, nous allons réduire à celui-ci une version un peu spéciale du problème de programmation entière (Integer Programming), i.e. :

Problème : *Problème de Programmation Entière (PPE)*
Données : *Un ensemble* $\{((a_1^i, b_1^i), .., (a_N^i, b_N^i)) \in \left(\mathbb{N}^2\right)^N \,|\, i \in \{0, .., p\}\}$, *avec* $N \in \mathbb{N}^*$ *et* $p \in \mathbb{N}$. *On peut aussi le voir comme* N *instances de la version simple de PCP.*
Question : *Existe-t-il une liste d'indices* $\{I_i \in \{1, .., p\}\}_{i \in \{1,..,m\}}$ *telle que pour tout* $j \in \{1, .., N\}$, $a_j^0 - b_j^0 + \sum_{i \in \{1,..,m\}} (a_j^i - b_j^i) = 0$

Les entiers sont naturellement représentés en base deux. On peut reconnaître assez facilement le problème bien connu Integer Programming (version simple, connu pour être NP-complet) dans le problème PPE ci-dessus. Voir par exemple [48] pour la complexité du problème Integer Programming. Ainsi :

Proposition 6.2.2.1 *PPE est NP-complet.*

Preuve. Reprenons les notations du problème PPE. Il existe une solution I à PPE :
- ssi pour tout $j \in \{1,..,N\}$, $a_j^0 - b_j^0 + \sum_{i\in\{1,..,m\}}(a_j^i - b_j^i) = 0$
- ssi il existe $(x_1,..,x_N) \in \mathbb{N}^N$ tel que pour tout $j \in \{1,..,N\}$, $\sum_{i\in\{1,..,p\}} x_i.(a_j^i - b_j^i) = b_j^0 - a_j^0$. En effet, x_i est simplement le nombre d'occurrences de l'indice I_i dans la solution I. On se contente donc d'une somme de 1 à p avec coefficients.
- ssi il existe un vecteur $X \in \mathbb{N}^N$ tel que $M.X = B$, avec $B_j = b_j^0 - a_j^0$ pour tout $j \in \{1,..,N\}$, et $M_{i,j} = a_j^i - b_j^i$ pour tous $i \in \{1,..,p\}$ et $j \in \{1,..,N\}$. On reconnaît ici naturellement le problème classique Integer Programming, version simple.

Enfin, la traduction de PPE vers Integer Programming (ou inversement) est naturellement linéaire, ce qui prouve la proposition. □

On remarque que le problème PPE reste NP-complet même si l'on impose à la liste d'indices $\{I_i \in \{1,..,p\}\}_{i\in\{1,..,m\}}$ de contenir au moins une fois chaque $i \in \{1,..,p\}$. Il ne nous reste maintenant plus qu'à réduire (en temps polynomial) le problème PPE à IPPC, pour prouver que IPPC est NP-difficile.

Proposition 6.2.2.2 *On peut réduire PPE à IPPC en temps linéaire.*

Preuve. Reprenant les notations de ces deux problèmes, nous allons construire une instance d'IPPC à partir de toute instance de PPE. Soit $\{((a_1^i, b_1^i),..,(a_N^i, b_N^i)) \in \left(\mathbb{N}^2\right)^N \mid i \in \{0,..,p\}\}$ une instance de PPE. On pose $n = N$, et :

$$\forall i \in \{0,..,p\},\ \alpha_i = \prod_{j\in\{1,..,N\}} E_j^{(a_j^i)}\, D_j^{(b_j^i)}$$

De cette manière, chaque α_i contient exactement a_j^i fois E_j et b_j^i fois D_j, pour tout $j \in \{1,..,N\}$ et $i \in \{1,..,p\}$. L'ordre dans les α_j n'a pas d'importance, car les E_i et D_i commutent. Montrons que ces α_i forment une instance d'IPPC ayant une solution ssi cette instance de PPE en a une. En effet, cette instance de PPE a une solution
- ssi il existe $\{I_i \in \{1,..,p\}\}_{i\in\{1,..,m\}}$ t.q. $\forall j \in \{1,..,N\}$, $a_j^0 - b_j^0 + \sum_{i\in\{1,..,m\}}(a_j^i - b_j^i) = 0$,
- ssi il existe $\{I_i \in \{1,..,p\}\}_{i\in\{1,..,m\}}$ tel que $\prod_{j\in\{1,..,N\}} E_j^{(a_j^0 + \sum_{i\in 1..m} a_j^i)}\, D_j^{(b_j^0 + \sum_{i\in 1..m} b_j^i)} \equiv \epsilon$,
- ssi il existe $\{I_i \in \{1,..,p\}\}_{i\in\{1,..,m\}}$ tel que $\left(\prod_{i\in\{1..m\}} \alpha_i\right) \alpha_0 \equiv \epsilon$.
- ssi l'instance $\{\alpha_i \mid i \in \{0,..,p\}\}$, pour l'alphabet $A = \{E_1, D_1,.., E_N, D_N\}$, admet une solution.

De plus, la construction de l'instance $\{\alpha_i \mid i \in \{0,..,p\}\}$ et de l'alphabet A est naturellement linéaire, ce qui prouve la proposition. □

6.2.3 Problème annexe.

Pour résoudre le problème IPPC, nous avons tout d'abord besoin de définir et résoudre un problème plus restreint. Il s'agit du problème suivant :

Problème : *Problème de mot sur ED^* (PMED)*

Données : Un ensemble $\Delta = \{\delta_0, \delta_1, .., \delta_p, \delta_{p+1}\}$ de mots sur $(ED)^*$, et une relation binaire $\lhd$ sur Δ.

Question : Existe-t-il une liste d'indices $\{I_k \in \{1, .., p\}\}_{k\in\{1,..,m\}}$ telle que $\delta_{p+1}\delta_{I_1}..\delta_{I_m}\delta_0 \equiv \epsilon$ et $\delta_{p+1} \lhd \delta_{I_1} \lhd ... \lhd \delta_{I_m} \lhd \delta_0$.

On utilise ici la même notion de taille que pour IPPC (restreinte à $(ED)^*$ évidement). Ce problème est important dans la mesure où nous en aurons besoin pour construire une procédure de décision pour IPPC. En particulier, on peut coder assez facilement une instance de PMED dans IPPC. L'idée est d'utiliser les opérateurs i_x et d_x pour coder la relation $\lhd$. Par exemple, si $\alpha_1 = i_{1,1}\delta_1 d_{2,1}$ et $\alpha_2 = i_{1,2}\delta_2 d_{2,2}$ codent respectivement δ_1 et δ_2 dans une instance d'IPPC, il suffit d'ajouter $\alpha'_{1,2} = i_{2,1}d_{1,2}$ à cette instance pour coder $\delta_1 \lhd \delta_2$. De cette manière, toute solution de cette instance d'IPPC sera nécessairement de la forme :

$$\delta_{p+1}... \left(i_{1,I_k}\delta_{I_k}d_{2,I_k}\right)\left(i_{2,I_k}d_{1,I_{k+1}}\right)\left(i_{1,I_{k+1}}\delta_{I_{k+1}}d_{2,I_{k+1}}\right)....\delta_0 \equiv \epsilon$$

i.e. $\delta_0\delta_{I_1}...\delta_m\delta_p \equiv \epsilon$ et $\delta_{p+1} \lhd \delta_{I_1} \lhd ... \lhd \delta_{I_m} \lhd \delta_0$. En outre, toute solution de PMED correspond à une solution de cette instance d'IPPC de cette forme.

Le problème PMED semble très lié à notre formulation d'IPPC. Mais en fait il est équivalent au problème suivant, de somme de vecteurs d'entiers sur un graphe :

Problème : PMED, seconde version (PMED2)

Données : Un graphe orienté G de transitions étiquetées sur $\boldsymbol{Z}^t$, et un état $s \in E$.

Question : Existe-t-il un cycle (orienté) dans G, passant par s, d'étiquettes $v_1, .., v_m$ telles que $\sum_{i=1..m} v_i = (0, .., 0)$.

Il suffit de remarquer que tout mot $\delta \in ED^*$ peut être représenté par un vecteur $\overline{\delta} = (r_1, .., r_t) \in \boldsymbol{Z}^t$, avec $\delta = \delta_1..\delta_t$ et $\delta_i = E_i^{(r_i)}$ si $r_i \geq 0$, $\delta_i = D_i^{(r_i)}$ sinon. De cette manière, un produit sur ED^* est une somme sur $\boldsymbol{Z}^t$. Pour toute instance $(\{\delta_0, \delta_1, .., \delta_p, \delta_{p+1}\}, \lhd)$ de PMED, il suffit alors de considérer le graphe $G = (E, T)$, d'états $E = \{q_i, q'_i \,|\, i = 0..p+1\}$, et de transitions $(q_i, \overline{\delta_i}, q'_i) \in T$ pour tout $i \in \{0, .., p+1\}$ et $(q'_i, \overline{1}, q_j) \in T$ pour tout $\delta_i \lhd \delta_j$ avec $i \neq p+1$ et $j \neq 0$. En effet, cette instance de PMED admet une solution ssi il existe un chemin dans G de $s = q_0$ à q'_{p+1} étiqueté par $v_1, .., v_m$ avec $\sum_{i=1..m} v_i = 0$. Comme q_0 n'a pas de précédesseur et q'_{p+1} n'a pas de successeur, il suffit d'ajouter une transition $(q'_{p+1}, \overline{1}, q_0)$ à T pour fermer la boucle. Ce codage est linéaire. De plus, on peut naturellement effectuer le codage inverse de manière équivalente[15].

On définit naturellement la taille du graphe $G = (E, T)$ étiqueté sur $\boldsymbol{Z}^t$ à l'aide de la taille $\|..\|$ sur $(ED)^*$, grâce à la correspondance précédente entre $(ED)^*$ et $\boldsymbol{Z}^t$:

$$\|G\| \;=\; \#E \;+ \sum_{(q,\overline{\delta},q')\in T} \|\delta\|$$

Cette taille s'étend naturellement à l'ensemble des chemins de G :

$$\forall c \in \mathcal{T}, \text{ si } c = q_0 \rightarrow_{\overline{\delta_1}} .. \rightarrow_{\overline{\delta_n}} q_n \text{ alors } \|c\| = \sum_{i\in 1..n} \|\delta_i\|$$

[15] En ajoutant au graphe G un nombre linéaire d'états pour séparer toutes les transitions de G par une transition étiquetée par $\overline{1}$.

avec la notation habituelle des chemins. On peut alors examiner la complexité du problème PMED :

Proposition 6.2.3.1 *Le problème PMED est décidable en temps NP.*

PREUVE. Nous allons utiliser la seconde formulation de ce problème (PMED2). Pour pouvoir construire un algorithme NP de décision de PMED2, nous allons montrer que ce problème est équivalent à l'existence d'une solution pour le problème Integer Programming[16]. □

PREUVE. Soient $G = (E, T)$ un graphe étiqueté sur $\boldsymbol{Z}^k$ et $s \in E$, tels que le problème PMED2 sur G et s admette une solution C. Nous allons examiner les transitions utilisées par C. Pour celà, notons $T' = \{t_1, .., t_p\}$ l'ensemble des transitions de G apparaissant dans C, et $E' \subseteq E$ l'ensemble des états de E traversés par C, i.e. :

$$\forall (q, v, q') \in T,\ q \rightarrow_v q' \in C \text{ ssi } q \in E' \text{ et } (q, v, q') \in T'$$

Par définition de C, nous avons les trois propriétés suivantes :

$$\left.\begin{array}{rl} \text{Etiquette nulle :} & \sum_{i\in 1..p} n_i.v_i \ = \ (0, .., 0) \ \in \boldsymbol{Z}^k \\ \text{Equation des flots :} & \forall r \in E',\ \sum_{i\in 1..p}^{r=q_i} n_i \ = \ \sum_{i\in 1..p}^{r=q'_i} n_i \\ \text{Contributions positives :} & \forall i \in 1..p,\ n_i > 0 \end{array}\right\} (Eq1)$$

avec $\forall i \in 1..p$, $t_i = (q_i, v_i, q'_i)$ et n_i le nombre d'occurences de la transition t_i dans C. La première propriété vient de la définition d'une solution à PMED2. La seconde propriété vient de la définition d'un cycle dans G : tout état r doit avoir autant de transitions entrantes que sortantes dans C. Enfin, la troisième propriété est une tauthologie puisque T' ne contient que les transitions de C.

L'ensemble d'équations $(Eq1)$ est suffisant pour définir une solution de PMED2. En effet, tout d'abord on sait déjà que si (G, s) admet une solution pour PMED2, alors il existe $G' = (E', T')$ sous graphe de G et $(n_1, .., n_p) \in \mathbb{N}^p$ satisfaisant $Eq1$, avec les notations précédentes. Ensuite, étant donné un sous graphe $G' = (E', T')$ de G, s'il existe $(n_1, .., n_p) \in \mathbb{N}^p$ satisfaisant l'équation $Eq1$ avec G', alors (G, s) admet une solution pour PMED2. Pour s'en rendre compte, il suffit de constater que tout graphe G' pondéré (par $n_i > 0$) satisfaisant l'équation des flots $(Eq1)$ admet un cycle passant par tous les états (car $\forall i$, $n_i \neq 0$) et respectant les poids n_i : grâce à la première propriété de $(Eq1)$, ce cycle est solution de PMED2 sur (G, s).

Notre but est donc de donner une version matricielle de $(Eq1)$. Pour celà, posons :

$$\forall r \in E', \forall i \in 1..p,\ \alpha_i^r = \left\{\begin{array}{ll} 1 & \text{si } q_i \neq r \text{ et } q'_i = r \\ -1 & \text{si } q_i = r \text{ et } q'_i \neq r \\ 0 & \text{sinon.} \end{array}\right.$$

avec pour tout $i \in 1..p$, $t_i = (q_i, v_i, q'_i)$. Ainsi, $\alpha_i^r = 1$ si t_i est une transitions entrante sur r, $\alpha_i^r = -1$ si t_i est une transition sortante de r, et $\alpha_i^r = 0$ si t_i n'a rien à voir avec r ou si t_i est une boucle unitaire sur r. On ne compte pas les boucles unitaires, car elles respectent déjà l'équation des flots[17]. Pour donner une version matricielle de l'équation des flots, on pose de nouveaux vecteurs u_i :

$$\begin{array}{rl} \forall i \in 1..p, & u_i = (u_i^1,\ u_i^2,\ ...,\ u_i^{k+n}) \\ \text{avec} & \forall i, j,\ u_i^j = \left\{\begin{array}{ll} v_i^j & \text{si } j \leq k \\ \alpha_i^{q_{j-k}} & \text{si } j > k \end{array}\right. \end{array}$$

[16]Prob : C'est $\exists ?X$ t.q. $\forall i$, $X_i > 0$ et $M.X = 0$. C'est Integer Programming ou un autre ?

[17]On pourait indéférement les compter comme entrantes et sortantes, i.e. $1 - 1 = 0$ ce qui revient au même.

avec $E' = \{q_1, .., q_n\}$, $n = \#E'$, et $v_i = (v_i^1, .., v_i^k)$ pour tout $i \in 1..p$. De cette manière, on a :

$$\forall (n_1, .., n_p) \in \left(\mathbb{N}^+\right)^p,\ (n_1, .., n_p) \text{ sol. de } Eq1 \text{ ssi } \sum_{i \in 1..p} n_i.u_i = (0, .., 0)$$

En effet, pour les k premières coordonnées de $\{u_i\}$, on a la première prop. de $(Eq1)$, et pour les coordonnées $k+1$ à $k+\#E'$ on vérifie que chaque état de E' vérifie localement l'équation des flots. Ainsi, il existe une solution à PMED2 sur G et s ssi

$$\exists (n_1, .., n_p) \in \left(\mathbb{N}^+\right)^p \text{ tel que } \sum_{i \in 1..p} n_i.u_i = (0, .., 0) \qquad (Eq2)$$

On reconnait ici immédiatement une instance d'Integer Programming. En outre, on a $p \leq \#T$, $n \leq \#E$, et $\left\|\{u_i\}_{i \in 1..p}\right\| \leq \left\|\{v_i\}_{i \in 1..p}\right\| + p.\#E \leq 2.\|G\|^2$, i.e. cette instance est de taille bornée polynomialement en la taille de G. Il nous suffit donc de tester l'existence d'une solution de cette instance l'Integer Programming pour savoir si PMED2 adment une solution sur G et s. □

Un point important de la preuve précédente est qu'un algorithme NP de décision de PMED ne peut pas se contenter de choisir un cycle dans G puis de vérifier qu'il est bien solution du problème PMED2 correspondant sur G. Dans la section suivante, nous aurons besoin de séparer l'objet choisi pour résoudre une instance de PMED de l'algorithme polynomial vérifiant que cet objet est bien solution de PMED. Pour celà, on pose la définition suivante :

Définition 6.2.3.2 *Solution potentielle de PMED*[18].

Soit $(\Delta, \lhd)$ une instance de PMED. On appelle solution potentielle de PMED sur $(\Delta, \lhd)$ une représentation d'un entier $p \leq \|G\|$ et d'un vecteur $(n_1, .., n_p) \in (\mathbb{N}^+)^p$ t.q. $\forall i$, $ln(n_i) \leq p(2.\|G\|^2)$, avec (G, s) l'instance de PMED2 représentant $(\Delta, \lhd)$.

Grâce à la preuve précédente, on sait que PMED2 admet une solution sur G, s ssi il existe de tels entiers p et $n_1, .., n_p$ vérifiant $Eq2$. Etant donnés une instance de PMED et une solution potentielle de PMED sur cette instance, on peut donc décider en temps déterministe polynomial si cette solution potentielle satisfait $Eq2$: il suffit de construite de graphe G et l'état s (temps linéaire), de construire les vecteurs u_i (i.e. $Eq2$, en temps polynomial), puis de vérifier que cette solution potentielle est bien solution du système d'équations $Eq2$ ainsi construit (temps polynomial). De cette manière, une solution potentielle de PMED est bien un objet de taille polynomiale choisi pour l'algorithme NP de décision de PMED, et non une liste d'indices "réellement" solution de PMED, qui serait de taille exponentielle.

6.2.4 Algorithme.

Nous allons maintenant donner un algorithme (NP) permettant de décider si une instance d'IPPC admet une solution ou pas. Pour cela, supposons choisie une telle instance $\{\alpha_i \,|\, i \in \{0, .., p\}\}$, sur l'alphabet $A = \{E_1, D_1, ..E_t, D_t, i_1, d_1, .., i_t, d_t, d\}$ avec $t \in \mathbb{N}^*$ fixé. On considère autant de couples (E_x, D_x) que de couples (i_x, d_x) pour simplifier les notations (la taille du problème augmente au plus linéairement). On rappelle que l'on note $ED = \{E_1, D_1, ..E_t, D_t\}$, i.e. uniquement les opérateurs d'encryption et de décryptage, et que $ID = \{i_1, d_1, .., i_t, d_t, d\}$. De plus, on remarque que tout mot $w \in A^*$ possède une représentation dans $(ED^*.ID)^*.ED^*$, que l'on notera $\overline{w} = (w_1, w_1', .., w_m)$, avec $w = w_1 w_1' ... w_m$, pour tout $i \in \{1, ..m\}$, $w_i \in ED^*$ et pour tout $i \in \{1, .., m-1\}$, $w_i' \in ID$.

[18] Quel est $p(..)$ pour Integer Programming ?

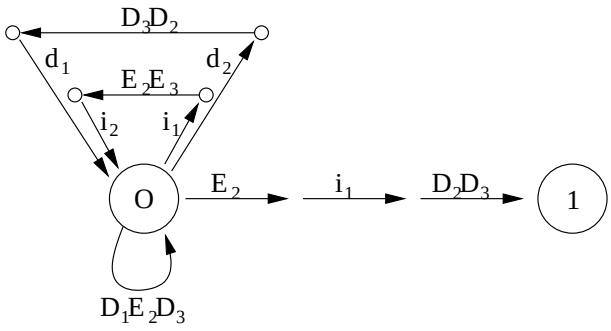


FIG. 6.1: Exemple d'automate.

Nous allons commencer par construire un automate $\mathcal{A}$ (en fait un simple graphe orienté étiqueté) sur l'alphabet $ED^* \cup ID$, représentant les différentes manières dont on peut agencer les mots α_i, a peu près de la même manière que [42]. La seule différence est que l'on n'utilise qu'une seule transition pour chaque sous mot (maximal) de α_i dans $(ED)^*$. Formellement, c'est le plus petit automate (non déterministe) tel que :

- $\mathcal{A}$ possède un seul état initial $Init$ et un seul état final Fin.
- Excepté $Init$ et Fin, tous les états de $\mathcal{A}$ possèdent un unique prédécesseur et un unique successeur.
- Il existe dans $\mathcal{A}$ un chemin de l'état $Init$ vers l'état Fin, étiqueté par α_0, et dont les étiquettes des transitions suivent $\overline{\alpha_0}$. De cette manière, on construit des chemins n'ayant jamais successivement deux étiquettes dans ED^*.
- Pour tout $i \in \{1,..,p\}$, il existe un chemin de l'état $Init$ vers l'état $Init$ étiqueté par α_i.

Notons S l'ensemble des états de $\mathcal{A}$, et $\{\beta_i\}_{i\in\{1,..,q\}} = \{\alpha_i \,|\, i \in \{1,..,p\} \text{ et } \alpha_i \in ED^*\}$. Pour tous $s_1, s_2 \in S$, on note $s_1 \xrightarrow{\delta} s_2$ quand il existe une arête dans $\mathcal{A}$ étiquetée par δ. Par extension, on dit qu'un chemin $s_1 \xrightarrow{\delta_1} s_2 ... \xrightarrow{\delta_n} s_n$ dans $\mathcal{A}$ est étiqueté par $\delta_1..\delta_n$. La figure 6.1 donne un exemple d'automate pour $\alpha_0 = E_2 i_1 D_2 D_3$, $\alpha_1 = i_1 E_2 E_3 i_2$, $\alpha_2 = d_2 D_3 D_2 d_1$, et $\alpha_3 = D_1 E_2 D_3$.

Il est intéressant de remarquer que s'il existe $i \in \{1,..,p\}$ tel que $\alpha_i \equiv a\, d_x\, b\, i_{x'}\, c$, avec $a, c \in A^*$, $b \in ED^*$, et $b \not\equiv \epsilon$ ou $x \neq x'$, alors aucune solution d'IPPC ne peut utiliser α_i. De même avec $\alpha_i \equiv a\, d\, b\, i_{x'}\, c$. En effet, il n'y a dans b aucun opérateur $i_{x'}$, d_x ou d permettant d'éliminer d_x ou $i_{x'}$. De plus, ce critère est facile à tester : il suffit d'enlever de α_i tous les sous mots δ tels que $\delta \equiv \epsilon$, puis de tester s'il existe un sous mot de la forme $d_x\, i_{x'}$, $d\, i_{x'}$, $d_x\, b\, i_{x'}$ ou $d\, b\, i_{x'}$ avec $b \in (ED)^*$ (on a $b \not\equiv \epsilon$ et/ou $x \neq x'$ par construction). En conséquence, on suppose à partir de maintenant que pour tout $i \in \{1,..,p\}$, $\alpha_i \in (EDi)^*\, ED^*\, (dED)^*$, avec $(EDi)^* = (ED^*\, \{i_1,..,i_t\})^*$ et $(dED)^* = (\{d, d_1,..,d_t\}\, ED^*)^*$, et que pour tout sous mot δ de α_i, on a $\delta \not\equiv \epsilon$.

Notons $\{\delta_i \,|\, i \in \{1,..,p\}\}$ l'ensemble de mots de ED^* tel que pour tout $i \in \{1,..,p\}$, $\alpha_i = a_i \delta_i b_i$ avec $a_i \in (EDi)^*$ et $b_i \in (dED)^*$. En outre, on a construit l'automate $\mathcal{A}$ de manière à avoir une boucle sur $Init$ pour chaque mot α_i. On pose donc $\{r_i, r'_i \,|\, i \in \{1,..,p\}\}$ un ensemble d'états de $\mathcal{A}$ tels que pour tout $i \in \{1,..,p\}$, la boucle dans $\mathcal{A}$ représentant α_i est $Init \xrightarrow{\in(EDi)^*} .. \xrightarrow{} r_i \xrightarrow{\delta_i} r'_i \xrightarrow{\in(dED)^*} .. \xrightarrow{} Init$. Remarque : si $\delta_i = \epsilon$, on a $r_i = r'_i$.

Nous allons utiliser le problème PMED pour construire une relation $\succ$ sur les états S de $\mathcal{A}$ telle que pour tous $s_1, s_2 \in S$, $s_1 \succ s_2$ ssi il existe un chemin dans $\mathcal{A}$ de s_1 vers s_2, et étiqueté par

un mot $\delta \equiv \epsilon$. Une fois cette relation construite, il ne nous restera plus qu'à tester si $Init \succ Fin$. Commençons par donner une définition de $\succ$ plus constructive. Nous montrerons ensuite que cette relation vérifie bien la propriété ci-dessus.

Définition 6.2.4.1 *Définition de $\succ$.*

Posons $\succ = \succ_{\#S^2}$, avec $\{\succ_i \mid i \in \{0, .., \#S^2\}\}$ l'ensemble des (plus petites) relations binaires sur S telles que :

1. *Pour tous $s \xrightarrow{\delta} s'$ avec $\delta \equiv \epsilon$, on a $s \succ_0 s'$.*
2. *Clôture réflexive : Pour tout $s \in S$, on a $s \succ_0 s$.*
3. *Clôture transitive : Pour tous $s_1 \succ_k s_2 \succ_k s_3$, on a $s_1 \succ_{k+1} s_3$.*
4. *Si $s_1 \xrightarrow{\delta} s_2 \succ_k s_3 \xrightarrow{\delta'} s_4$ avec $\delta, \delta' \in ID$ et $\delta\delta' \equiv 1$, alors $s_1 \succ_{k+1} s_4$.*
5. *Pour tous $s, s' \in S$ avec $s \xrightarrow{\delta_0 \in ED^*} r'_0 \longrightarrow .. \longrightarrow Init$ et $Init \longrightarrow .. \longrightarrow r_{p+1} \xrightarrow{\delta_{p+1} \in ED^*} s'$ deux chemins sans boucle (i.e. suffixe ou préfixe d'un α_i), s'il existe une solution à l'instance $(\{\delta_0, \delta_1, .., \delta_p, \delta_{p+1}\}, \triangleleft)$ de PMED, avec $\delta_i \triangleleft \delta_j$ ssi $r'_i \succ_k r_j$, alors on a $s \succ_{k+1} s'$.*
   De même pour $s = r$ (i.e. $\delta_0 = \epsilon$), ou $s' = r'$ (i.e. $\delta_{p+1} = \epsilon$), ou les deux.
6. *Si $s \succ_k s'$, alors $s \succ_{k+1} s'$.*

Rq : les états r_i et r'_i, pour $i \in \{1, .., p\}$, ont été définis avec δ_i, mais les états r'_0 e r_{p+1} sont quelconques.

Dans cette définition, le point 1 est le point de départ de la construction de $\succ$. Les points 2 et 3 forment une clôture réflexive et transitive tout à fait classique. Enfin, les points 4 et 5 sont le coeur de la construction de $\succ$. Le point 4 réalise une clôture par les opérateurs ID, et le point 5 identifie des chemins de $\mathcal{A}$ étiquetés dans ED^* (modulo $\succ$) réductibles sur ϵ. Enfin, le point 6 garanti simplement la croissance de $\{\succ_0, .., \succ_{\#S^2}\}$.

Commençons par prouver que $\succ$ vérifie bien la propriété annoncée :

Proposition 6.2.4.2 *$\succ$ est bien construit.*

Pour tous $s, s' \in S$, on a $s \succ s'$ ssi il existe un chemin de s vers s' étiqueté par $\delta \equiv \epsilon$.

Preuve. Tout d'abord, on constate que par construction, si $s \succ s'$ alors il existe bien un tel chemin dans $\mathcal{A}$. En effet, si cette propriété est vraie pour $\succ_k$, alors elle est nécessairement vraie pour $\succ_{k+1}$, par construction de $\succ_{k+1}$: Pour les points 3 et 4, c'est évident. Pour le point 5, une solution à PMED définit un chemin de s à s' étiqueté par $\delta_0\gamma_1...\delta_p\gamma_p\delta_{p+1}$, avec $\gamma_k \equiv \epsilon$ l'étiquette d'un chemin de r'_i à r_j pour $r'_i \succ_k r_j$, et $\delta_0..\delta_{p+1} \equiv \epsilon$. On a donc bien un chemin de s à s' étiqueté par $\delta \equiv \epsilon$.

Il nous faut donc montrer que ces points sont suffisants. Tout d'abord, on remarque que $\succ$ est nécessairement clos par les points 3 à 5, i.e. $\succ_{\#S^2+1} = \succ_{\#S^2}$. En effet, $\succ$ contient au plus $\#S^2$ relations $s \succ s'$ distinctes, et si $\succ_i = \succ_{i+1}$ alors $\succ_i = \succ_{i+k}$ pour tout k (c'est l'égalité sur les relations, i.e. $\succ_i = \succ_j$ ssi $\forall s, s'$, $(s \succ_i s') \Leftrightarrow (s \succ_i s')$).

Nous allons procéder par itération sur la longueur du plus petit chemin de s vers s', étiqueté par $\delta \equiv \epsilon$. On note $s \succcurlyeq_n s'$ quand il existe un chemin de s à s' étiqueté par $\delta \equiv \epsilon$ avec n est la longueur du plus petit chemin de ce type. Nous allons itérer sur n en montrant que pour tout $m \leq n$, si $s \succcurlyeq_m s'$ alors $s \succ s'$ (propriété P_n). Le point de départ de d'itération est P_1 : si $s \xrightarrow{\delta} s'$ avec $\delta \equiv \epsilon$, alors grâce au point 1, on a $s \succ s'$, et si $s = s'$ alors on utilise le point 2. Supposons à présent que la propriété P_{n-1} est vérifiée, avec $n \geq 2$, et que $s_1 \succcurlyeq_n s_n$.

Posons $C = s_1 \xrightarrow{\lambda_1} s_2 .. \xrightarrow{\lambda_{n-1}} s_n$ l'un des plus petits chemins de s_1 vers s_n avec $\lambda_1..\lambda_n \equiv \epsilon$. On a plusieurs cas :

- S'il existe $i \in \{1, .., n-1\}$ tel que $\lambda_1..\lambda_i \equiv \epsilon$ et $\lambda_{i+1}..\lambda_n \equiv \epsilon$, alors on a $s_1 \succcurlyeq_i s_i$ et $s_{i+1} \succcurlyeq_{n-i} s_n$. Grâce à P_{n-1}, on obtient $s_1 \succ s_i$ et $s_{i+1} \succ s_n$, d'où $s_1 \succ s_n$ grâce au point 3.

- Sinon, on examine la nature de λ_1 :
  - Si $\lambda_1 \in ID$, alors nécessairement $\lambda_1 \in \{d_x, d \,|\, x \in \{1, .., t\}\}$ et $\lambda_1\lambda_n \equiv \epsilon$. En effet, si $\lambda_1 = i_x$ il n'existe aucun moyen de le simplifier. On a donc $s_2 \succcurlyeq_{n-2} s_{n-1}$, d'où $s_2 \succ s_{n-1}$ grâce à P_{n-1}, et $s_1 \succ s_n$ grâce au point 4.
  - Si $\lambda_1 \in ED^*$, nous devons analyser plus finement la structure de ce chemin. Tout d'abord, on peut identifier tous les sous mots h de $\lambda_1..\lambda_n$ de la forme $d_x\gamma i_x$ ou $d\gamma i_x$ avec $\gamma \equiv \epsilon$, maximaux pour la relation de sous mot. Soit H l'ensemble de ces sous mots. Tout $h \in H$ correspond à un sous chemins de C d'un état $s_i = q_h$ à un état $s_j = q'_h$, $i < j$, avec $s_i \succcurlyeq_m s_j$ pour $m < n$. Grâce à P_{n-1}, on a $q_h \succ q'_h$ pour tout $h \in H$. De plus, tous les opérateurs d, d_x ou i_x de $\lambda_1..\lambda_n$ sont dans un sous mot de cette forme (sinon, on aurait $\lambda_1..\lambda_i \not\equiv \epsilon$). En conséquence, il existe $\{s'_1, s''_1, .., s'_m, s''_m\} \subseteq \{s_1, .., s_n\}$ tels que $s_1 = s'_1$, $s_n = s''_m$, et :

$$(s_1 =) \overbrace{s'_1 \longrightarrow .. \longrightarrow s''_1}^{\delta'_1 \in ED^*} \succ \overbrace{s'_2 \longrightarrow .. \longrightarrow s''_2}^{\delta'_2 \in ED^*} \succ \ldots \succ \overbrace{s'_m \longrightarrow .. \longrightarrow s''_m}^{\delta'_m \in ED^*} \ (= s_n)$$

avec pour tout i, $C_i = \overbrace{s'_i \longrightarrow .. \longrightarrow s''_i}^{\delta'_i \in ED^*}$ est un chemin de s'_i vers s''_i étiqueté par $\delta'_i \in ED^*$. De plus, $s''_i \succ s'_{i+1}$ représente un sous chemin de C étiqueté par $d_x\gamma i_x$ ou $d\gamma i_x$, $x \equiv \epsilon$ quelconque. On suppose ici que $s_1 \neq s''_1$ et $s_n \neq s'_m$, mais les trois autres cas sont équivalents si l'on prend $\delta'_1 = \epsilon$ et/ou $\delta'_m = \epsilon$. On peut alors remarquer que pour tout $i \in \{2, ..m-1\}$, les états de C_i sont nécessairement dans $R = \{r_i, r'_i \,|\, i \in \{1, .., p\}\}$. En effet, supposons que ce ne soit pas le cas, i.e. il existe $i \in \{2, .., m-1\}$ tel que $s'_i \notin R$ et $s''_i \in R$ (Rq : si $s'_i \in R$, alors $s''_i \in R$ et inversement par définition de R). Soit B_k le chemin d'étiquette α_k d'$Init$ vers $Init$ et passant par s'_i, avec k quelconque. On n'a alors que deux cas, selon la position de s'_i dans B_k par rapport à r_k et r'_k :

$$\begin{array}{lll} \text{soit} & B_k = Init \longrightarrow .. \longrightarrow s_i \longrightarrow .. \longrightarrow r_k \longrightarrow .. \longrightarrow Init & (1) \\ \text{soit} & B_k = Init \longrightarrow .. \longrightarrow r'_k \longrightarrow .. \longrightarrow s_i \longrightarrow .. \longrightarrow Init & (2) \end{array}$$

Pour (1), il existe $j \in \{1, .., n\}$, i.e. une position dans le chemin C, tel que $\lambda_j = i_x$ et $\lambda_1..\lambda_{j-1} \equiv \epsilon$. Pour (2), il existe $j \in \{1, .., n\}$ tel que $\lambda_j = d_x$ (ou d) et $\lambda_{j-1}..\lambda_n \equiv \epsilon$. Dans les deux cas, on a une contradiction avec $\lambda_1..\lambda_n\lambda_j = i_x$ et $\lambda_1..\lambda_{j-1} \equiv \epsilon$. En conséquence :

$$\{\delta'_i \,|\, i \in \{2, .., m-1\}\} \subseteq \{\delta_i \,|\, i \in \{1, .., p\}\}^*$$

De la même manière, $\delta'_1 = \delta_0\delta''_1$ et $\delta'_m = \delta''_m\delta_{p+1}$ avec δ_0 et δ_{p+1} les étiquettes respectives de la première et de la dernière transition de C, et avec $\delta''_1, \delta''_m \in \{\delta_i \,|\, i \in \{1, .., p\}\}^*$. Posons r_0 et r_{p+1} tels que $s_1 \xrightarrow{\delta_0} r'_0$ et $r_{p+1} \xrightarrow{\delta_{p+1}} s_n$. Il existe donc une solution à l'instance $(\{\delta_0, \delta_1, .., \delta_p, \delta_{p+1}\}, \triangleleft)$ de PMED, avec $\delta_i \triangleleft \delta_j$ ssi $r'_i \succ_k r_j$. Ceci montre que l'on peut utiliser le point 5 sur s_1 et s_n, et donc que $s_1 \succ s_n$.

Ainsi, dans tous les cas on a obtenu $s_1 \succ s_n$, et la propriété P_{n-1} implique donc P_n pour tout n. Comme on a déjà prouvé P_1, on en déduit immédiatement que pour tous s et s', s'il existe un chemin de s à s' étiqueté par $\delta \equiv \epsilon$, alors $s \succ s'$. Et comme on a déjà prouvé le sens inverse, il y a bien équivalence entre $s \succ s'$ et l'existence d'un chemin de s à s' étiqueté par $\delta \equiv \epsilon$. □

Soit une instance $\{\alpha_i \mid i \in \{0,..,k\}\}$ d'IPPC à résoudre, de taille T.
De plus, soit $\mathcal{A}$ l'automate construit au début de cette section pour ce problème, et soient δ_i, r_i et r'_i pour $i \in \{1,..,p\}$ définis comme au début de cette section.

1. Choisir une liste $c_1,..,c_f$ d'éléments de S^2 sans redondance.
Pour chaque couple $(s,s') \in S^2$, choisir :
- $\delta_{s,s'}$ et $\delta'_{s,s'}$ deux étiquettes de transitions de $\mathcal{A}$ dans $(ED)^*$,
- $\lhd_{s,s'}$ une relation binaire sur $\Delta_{s,s'} = \{\delta_{s,s'}, \delta_1,..,\delta_p,\delta'_{s,s'}\}$, et
- $E_{s,s'}$ une solution potentielle à l'instance $(\Delta_{s,s'}, \lhd_{s,s'})$ de PMED.

2. Vérifier que pour tout $k \in \{1,..,f\}$, avec $c_f = (s,s')$, on a :
- soit $s = s'$ ou $s \xrightarrow{\delta} s'$ avec $\delta \equiv \epsilon$.
- soit il existe $c_i = (s,s_2)$ et $c_j = (s_2,s')$ avec $i < k$ et $j < k$.
- soit il existe $c_i = (s_2,s_3)$, $i < k$, tel que $s' \xrightarrow{\delta} s_2$ et $s_3 \xrightarrow{\gamma} s'$ avec $\delta\gamma \equiv \epsilon$.
- soit il existe $s \xrightarrow{\delta_{s,s'}} r'_0$ et $r_{p+1} \xrightarrow{\delta'_{s,s'}} s'$, avec
  - pour tous i,j, on a $\delta_i \lhd_{s,s'} \delta_j$ ssi il existe $k' < k$ t.q. $c_{k'} = (r'_i, r_j)$,
  - et $E_{s,s'}$ est une solution à l'instance $(\Delta_{s,s'}, \lhd_{s,s'})$ de PMED.

  Idem pour $s = r'_0$ et/ou $s' = r_{p+1}$.

Si $\exists i$, $c_i = (Init, Fin)$ alors répondre VRAI.

FIG. 6.2: Algorithme NP pour le problème IPPC.

Nous pouvons à présent donner un algorithme de décision d'IPPC, en suivant la définition de $\succ$. Il s'agit de l'algorithme présenté à la figure 6.2. Dans cet algorithme, une solution potentielle à une instance de PMED est une l'un des ensembles objets pouvant être choisi par l'algorithme de décision de PMED de la sous section précédente.

La justification de cet algorithme est assez simple. D'une part, on constate aisément toute liste $c_1, .., c_f$ vérifiant le point 2 de l'algorithme 6.2 est une restriction de $\succ$ (i.e. pour tout $c_k = (s, s')$, on a $s \succ s'$,). D'autre part, on peut naturellement choisir une liste $c_1, .., c_f$ complète (i.e. pour tout $s \succ s'$, il existe $k \leq f$ t.q. $c_k = (s, s')$) et respectant l'ordre induit par $\succ_1, .., \succ_{\#S^2}$ (i.e. si $s_1 \succ_k s_2$, $s_3 \not\succ_k s_4$, mais $s_3 \succ s_4$, alors $(s_1, s_2) = c_{k_1}$ et $(s_2, s_3) = c_{k_2}$ avec $k_1 < k_2$), ainsi que $\delta_{s,s'}$, $\delta'_{s,s'}$, et $\lhd_{s,s'}$ pour chaque $(s, s') \in S^2$ selon la définition de $\succ$. Par construction, $c_1, .., c_f$, $\delta_{s,s'}$, $\delta_{s,s'}$ et $\lhd_{s,s'}$ pour chaque $(s, s') \in S^2$ vérifient le point 2 de l'algorithme. Celui-ci est donc correct et complet. De manière équivalente, on pourrait calculer la clôture de $\succ_1$ par les points 3 à 6 de la définition de $\succ$, mais l'appartenance à NP serait moins évidente. En effet, on choisit un nombre polynomial d'objets de tailles polynomiales (en la taille de l'instance d'IPPC), puis on effectue une vérification en temps polynomial. Et comme on sait déjà que IPPC est NP-difficile, on obtient :

Théorème 6.2.4.3 *Le problème IPPC est NP-complet.*

6.3 Sessions bornées : Modèle par rôles étendu

Nous venons de voir que l'ajout de propriétés de commutation de clefs, même sur des protocoles aussi simples que les protocoles Ping-Pong, rend déjà le problème de l'insécurité décidable mais au moins NP-difficile. C'est un point intéressant, car cela montre la difficulté inhérente à la vérification de protocoles face à un opérateur d'encryption possédant cette propriété. D'un autre côté, nous avons présenté plusieurs algorithmes NP pour résoudre le problème de l'insécurité de protocoles face à des propriétés algébriques de certains opérateurs (xor, exponentielle). On peut donc se demander si l'utilisation d'un opérateur d'encryption commutative garde le problème de l'insécurité de protocoles "seulement" NP-complet ou s'il devient du coup plus difficile. C'est ce que nous allons examiner dans cette section.

A la manière des chapitres précédents, nous ne considérerons ici que des protocoles utilisant les opérateurs de Dolev-Yao habituels, plus les opérateurs produit et encryption commutative. On exclut donc les opérateurs xor et exponentiation, mais nous réutilisons l'opérateur produit. Cependant, même si nous utilisons dans cette section le même opérateur produit que pour l'intrus DH, il correspond en pratique à un produit différent : ici, personne n'est capable de calculer l'inverse m^{-1} d'un message m, alors que c'était possible avec l'opérateur produit sous une exponentielle. En fait, nous avons fait en sorte, lors de la définition des opérateurs produit, exponentielle, et encryption commutative, d'équiper l'opérateur produit avec les propriétés communes aux intrus DH et EC, et de reporter les différences entre ces deux intrus d'une part dans les règle d'intrus elles-mêmes, et d'autre part dans la normalisation de l'opérateur d'encryption commutative. Nous rappelons cette seconde différence :

Simplification de l'inverse

produit sous l'encryption

$$\{t\}^{pc}_{(k^a\cdot(k^*)^b)\bullet M_1} = \{t\}_{k^{a-b}\bullet M_1} \text{ si } a \geq b$$

$$\{t\}^{pc}_{(k^a\cdot(k^*)^b)\bullet M_1} = \{t\}_{(k^*)^{b-a}\bullet M_1} \text{ si } b < a$$

Il s'agit des propriétés de normalisation permettant de rassembler les clefs et leurs inverses dans un produit. Ceci est nécessaire, car on doit formellement séparer la connaissance d'une clef k de la connaissance de son inverse k^*, puisque l'intrus ne peut pas inverser k à la création d'une encryption $\{..\}^{pc}_{..}$. Néanmoins, on ne considère pas ici l'opérateur exponentielle. Ainsi, pour cette section ces règles de normalisation peuvent être indifféremment remplacées par des version plus simples et intuitives :

Simplification de l'inverse

opérateur produit

$$(k^a \cdot (k^*)^b) \bullet M_1 = k^{a-b} \bullet M_1 \text{ si } a \geq b$$

$$(k^a \cdot (k^*)^b) \bullet M_1 = (k^*)^{b-a} \bullet M_1 \text{ si } b < a$$

De plus, les règles de l'intrus EC pour les opérateurs produit et encryption commutative sont assez semblables à celles de l'intrus DH. En effet, la seule différence est que l'on impose aux coefficients choisis par l'intrus d'être positifs :

Règles de décomposition

$$L_{ECd}(\ulcorner\{t_0\}^{pc}_{t_1^{a_1}\cdot\ldots\cdot t_n^{a_n}}\urcorner) : \quad t_0, ..., t_n \rightarrow \ulcorner\{t_0\}^{pc}_{t_1^{a_1}\cdot\ldots\cdot t_n^{a_n}}\urcorner \quad \text{si } \ulcorner\{t_0\}^{pc}_{t_1^{a_1}\cdot\ldots\cdot t_n^{a_n}}\urcorner \neq \{..\}^{pc}_{..}$$
avec $\forall i,\ a_i \in \mathbb{N}$

Règles de composition

$$L_{ECc}(\ulcorner\{t_0\}^{pc}_{t_1^{a_1}\cdot\ldots\cdot t_n^{a_n}}\urcorner) : \quad t_0, ..., t_n \rightarrow \ulcorner\{t_0\}^{pc}_{t_1^{a_1}\cdot\ldots\cdot t_n^{a_n}}\urcorner \quad \text{si } \ulcorner\{t_0\}^{pc}_{t_1^{a_1}\cdot\ldots\cdot t_n^{a_n}}\urcorner = \{..\}^{pc}_{..}$$
avec $\forall i,\ a_i \in \mathbb{N}$

On peut remarquer, si ce n'est pas déjà fait, que l'intrus EC est structurellement très proche de l'intrus DH, mais si en pratique on a l'impression que ce sont des intrus assez différents. En fait, l'intrus EC est suffisamment proche de l'intrus DH pour que nous n'ayons pas besoin de refaire toutes les preuves dans ce cas. Au contraire, nous allons plutôt montrer comment adapter très simplement les preuves du chapitre précédent pour décider (en temps NP) du problème de l'insécurité de protocoles cryptographiques face à l'intrus EC.

En reprenant toutes les notations introduites au chapitre 5 pour l'intrus DH, nous allons examiner (sans trop entrer dans les détails des preuves pour ne pas recopier simplement le chapitre précédent) les trois grandes étapes du chapitre 5. Il s'agit tout d'abord de montrer que l'intrus EC définit un ensemble de règles d'oracles dont on peut décider de l'applicabilité en temps polynomial. Puis il s'agit de borner les tailles DAG des substitutions minimales. Enfin,

il s'agit de montrer l'existence d'attaques minimales à coefficients produit de taille bornée (polynomialement). Il n'y a fondamentalement pas de difficultés majeures à adapter ces preuves à l'intrus EC.

Commençons par regarder les lemmes de remplacement de la section 5.3. Les lemmes 5.3.1.1 et 5.3.2.1 ne travaillent que sur des sous termes, sans prendre en compte des coefficients des produits. Pour le lemme 5.3.3.1, on prend $z_i \in \mathbb{N}$ pour tout i. Ces coefficients ne sont utilisés que pour expliciter les sous termes sur lesquels il s'appliquent (dans un produit), leur valeurs n'interviennent donc pas dans la preuve. Les règles de normalisation supplémentaires n'ont ici pas plus d'influence que la règle $a^p \cdot a^q \to a^{p+q}$ habituelle. Ces trois lemmes sont donc toujours valables pour l'intrus EC. De plus :

Théorème 6.3.0.4 *Les règles de l'intrus EC sont des règles d'oracle.*

On suit ici la structure de la section 5.4. Ici encore, seules preuves où les coefficients des produits apparaissent n'en font pas intervenir les valeurs. De plus, les règles de normalisation supplémentaires n'interviennent pas de manière significative. En particulier, les lemmes 5.4.1.1 et 5.4.1.2, et la proposition 5.4.1.3 qui en découle restent inchangés (Existence de dérivations bien formées). Pour les mêmes raisons, on se contente au lemme 5.4.2.1 d'utiliser $z_i \in \mathbb{N}$ pour tout i (au lieu de $\mathbb{Z}$) : les valeurs elles-mêmes n'interviennent pas, de même que les règles de normalisation supplémentaires. Enfin, le théorème 5.4.2.2 fonctionne aussi bien pour l'intrus EC (il s'appuie essentiellement sur les propriétés précédentes). On a donc bien la propriété désirée : l'intrus EC définit des règles d'oracle. Nous devons également vérifier que le problème de décider si $E \to_{L_{EC}} E, t$ est décidable en temps polynomial :

Proposition 6.3.0.5 *Décision de l'applicabilité pour EC.*
Le problème (d'appartenance à) Application(L_{EC}) est décidable en temps (déterministe) polynomial en $|E, t|_{dag}$.

Preuve. Ici, le signe des coefficients entiers intervient de manière moins triviale qu'ailleurs. Détaillons donc cette courte preuve. On doit donner un algorithme qui, étant donnés E et t normalisés, décide de l'existence de $t', t_1, ..t_n \in E$ et $a_1, ..a_n \in \mathbb{N}$ tels que $t = \ulcorner Exp(t', t_1^{a_1} \cdot .. \cdot t_n^{a_n})\urcorner$. Pour cela, on donne une caractérisation de ce problème : On a $E \to_{L_{DH}} E, t$ ssi :

1. soit $t \neq Exp(\cdot, \cdot)$ et :
   (a) $t \in E$, ou
   (b) il existe M tel que $Exp(t, M) \in E$ et pour tout $m \in M$, $m^* \in E$.
2. soit $t = Exp(v, M)$ et :
   (a) $v \in E$ et $Facteur(M) \subseteq E$, ou
   (b) il existe un produit $M' = m_1^{a_1} \cdot .. \cdot m_n^{a_n}$ tel que

$$Exp(v, M') \in E \text{ et } Facteur(\ulcorner(m_1{}^{*a_1} \cdot .. \cdot m_n{}^{*a_n}) \bullet M\urcorner) \subseteq E$$

   Ainsi, l'intrus doit pouvoir construire la "différence" entre M et M'.

Tous ces points sont simples à vérifier. En particulier, pour 2.(b) on peut examiner chaque $Exp(v, M') \in E$ en calculant les facteurs de la "différence" M et M'. On en déduit donc un algorithme de décision de $E \to_{L_o} E, t$ en temps polynomial en $|E, t|_{dag}$. □

On peut alors examiner la seconde grande étape du chapitre 5 :

Théorème 6.3.0.6 *Les attaques minimales sont bornées en taille DAG.*

Pour toute attaque minimale (π, σ) sur un protocole P (bien formé), on a

$$|\{\sigma(x) \,|\, x \in Var\}|_{dag} \leq 8.|P|_{dag} + 1 \quad avec\ |P|_{dag} = |\mathcal{SP}|_{dag}.$$

Il s'agir ici des sections 5.5 et 5.6. Ces deux sections se concentrent sur les sous termes des substitutions, et non pas aux coefficients entiers. De plus, ici aussi les règles de normalisation supplémentaires se comportent comme $a^p \cdot a^q \rightarrow a^{p+q}$. On se rend ainsi facilement compte que toutes les preuves de ces deux section fonctionnent de la même façon pour l'intrus EC, ce qui nous conduit au théorème précédent. Il nous reste la dernière étape du chapitre 5 :

Théorème 6.3.0.7 *Attaques minimales à coefficients bornés.*

Pour tout protocole $P = (\{R_\iota \Rightarrow S_\iota, \iota \in \mathcal{I}\}, <_{\mathcal{I}}, S_0)$ admettant au moins une attaque face à l'intrus EC, il existe une attaque (π, σ) sur P avec $\|\sigma\|^+$ borné par un polynôme p en $\|P\|$, face à ce même intrus.

Ceci correspond à la section 5.7. Par nature, les coefficients interviennent nettement plus dans cette section (on veut les borner). En contre partie, on ne change pas les sous termes des messages formant l'attaque considérée, ce qui fait que les règles de normalisation supplémentaires n'auront que peu d'importance. La première partie de cette section a pour but de construire et borner des systèmes d'équations linéaires représentant les contraintes devant être vérifiées par les coefficients entiers d'une attaque. L'idée était de borner la taille de tels systèmes pour borner la taille de leurs solutions minimales, et ainsi prouver l'existence d'attaques à coefficients bornés. Avec des coefficients positifs uniquement, il n'y a somme toute que peu de changements à faire. D'une part, on ne doit considérer que des solutions positives aux systèmes d'équations linéaires (Diophantiennes) que l'on considère, c'est à dire des valuations à valeur dans $\mathbb{N}$. Ceci ne change pas l'existence de tels systèmes d'équations, et par nature toutes leurs solutions positives correspondront bien à des messages à coefficients positifs. On fait de même pour les tuple, qui seront ainsi de taille polynomialement bornée (comme pour DH). En fait, la différence essentielle se situe au niveau des plus petites solutions de ces systèmes d'équations linéaires (bornés) : on doit pouvoir les borner. Typiquement, décider si un système d'équations linéaires admet au moins une solution (positive ou négative) est polynomial, alors que décider s'il admet au moins une solution positive est NP (en la taille du système). Cependant, dans les deux cas la taille de la solution minimale (la plus petite des représentations des solutions) est bornée polynomialement par la taille du système d'équations linéaires. Cela nous suffit, car la première partie de l'algorithme est déjà consacrée à choisir une attaque, et donc les coefficients qui la composent. De cette manière, on obtient assez aisément une borne sur les tailles de certaines attaques minimales pour l'intrus EC, comme annoncé plus haut.

Il ne nous reste donc plus qu'à adapter naturellement l'algorithme NP du chapitre 5 à l'intrus EC : il nous suffit de choisir une attaque à coefficients positifs, avec les mêmes bornes que pour l'intrus DH. Comme précédemment, cet algorithme est juste et complet, ce qui donne :

Théorème 6.3.0.8 *L'insécurité modulo EC est NP-complète.*

Le problème de l'insécurité de protocoles cryptographiques (basé sur les opérateurs standards, produit, et encryption commutative) face à l'intrus EC est NP-complet.

En effet, les codages de problèmes 3-SAT du chapitre 3 sont toujours valables. De plus, ici encore le problème de décider si $t \in forge_{L_{EC}}(E)$ est décidable en temps polynomial en $\|E, t\|$.

6.4 Conclusion

Dans ce chapitre, nous avons présenté plusieurs résultats autour de l'ajout de propriétés de commutation de clefs sur certains protocoles cryptographiques (Ping-Pong et modèle par rôles). Nous avons présenté formellement la NP-complétude du problème de l'insécurité avec commutation de clefs dans le cadre des protocoles Ping-Pong. Puis nous avons tenté de montrer quelles modifications (mineures) permettent d'utiliser le résultat de complexité du modèle par rôle face à l'intrus DH pour montrer que ce même problème est NP-complet face à l'intrus EC (avec commutation de clefs). Ainsi, on peut dire que la commutation de clef est par elle-même un problème assez difficile (elle rend la recherche d'attaques NP-complète), mais pas plus que d'autres propriétés comme celles du xor. De plus, elle s'intègre plutôt bien à la méthode de preuve développée jusqu'ici. En outre, on peut facilement généraliser ces résultats à la commutation partielle de clefs, c'est à dire avoir plusieurs groupes de clefs, avec commutation de deux clefs d'un même groupe, mais pas de commutation entre groupes différents. Il suffirait d'utiliser plusieurs opérateurs d'encryption commutative : leur indépendance mutuelle permet d'utiliser les mêmes preuves que pour un seul opérateur.

7

Combinaison de modèles : Modèle par rôles et sessions infinies.

Sommaire

Le but de ce chapitre est d'ajouter au modèle par rôles une approximation de sessions infinies. Plus précisément, il s'agit de modéliser un nombre infini de sessions dont les messages transmis sont de taille bornée, en plus du nombre fini de sessions à taille de messages non bornée déjà présent dans le modèle par rôles. Nous allons donc combiner deux modèles très différents de protocoles cryptographiques, à savoir le modèle de protocoles par rôles à nombres de sessions bornées mais tailles de messages non bornés du chapitre 2, et le modèle de protocoles à nombre de sessions non borné mais tailles de messages bornés de [45]. Pour cela, nous allons en fait étendre les capacités de l'intrus du modèle par rôles de manière à modéliser n'importe quelle session de protocole à messages borné. Comme l'intrus peut utiliser ses règles de déduction un nombre non borné de fois, cela modélisera l'utilisation par l'intrus d'un nombre quelconque de sessions du protocole à messages bornés.

Nous allons montrer que la sécurité de protocoles cryptographiques dans ce modèle "combiné" est dans DEXPTIME. Comme ceci permet au moins de modéliser tous les protocoles à messages bornés et nombre de sessions quelconques de [45], on obtiendra que le problème de l'insécurité de protocoles dans ce modèle "combiné" est DEXPTIME-complet. Nous allons donner la structure générale de ce résultat. Pour plus de détails sur les preuves, on se reportera au rapport technique de [23].

7.1 Le modèle de protocole combiné.

Dans tout ce chapitre, nous n'utiliserons jamais les opérateurs xor, produit, exponentielle et encryption commutative. Ainsi, tous les protocoles et toutes les attaques considérées ici seront toujours définis sans ces opérateurs, et donc sans normalisation. De manière à expliciter les protocoles cryptographiques étudiés dans ce chapitre, nous commençons par définir formellement le modèle "combiné" de protocoles. Il s'agit d'une extension du modèle de protocoles par rôles de la section 2.3.1 contenant deux ensembles de pas de protocoles, correspondant aux deux modèles de protocoles décrits ci-dessus.

Définition 7.1.0.9 *Modèle de protocole combiné.*

Un protocole combiné est un quadruplet $(F_u, F_b, S, \mathcal{D})$ *où* $F_u = (\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}_u\}, <_u)$, $F_b = (\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}_b\}, <_b)$, S *est un ensemble fini de messages avec* $Init \in S$, $\mathcal{D} \in \mathbb{N}$, *et pour tout* $\iota \in \mathcal{I}_u$ *(resp.* $\mathcal{I}_b$*) :*

$$\textit{Pour tout } x \in Var(S_\iota),\ \textit{il existe } \iota' \leq_u \iota \textit{ (resp. } \leq_b \textit{) tel que } x \in Var(R_{\iota'})$$

avec $\leq_u$ *la clôture réflexive de* $<_u$ *(i.e.* $\iota \leq_u \iota'$ *ssi* $\iota <_u \iota'$ *ou* $\iota = \iota'$*), resp.* $\leq_b$ *et* $<_b$*.*

Comme dans les chapitres précédents, *Atomes* et $Variables$ représentent les ensembles d'atomes et de variables apparaissant dans P. La taille DAG d'un protocole est étendue aux protocoles combinés de manière naturelle : si $P = (F_u, F_b, S, \mathcal{D})$, alors $|P|_{dag}$ est le nombre de sous termes distincts de F_u, F_b et S plus $\mathcal{D}$. Typiquement, l'entier $\mathcal{D}$ est la borne sur la taille des messages des pas de protocole pouvant être itérés un nombre non borné de fois. On remarque que la taille DAG de l'ensemble des messages de taille DAG inférieure à $\mathcal{D}$ est exponentielle en $\mathcal{D}$, et donc en $|P|_{dag}$.

L'idée derrière cette définition de protocole combiné est la suivante. Dans une attaque sur P, l'intrus peut utiliser les pas de protocoles de F_u au plus une fois. Cependant, les pas de protocole de F_b peuvent être utilisés aussi souvent que nécessaire (i.e. un nombre non borné de sessions pour F_b). Comme la sécurité de protocoles cryptographiques à nombre de sessions non borné est indécidable (dans le cas général), on restreint les valeurs des variables apparaissant dans F_b aux messages de taille DAG inférieure à $\mathcal{D}$. En revanche, on n'impose aucune restriction sur les variables de F_u. En conséquence, les pas de protocole de F_u sont appelés *non bornés*, et ceux de F_b sont appelés *bornés*.

Nous pouvons à présent définir formellement les attaques dans le modèle de protocole combiné. Celles-ci ne sont pas fondamentalement différentes des attaques dans le modèle par rôle, mais elles permettent un nombre quelconque d'itération des pas de F_b. Pour cela, on appelle instance de F_b pour $\mathcal{D}$, un couple $(\{R_\iota \Rightarrow S_\iota \,|\, \iota \in \mathcal{I}'_b\}, <_b)$ identique à F_b mais où l'on a instancié toutes les variables par des messages de taille DAG inférieure à $\mathcal{D}$ et remplacé tous les indices de pas de protocole I_b par de nouveaux indices I'_b. Ceci nous permet d'utiliser différentes instances de F_b : on note $F_b^{\mathcal{D}}$ l'union de toutes les instances de F_b pour $\mathcal{D}$. Formellement, si $\{(E_i, <_i)\}$ est l'ensemble des instances de F_b pour $\mathcal{D}$, alors $F_b^{\mathcal{D}} = (\bigcup_i E_i, <)$ avec $\iota < \iota'$ ssi $\iota <_u \iota'$ ou $\exists i$, $\iota <_i \iota'$. Nous pouvons alors définir les attaques sur les protocoles combinés comme des attaques sur un nombre non borné d'instances de F_b pour $\mathcal{D}$ (au sens du modèle par rôles) :

Définition 7.1.0.10 *Attaque sur un protocole combiné.*

Soit $P = ((E_u, <_u), F_b, S, \mathcal{D})$ *un protocole combiné. Notons* $F_b^{\mathcal{D}} = (E_b, <'_b)$*. Une attaque* (π, σ) *sur* P *est une attaque sur le protocole (par rôles)* $P' = (E_u \cup E_b, <, S)$ *avec* $\iota < \iota'$ *ssi* $\iota <_u \iota'$ *ou* $\iota <'_b \iota'$*, et face à l'intrus* L_{DY}*.*

avec L_{DY} les règles de l'intrus de Dolev-Yao, cf. chapitre 2. On remarque immédiatement que dans cette définition d'attaque, l'intrus ne peut pas utiliser deux fois le même pas de $F_b^{\mathcal{D}}$. Ceci n'est pas une restriction, car les pas de protocole de $F_b^{\mathcal{D}}$ sont clos : une seconde exécution d'un pas de $F_b^{\mathcal{D}}$ ne peut donner à l'intrus qu'un message déjà reçu à la première exécution de ce pas. Le problème à résoudre est donc le suivant :

$$Insecure := \{\, P \text{ protocole fusionné} \,|\, \text{Il existe une attaque sur } P \,\}$$

Une procédure de décision naïve pour ce problème consiste simplement à tester la sécurité du protocole P' issu de la définition d'attaque. Cependant, ce protocole a naturellement une taille

exponentielle en $|P|_{dag}$, ce qui donnerait un algorithme en temps NEXP, non optimal. Au lieu de cela, on veut un algorithme en temps DEXPTIME, ce qui prouvera le théorème suivant :

Théorème 7.1.0.11 *Le problème Insecure est DEXPTIME-complet.*

7.2 Règles d'oracle.

Pour résoudre ce problème, nous allons étendre les capacités de l'intrus du modèle par rôle de manière à modéliser l'ensemble des pas de protocole $F_b^{\mathcal{D}}$. Pour cela, nous utilisons à nouveau une notion de règles d'oracle, c'est à dire un ensemble de règles d'intrus possédant des propriétés adaptées à ce problème. Même si la démarche est la même que pour les intrus XOR et DH, la définition de règle d'oracle est ici très différente. En effet, au lieu de caractériser la manière dont ces règles créent de nouveaux messages, nous allons en borner la taille.

Définition 7.2.0.12 *Règles d'oracle.*

Soit un intrus $\mathcal{L}$ disposant des règles d'intrus $L_c \cup L_d \cup L_o$ avec $L_c \cup L_d$, et L_o disjoints, et soit P un protocole combiné. Alors $\mathcal{L}$ est un oracle (ou dispose de règles d'oracle) ssi il existe un polynôme $p(..)$ tel que :

1. *Pour tous E et t, si $t \in forge_{\mathcal{L}}(E)$ alors il existe une dérivation bien formée (construite sur $\mathcal{L}$) partant de E et de but t.*
2. *Pour tous E, a et t, si $E \rightarrow_{L_o} E, t$ et $E, t \rightarrow_{L_d(t)} E, t, a$, alors il existe une dérivation D partant de E et de but a telle que $L_d(t) \notin D$.*
3. *Pour toute règle $F \rightarrow t \in L_o$, on a $|t|_{dag} \leq p(|P|_{dag})$ et pour tout $t' \in F$, $|t'|_{dag} \leq p(|P|_{dag})$.*

La première condition, déjà utilisée pour le xor, nous permet de borner la longueur et la nature des dérivations dont on a besoin pour prouver la validité d'une attaque. Les condition 2 et 3 quant à elles vont nous permettre d'effectuer des remplacements d'un terme par un autre plus petit.

7.3 Structure de la preuve

Cette définition d'oracle va nous permettre d'étudier la complexité du problème suivant à la place d'$Insecure$, pour un oracle L bien choisi :

$$Insecure(L) := \{\, P \text{ dans le modèle par rôle} \mid \text{Il existe une attaque sur } P \text{ face à } L \,\}$$

L'idée est de construire, pour tout protocole combiné P, un oracle L et un protocole P' dans le modèle par rôles tels que $|P'|_{dag} \leq |P|_{dag}$ et P admet une attaque ssi P' admet une attaque face à L. Il nous suffira alors de montrer que pour cet oracle L, le problème $Insecure(L)$ est DEXPTIME.

Commençons par borner la taille des substitutions des attaques minimales, pour n'importe quel oracle. Pour cela, on montre un lemme équivalent à 3.4.1.2 mais adapté à notre définition d'oracle :

Lemme 7.3.0.13 *Caractérisation des pré termes de substitutions.*

Pour tout protocole P, pour tout oracle L de polynôme associé $p(..)$, pour toute attaque minimale (π, σ) de P face à L, et pour toute variable $x \in Var$, on a :

$$|\sigma(x)|_{dag} \leq p(|P|_{dag}) \text{ ou il existe } t \sqsubseteq_\sigma \sigma(x) \text{ t.q. } t \in \mathcal{SP}$$

Ceci nous permet de borner les tailles des substitutions d'attaques minimales :

Théorème 7.3.0.14 *Borne sur la taille des substitutions.*

Pour tout protocole P (dans le modèle par rôles) et pour tout oracle L, avec $p(..)$ le polynôme associé, si (π, σ) est une attaque minimale sur P face à L, alors :

$$|\{\, \sigma(x) \mid x \in Var(P) \,\}|_{dag} \leq 3 \cdot p(|P|_{dag})$$

Nous obtenons donc une borne polynomiale sur la taille des substitutions des attaques minimales. Ceci nous permet naturellement d'utiliser le même algorithme qu'au chapitre 3, avec $3 \cdot p(|P|_{dag})$ comme borne sur $|\sigma|_{dag}$. On en fait simplement une version déterministe : pour chaque ordre π et chaque substitution σ (taille polynomiale en $|P|_{dag}$), on teste si l'intrus peut construire tous les messages qu'il doit envoyer. Cependant, à la différence des chapitres précédents, l'oracle que nous utiliserons ne permet pas d'effectuer ces tests en temps polynomial mais nécessite un temps exponentiel en $|P|_{dag}$. On obtient donc un nombre exponentiel (en $|P|_{dag}$) de tests, chacun réalisé en temps exponentiel (en $|P|_{dag}$), d'où un algorithme exponentiel (en $|P|_{dag}$) :

Théorème 7.3.0.15 *Complexité d'$Insecure(L)$.*

Soit L un oracle. S'il existe une procédure pour décider si $E \rightarrow_L t$ en temps exponentiel en $|E, t|_{dag}$, alors il existe un algorithme de décision DEXPTIME pour $Insecure(L)$.

Il nous reste à définir un ensemble de règles d'oracle L permettant d'utiliser ce théorème pour décider $Insecure$. On remarque que l'algorithme pour $Insecure(L)$ n'utilise pas de représentation de L (i.e. la taille des règles de L n'a pas d'importance, de même que la manière dont on les construit). A la place, on doit fournir un algorithme permettant de décider si $E \rightarrow_L t$, pour un ensemble de messages E et un message t quelconques. Nous allons construire l'oracle L en deux temps. Tout d'abord, on remplace chaque pas de protocole $R_\iota \Rightarrow S_\iota$ de $F_b^{\mathcal{D}} = (E_b, <'_b)$ par une règle d'intrus $R_{\iota_1}, .., R_{\iota_n}, R_\iota \rightarrow S_\iota$, avec $\{\iota_i\}_{i=1..n}$ l'ensemble des prédécesseurs de ι, i.e. $\iota_i <_b^+ \iota$ avec $<_b^+$ la clôture transitive de $<_b$. Ceci donne un ensemble de règles d'intrus, que l'on nomme L_{agg} (pour règles agrégées). Malheureusement, cet ensemble de règles ne forme pas un oracle (techniquement, il manque certaines règles). Nous devons donc le compléter :

Définition 7.3.0.16 *Règles d'intrus adaptées à un protocole combiné.*

Soit un protocole combiné P. On pose L_P l'ensemble des règles d'intrus de la forme $E \rightarrow t$ telles que :

- *$t \in forge_{L_{DY} \cup L_{agg}}(E)$,*
- *pour tout $u \in E$, $|u|_{dag} \leq |P|_{dag}{}^2$, et*
- *il existe $R_{\iota_1}, .., R_{\iota_n}, R_\iota \rightarrow S_\iota \in L_{agg}$ telle que $t \in STermes(S_\iota)$.*

On remarque immédiatement que $L_{agg} \subseteq L_P$, puisque pour tout ι on a $|R_\iota|_{dag} \leq |P|_{dag} \leq |P|_{dag}{}^2$. On peut alors montrer que L_P représente toutes les instances de $F_b^{\mathcal{D}}$ et que l'on peut l'utiliser pour le théorème 7.3.0.15 :

Théorème 7.3.0.17 *L_P est bien construit.*

Soit un protocole combiné $P = (F_u, F_b, S, \mathcal{D})$, avec $F_u = (E_u, <_u)$. Alors :

- *L_P est un ensemble de règles d'oracle.*
- *P admet une attaque ssi $P' = (E_u, <_u S)$ admet une attaque face à l'intrus L_P.*
- *Pour tous E et t, on peut décider si $E \rightarrow t \in L_P$ en temps exponentiel en $|E, t|_{dag}$.*

On peut donc utiliser les règles L_P pour décider en temps exponentiel si P admet une attaque, d'où le théorème 7.1.0.11.

8

La recherche d'attaques en pratique : Le logiciel Atsé.

Sommaire

En 2001, le projet Européen AVISS (c.f. [7]) a été créé dans le but de fournir des outils et des méthodes adaptées à la recherche exhaustive, en pratique, d'attaques dans les protocoles cryptographiques. Dans ce but, le projet AVISS s'est appuyé sur la formalisation du problème de décision et sur les méthodes développées pour l'outil CASRUL, c.f. [26, 25, 27, 19], lui-même intégré au projet. Les méthodes ainsi développées permettent d'avoir une procédure de semi-décision du problème de l'insécurité sans aucune contrainte sur la spécification de protocoles : l'outil recherche les attaques en énumérant tous les ordonnancements possibles de pas de protocoles. Dans le cas d'un nombre non borné de sessions, on rappelle que le problème de sécurité est indécidable. Néanmoins, si l'on borne à priori le nombre de sessions à considérer, on obtient naturellement une procédure correcte et complète.

Les méthodes développées pour le projet AVISS procèdent en deux étapes. La première étape consiste en la traduction de la spécification du protocole donnée par l'utilisateur, et écrite dans le modèle Alice-Bob, en une spécification plus facilement utilisable, sous forme de règles de réécriture, et dont l'idée rejoint celle du modèle par rôles. Cette première traduction n'est pas triviale. En effet, on a vu que le modèle Alice-Bob de protocoles cryptographiques comporte de nombreuses lacunes. En particulier, il n'est pas suffisamment précis sur la manière dont les principaux interprètent les messages qu'ils reçoivent. Cette première étape a donc pour objectif de calculer les connaissances effectives des principaux lors d'une exécution du protocole, et d'en déduire une représentation du protocole par règles de réécriture, appelée Format Intermédiaire (ou IF en anglais). Cette description du protocole inclus d'une part la spécification elle-même du protocole, équivalente au modèle par rôles avec contrainte, et d'autre part un ensemble de règles de réécriture décrivant les actions possibles de l'intrus (pour trouver une attaque, ce qui donne un algorithme de semi-décision). La seconde étape de la vérification consiste alors à utiliser l'un

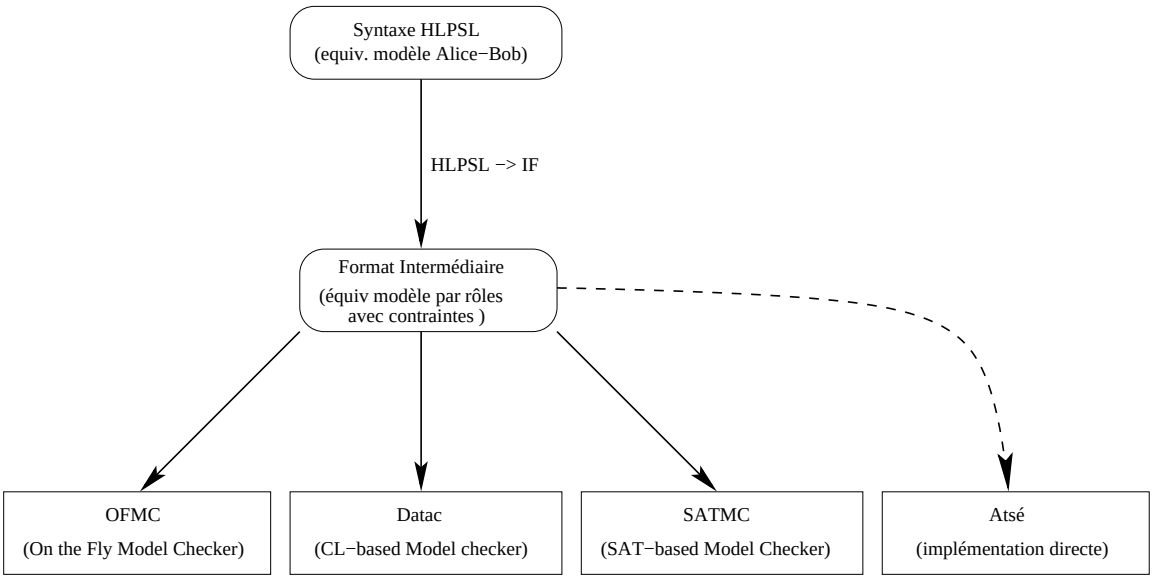


FIG. 8.1: Projet AVISS et Outil Atsé.

des outils du projet (OFMC, DaTac, ou SATMC) pour déterminer si l'ensemble de règles de réécriture obtenu conduit à une attaque ou pas.

L'outil Atsé implémente directement (et optimise) la méthode exhaustive de recherche d'attaques à nombre de sessions borné décrite par le format intermédiaire. A la différence des algorithmes NP des chapitres précédents, nous utilisons ici une représentation symbolique des différents états du protocole (connaissances des principaux et de l'intrus). Cependant, il sera intéressant de remarquer que les représentations symboliques des états du protocole que nous utiliserons seront d'une taille comparable aux attaques minimales des algorithmes NP précédents. Par exemple, nous utiliserons une version symbolique des dérivations bien formées (dérivations construites sur des sous termes des connaissances ou du but, modulo les variables). Le problème, c'est que nous aurons dans le pire des cas un nombre exponentiel de ces représentations. Mais heureusement cette situation ne se produit que rarement sur des protocoles concrets. En résumé, la structure d'AVISS et la place d'Atsé sont décrits à la Figure 8.1.

8.1 L'algorithme.

8.1.1 Les connaissances de l'intrus.

Le coeur de l'outil repose sur la gestion des connaissances de l'intrus. En effet, nous la représenterons par deux types de données. Il s'agira d'une part bien évidement des termes (pas nécessairement clos) reçus par l'intrus de la part des principaux. Mais il s'agira également des termes que l'intrus est supposé avoir envoyé, i.e. des termes attendus par tel ou tel principal à un moment donné de l'exécution du protocole. Ces derniers termes sont considérés comme des hypothèses dépendant des connaissances de l'intrus au moment de l'exécution du protocole où il les a construites.

Pour simplifier la présentation de l'algorithme, nous ne considérerons toujours qu'une seule d'exécution d'un protocole donné. Cependant, en général il faudra considérer toutes les traces

d'exécutions possibles.

Commençons par définir les termes et les substitutions utilisées dans ce chapitre :

$$\begin{aligned} Terme &:= Var \mid Atomes \\ &\quad \mid \langle Terme, Terme\rangle \mid \{Terme\}^s_{Terme} \mid \{Terme\}^p_{Terme} \mid Terme^* \\ Substitution &:= [] \mid [Var \leftarrow Terme].Substitution \end{aligned}$$

On remarque tout d'abord que le langage des termes est étendu pour permettre l'inversion t^* de tout terme t. Par exemple, nous pouvons écrire x^* pour $x \in Var$, si x désigne une clef publique/privée. En contre partie, nous autoriserons l'utilisation de n'importe quelle clef non atomique dans l'encryption asymétrique, puisque l'on peut en dénoter l'inverse (même s'il n'est pas calculable). Pour faciliter l'écriture de l'inverse d'un terme, on notera :

Dans tout ce chapitre : $(t^*)^* = t$ pour tout terme t.

On remarque de plus que les substitutions utilisées peuvent être non closes. Elles représentent en fait les choix de valeurs de variables réalisés par l'intrus. Comme dans le modèle par rôles, une variable représente une connaissance acquise par un principal, i.e. un sous terme d'un message envoyé (et construit) par l'intrus. On peut donc voir une substitution comme une contrainte sur l'ensemble des connaissances possibles de principaux, et on définit l'ensemble des solutions de cette contrainte :

$$\text{Pour tout } \delta \in Substitutions,\quad |[\,\delta\,]| = \{\sigma \in \mathcal{SC} \mid \forall x \in Var,\ x\sigma = (x\delta)\sigma\}$$

avec $\mathcal{SC}$ l'ensemble de toutes les substitutions *closes*. A présent, nous allons définir les différentes contraintes sur les connaissances du l'intrus utilisées par l'algorithme. Commençons par un état élémentaire des connaissances de l'intrus, appelé *Etat* :

$$\begin{aligned} Etat &:= Actions \triangleleft W(LstTermes) \triangleleft Connaissances \\ Actions &:= \epsilon \mid H(Terme) \triangleleft Actions \mid K(Terme) \triangleleft Actions \\ Connaissances &:= \epsilon \mid H(Terme) \triangleleft Connaissances \mid D(Terme) \triangleleft Connaissances \\ LstTermes &:= \text{ensemble de termes (vu comme une liste non ordonnée).} \end{aligned}$$

Examinons cette définition en détail. *Connaissances* représente l'ensemble des connaissances de l'intrus et les contraintes sur ces connaissances à un moment donné de l'exécution du protocole. Les connaissances de l'intrus, au sens du modèle par rôles, sont les éléments $D(t)$. En revanche, les éléments $H(t)$ représentent des hypothèses sur l'ensemble des connaissances de l'intrus. Par exemple, écrire $H(x) \triangleleft D(a) \triangleleft D(b)$ signifie que l'intrus connaît les atomes a et b, et que la valeur $\sigma(x)$ de $x \in Var$ doit vérifier $\sigma(x) \in forge(a, b)$. Cette notation permet d'empiler naturellement plusieurs contraintes : $H(x) \triangleleft D(a) \triangleleft H(y) \triangleleft D(b)$ impose que $\sigma(x) \in forge(a, b)$ et $\sigma(y) \in forge(b)$. Dans cette définition, $D(t)$ représente en fait une connaissance ayant déjà été décomposée par l'intrus, ou n'ayant pas besoin de l'être. Cependant, l'intrus peut recevoir des messages qu'il ne peut pas décomposer immédiatement. La liste de termes $W(L)$ représente ces termes connus de l'intrus mais pas encore décomposés. C'est une sorte de stockage temporaire. Enfin, l'ensemble *Actions* représente la liste des actions futures de l'intrus, i.e. les futurs envois et réception de message, notés $H(t)$ pour un message envoyé par l'intrus et $K(t)$ pour une connaissance acquise par l'intrus (en réponse). Ce sera typiquement la liste des échanges de messages R_i, S_i d'une exécution du protocole suivie par l'intrus. D'un point de vue intuitif, un état $E = A \triangleleft W(L) \triangleleft C \in Etat$ représente un moment d'un exécution d'un protocole, avec C le passé, L le présent et A l'avenir.

L'élément ϵ ne sert qu'à noter un état, une liste d'actions, ou une liste de connaissances vide. On l'omettra lorsqu'il n'y aura pas de confusion possible. De plus, on notera $e \in A$, avec $A \in Actions$, $A \in Connaissances$, ou $A \in Etat$ lorsque l'élément $e = D(t)$, $e = K(t)$ ou $e = H(t)$ apparaît dans la liste A. Pour identifier les connaissances de l'intrus, on pose la notation suivante :

$$\forall E \in Etat, \quad \bar{E} = \{\, t \mid K(t) \in E \text{ ou } D(t) \in E \,\}$$
$$\forall C \in Connaissances, \quad \bar{C} = \{\, t \mid D(t) \in E \,\}$$

C'est l'ensemble des connaissances (non hypothétiques) représentées par un état ou une connaissance. On peut alors définir une version plus restreinte de $forge$, adaptée à ces ensembles de connaissances particuliers :

Définition 8.1.1.1 *Messages créés à partir de connaissances ou d'actions.*
Soient $E = A \triangleleft W(L) \triangleleft C \in Etat$ et σ une substitution close. On note $forge_\sigma(E)$ l'ensemble des messages t tels qu'il existe une dérivation D partant de $\bar{E}\sigma$, de but t, et telle que pour toute règle d'intrus $l \in L_d(t') \cap D$, on a $t' \notin \bar{C}\sigma$ ou $t' \in L\sigma \cup \bar{A}\sigma$.

Ainsi, les éléments $D(..)$ d'une connaissance permettent d'éviter certaines décompositions dans les dérivations considérées (sauf en cas de conflit entre une connaissance $D(t)$ et une connaissance plus récente $K(t)$). On étend naturellement $forge_\sigma$ à $forge_\sigma(C)$, avec $C \in Connaissances$ (i.e. quand A et L sont vides). Il est intéressant de remarquer que les termes créés avec $forge_\sigma(C)$, pour $C \in Connaissances$, le sont uniquement avec des règles d'intrus de composition puisque l'on ne peut décomposer aucun terme de $\bar{C}\sigma$.

Comme pour les substitutions, nous pouvons à présent expliciter l'ensemble des solutions aux contraintes définies dans un état, avec $E \in Etat$ ou $E \in Connaissances$:

$$\begin{array}{ll} & |[\,\epsilon\,]| = \mathcal{SC} \\ \forall t \in Terme, & |[\,H(t) \triangleleft E\,]| = \{\sigma \in \mathcal{SC} \mid t\sigma \in forge_\sigma(E)\} \cap |[\,E\,]| \\ \forall t \in Terme, & |[\,K(t) \triangleleft E\,]| = |[\,D(t) \triangleleft E\,]| = |[\,E\,]| \end{array} \qquad (1)$$

$$\forall L \in LstTermes, \quad |[\,W(L) \triangleleft C\,]| = \left\{ \sigma \in |[\,C\,]| \;\middle|\; \begin{array}{l} \forall \{u\}^p_{v^*} \in L \text{ ou } \{u\}^s_v \in L \\ \text{on a } v \notin forge_\sigma(W(L) \triangleleft C) \end{array} \right\} \qquad (2)$$

Dans (1), on se rend bien compte de l'intérêt des hypothèses : elles génèrent des contraintes sur les valeurs possibles des variables, décrites par un ensemble de propriétés de la forme $t\sigma \in forge_\sigma(C)$. Dans (2), on élimine de $|[\,C\,]|$ toutes les substitutions permettant de décomposer un terme de L. Ainsi, les termes de L sont des termes que l'on n'a pas pu décomposer jusqu'à présent (i.e. avec C) mais que l'on pourra peut être décomposer plus tard.

Nous avons donc défini $Etat$, un ensemble de connaissances et d'hypothèses décrivant un état possible de l'intrus et du protocole. Cet état correspond à différents choix non déterministes réalisés lors de la recherche d'attaques, comme la manière dont l'intrus construit un message donné, la manière dont il décompose un texte chiffré par une clef non atomique, etc... Il nous reste donc à rassembler les différents états de l'intrus pour représenter l'ensemble des états de l'intrus et du protocole accessibles à partir de l'état initial, noté $EIntrus$:

$$EIntrus \;:=\; \bot \mid (Etat, Substitution) \vee EIntrus$$

Le symbole $\vee$ est la disjonction habituelle, et $\bot$ représente "Faux". Ainsi, $EIntrus$ n'est rien d'autre qu'une disjonction de contraintes élémentaires $(Etat, Substitution)$. L'ensemble des solutions de $EIntrus$ est défini de la manière suivante :

Pour tout $C \in Connaissances$, pour toute substitution δ, pour tout $E \in Etat$, et pour toute action (ou connaissance) A, on pose :

$$\begin{array}{rl} \forall I \in EIntrus, & |[\,I\,]| = \bigcup_{i\in\{1,..,n\}} |[\,E_i\,]| \cap |[\,\delta_i\,]| \quad \text{avec } I = (E_1,\delta_1) \vee .. \vee (E_n,\delta_n) \\ \text{et} & |[\,\bot\,]| = \emptyset \end{array}$$

Pour tout i, la substitution (non close) δ_i définit une contrainte sur les variables : toutes les substitutions closes σ solutions de (E_i, δ_i) doivent être compatibles avec δ_i. Le but de l'outil sera de décomposer les connaissances et hypothèses décrites par $EIntrus$, à l'aide de règles de réécriture, pour décider si l'ensemble des solutions correspondantes est vide ou pas, i.e. s'il existe au moins une substitution close satisfaisant les contraintes décrites. On saura alors s'il existe une attaque ou si le protocole donné est sûr. En effet, on a par construction de $|[..]|$:

Proposition 8.1.1.2 *Lien avec le modèle par rôles.*
Soit $P = (\{R_\iota \Rightarrow S_\iota, \mathcal{I}\}, <_\mathcal{I}, S_0)$ un protocole bien formé, et π un ordre d'exécution sur P vu comme une bijection de $\mathcal{J} \subset \mathcal{I}$ dans $\{1,..,n\}$. Pour toute substitution close σ, (π,σ) est une attaque sur P ssi :

$$\sigma \in |[(K(S_n) \lhd H(R_n) \lhd .. \lhd K(S_1) \lhd H(R_1) \lhd K(S_0) \lhd W(\emptyset),\ [])]|$$

avec $[]$ la substitution identité, i.e. pour tout $x \in Var$, $x[] = x$.

On suppose donc, comme décrit au début de la section, qu'un protocole et un ordre d'exécution ont été choisis. Le point de départ de l'algorithme, i.e. l'état initial, est naturellement de la forme :

$$(K(S_n) \lhd H(R_n) \lhd .. \lhd K(S_1) \lhd H(R_1) \lhd K(S_0) \lhd W(\emptyset) \lhd \epsilon,\ [])$$

L'algorithme fonctionne par décomposition d'hypothèses et de connaissances. Il assure l'invariance des quatre propriétés suivantes, pour une disjonction $I = (E_1,\delta_1) \vee .. \vee (E_p,\delta_p) \in EIntrus$ avec $E_i = A_i \lhd W(L_i) \lhd C_i$ pour tout i :

A) Si $A_i = e_1 \lhd .. \lhd e_n$ et $C_i = e_{n+1} \lhd .. \lhd e_p$, alors pour toute variable x de E_i, il existe $j \in \{1,..,p\}$ tel que $e_j = H(t)$, $x \in Var(t)$, et $x \notin Var(e_{j+1},..,e_p)$[19].

B) $|[A_i \lhd K(t_1) \lhd ... \lhd K(t_n) \lhd C_i]| \subseteq |[I]|$, avec $L_i = \{t_1,..,t_n\}$.

C) $L_i \subseteq \bar{C}_i$ et pour tout $t \in L_i$, t est de la forme $\{..\}_{..}^{s \text{ ou } p}$.

D) La substitution δ_i est idempotente, i.e. $(x\delta_i)\delta_i = x\delta_i$ pour tout $x \in Var$.

La propriété A) assure que toutes les variables utilisées dans une connaissance $D(..)$ ou $K(..)$ représentent des messages créés par l'intrus à un moment antérieur de l'exécution, c'est à dire que pour chaque variable x utilisée par un principal, il existe une hypothèse $H(x) \lhd C$ avec $x \notin C$. La propriété B) sera utilisée pour tester facilement si $|[I]| = \emptyset$. La propriété C) donne simplement la structure des listes L_i : elles ne contiennent que des encryptions déjà présentes dans $\bar{C}_i$. Enfin, la propriété D) évite simplement d'avoir des substitutions incohérentes. On constate aisément que ces quatre propriétés A), B), C) et D) sont vérifiées par l'état initial de la propriété 8.1.1.2. En particulier, la propriété A) nécessite que le protocole considéré soit bien formé.

Par ailleurs, le but des décompositions que nous allons décrire est d'atteindre une disjonction de contraintes dite décomposée, et définie comme suit :

[19] avec bien sûr $Var(H(t)) = Var(D(t)) = Var(t)$ par définition, pour tout t.

Définition 8.1.1.3 *Contraintes décomposées.*

Soit $I = (E_1, \delta_1) \vee .. \vee (E_p, \delta_p) \in EIntrus$, avec $E_i = A_i \triangleleft W(L_i) \triangleleft C_i$ pour tout i. On dit que I est décomposé quand :

1. *Pour tout i, si $H(t) \in C_i$ alors $t \in Var$ et $\delta_i(t) = t$.*
2. *Pour tout i, $A_i = \epsilon$.*

En effet, décider si une disjonction de contraintes décomposée vérifiant la propriété B) admet au moins une solution est trivial. Si elle est réduite à $\perp$, alors évidement il n'y a aucune solution. Sinon, elle contient au moins un couple $(W(L) \triangleleft C, \delta)$. Posons σ telle que pour tout $x \in Var$, $x\sigma = (x\delta)[y \leftarrow a \,|\, y \in Var]$, avec $a \in S_0$. C'est une instance possible de δ. On remarque alors que toutes les contraintes sur les variables libres de δ sont de la forme $x\sigma \in forge_\sigma(C)$, i.e. $a \in forge_\sigma(C)$ avec $D(a) \in C$ nécessairement (point 1 ci-dessus). Ce sont les seules contraintes de $K(t_1) \triangleleft .. \triangleleft K(t_n) \triangleleft C$, et elles sont toutes satisfaites. On a donc au moins une solution dans $|[(K(t_1) \triangleleft .. \triangleleft K(t_n) \triangleleft C, \delta)]|$, et donc dans $|[(W(L) \triangleleft C, \delta)]|$ grâce à la propriété B).

8.1.2 Règles de décomposition d'hypothèses.

Nous allons tout d'abord décrire les règles de décomposition d'hypothèses, pour une connaissance ou un état de l'intrus. Pour cela, nous aurons besoin de calculer une solution à un problème d'unification de deux termes. On définit donc :

Définition 8.1.2.1 *Solutions d'un problème d'unification.*

Soient deux termes t_1 et t_2, et soit δ une substitution. Posons $Var = \{x_1, .., x_n\}$. Alors on note $\delta' = Unif_\delta(t_1, t_2)$ un unificateur principal de $\langle t_1, x_1, .., x_n\rangle$ et $\langle t_2, x_1\delta, .., x_n\delta\rangle$ tel que $(x_i\delta')\delta' = x_i\delta'$, pour tout i.

Les règles de réécritures sur les disjonctions de contraintes relatives à la décomposition d'hypothèses sont décrites dans la Table 8.1 ($A \in Actions$, $L \in LstTermes$, et $C \in Connaissances$). De manière à satisfaire à la propriété D), on suppose que l'unificateur principal $\delta' = Unif_\delta(t_1, t_2)$ est clos par lui-même, i.e. pour tout $x \in Var$, $x\delta' = (x\delta')\delta'$.

On vérifie assez simplement que l'ensemble des solutions reste inchangé par applications des règles de décomposition d'hypothèses :

Proposition 8.1.2.2 *Correction & Complétude.*

Pour tous intrus I et I' tels que I vérifie les propriétés A) à D) et tels que $I \rightarrow I'$ par une règle de la Table 8.1, on a $|[I]| = |[I']|$.

Preuve. Les règles 2 à 5 de la Table 8.1 ne sont que des décompositions conditionnées par la nature de l'hypothèse. Elles suivent exactement les différents moyens dont dispose l'intrus pour créer le terme en question, et vérifie donc naturellement la proposition. En particulier, le point 5 ne fait que réaliser une partition (peut-être avec recoupements) des différentes substitutions closes : leur union reste inchangée.

Pour la règle 1, il nous suffit d'utiliser la propriété C) et le point (2) de la définition de $|[..]|$. En effet, ces deux propriétés donnent respectivement $L \subseteq \bar{C}$ et aucun terme de L ne peut être décomposé à partir de $L \cup \bar{C} = \bar{C}$. En conséquence, on a nécessairement $forge_\sigma(W(L) \triangleleft C) = forge_\sigma(C)$. □

Il nous reste alors à vérifier que les règles de décomposition d'hypothèses respectent les propriétés A) à D) :

1.	$(A \lhd H(t) \lhd W(L) \lhd C,\, \delta)$	$\rightarrow$	$(A \lhd W(L) \lhd H(t\delta) \lhd C,\, \delta)$
2.	Si $H(t) \in C$ ou $D(t) \in C$, $H(t) \lhd C$	$\rightarrow$	C
3.	$H(\langle a, b\rangle) \lhd C$	$\rightarrow$	$H(a) \lhd H(b) \lhd C$
4.	Si $D(a) \notin C$ avec $a \in Atomes$,		
	$(A \lhd W(L) \lhd C' \lhd H(a) \lhd C,\, \delta)$	$\rightarrow$	$\bot$
5a.	Si $D(t) \notin C$ avec $t\delta = \{a\}^s_b$ ou $t\delta = \{a\}^p_{b^*}$,		
	$(A \lhd W(L) \lhd C' \lhd H(t) \lhd C,\, \delta)$	$\rightarrow$	$(A \lhd W(L) \lhd C' \lhd H(a) \lhd H(b) \lhd C,\, \delta)$ $\vee \bigvee_{c \in \bar{C}} (A \lhd W(L\delta_c) \lhd C'\delta_c \lhd C\delta_c,\, \delta_c)$
5b.	Sinon, $(A \lhd W(L) \lhd C' \lhd H(t) \lhd C,\, \delta)$	$\rightarrow$	$\vee \bigvee_{c \in \bar{C}} (A \lhd W(L\delta_c) \lhd C'\delta_c \lhd C\delta_c,\, \delta_c)$
			avec $\delta_c = Unif_\delta(t, c)$

TAB. 8.1: Décompositions d'hypothèses.

$$
\begin{array}{lcl}
ForgeWith(x,\, t_{ref},\, \delta,\, h) & = & \emptyset \quad \text{si } x \in Var \\
ForgeWith(t,\, t_{ref},\, \delta,\, h) & = & \{\, (Unif_{\delta}(t, t_{ref}), h)\,\} \\
 & \cup & ForgeWith(a,\, t_{ref},\, \delta,\, h \cup \{b\}) \\
 & \cup & ForgeWith(b,\, t_{ref},\, \delta,\, h \cup \{a\}) \\
 & & \text{pour } t = \{a\}_b^{s \text{ ou } p} \text{ ou } t = \langle a, b \rangle \\
ForgeWith(t,\, t_{ref},\, \delta,\, h) & = & \{\, (Unif_{\delta}(t, t_{ref}), h)\,\} \quad \text{sinon.}
\end{array}
$$

TAB. 8.2: Définition de $ForgeWith(...)$

Proposition 8.1.2.3 *Invariance des prop. A) à D)*

Pour tous intrus I et I' tels que $I \rightarrow I'$ par une règle de la Table 8.1, si I vérifie les propriétés A) à D) alors I' aussi.

PREUVE. On a vu que D) est vraie par construction. La propriété A) quant à elle n'est qu'un ordre sur les variables dans I et I', qui est naturellement conservé par application d'une substitution. De plus, la propriété C) est naturellement vérifiée puisque L reste inchangé et C croit strictement. Il ne nous reste plus que la propriété B) : pour les règles 2 à 5, elle est vérifiée par I' exactement pour les mêmes raisons que $|[I]| = |[I']|$ dans la preuve précédente. Et pour la règle 1, il suffit de remarquer que $|[(A \triangleleft K(t_1) .. \triangleleft K(t_n) \triangleleft H(t\delta) \triangleleft C,\, \delta)]| \subseteq |[(A \triangleleft H(t) \triangleleft K(t_1) .. \triangleleft K(t_n) \triangleleft C,\, \delta)]|$, avec $L = \{t_1, .., t_n\}$. □

8.1.3 Règles de décomposition de connaissances

Nous allons à présent décrire les règles de réécriture nous permettant de décomposer les connaissances $K(..)$ et $D(..)$. Pour cela, nous avons tout d'abord besoin de construire un ensemble représentant toutes les façons de construite un terme t donné en utilisant au moins une fois un autre terme t_{ref} donné. La construction de cet ensemble, nommé $ForgeWith(t, t_{ref}, \delta, h)$, est décrite dans la Table 8.2 et sera appelé avec $h = \emptyset$. C'est un ensemble de couples (δ', h) avec δ' une substitution compatible avec δ (i.e. $\delta' = \delta\delta'$), et h un ensemble de nouvelles hypothèses, i.e. un ensemble de termes que l'intrus doit construire pour obtenir t en utilisant au moins une fois t_{ref}. On se rend compte aisément que par construction, $ForgeWith(t, t_{ref}, \delta, \emptyset)$ décrit un ensemble contenant toutes les manières dont l'intrus peut créer t à partir de $\bar{C}$ (avec des règles d'intrus de composition) en utilisant au moins une fois t_{ref}, et en respectant la substitution δ. Le but de cette construction sera de décomposer les termes présents dans $W(L)$, avec $L \in LstTermes$, quand une nouvelle connaissance apparaîtra. En effet, L représente ici un ensemble de termes que l'on n'a pas réussi à décomposer pour le moment, mais que l'on a gardés pour les décomposer plus tard. Cependant, pour ne pas refaire des décompositions déjà réalisées par le passé, on devra utiliser au moins une fois la nouvelle connaissance dans la décomposition d'un terme de L. D'où la construction de $ForgeWith$.

Nous pouvons maintenant présenter les règles de déduction de connaissances, décrites par la Table 8.3. Pour faciliter les notations, nous avons factorisé ces règles en étendant la syntaxe des états :

$$Etat' \;:=\; Etat \mid Actions \triangleleft T(Terme) \triangleleft W(LstTermes) \triangleleft Connaissances$$

1. Si $t\delta \in \bar{C}$ ou $t\delta \in Var$,
$$K(t) \lhd W(L) \lhd C \quad \rightarrow \quad W(L) \lhd C$$
$$D(t) \lhd C \quad \rightarrow \quad C$$

2. Si $t\delta = \langle a, b \rangle$,
$$K(t) \lhd W(L) \lhd C \quad \rightarrow \quad K(a) \lhd K(b) \lhd W(L) \lhd C$$

3. Si $t\delta = \{a\}_b^s$ ou $t\delta = \{a\}_{b^*}^p$,
$$(A \lhd K(t) \lhd W(L) \lhd C,\, \delta) \quad \rightarrow_3 \quad (A \lhd K(a) \lhd H(b) \lhd T(t\delta) \lhd W(L) \lhd C,\, \delta)$$
$$\vee (A \lhd T(t\delta) \lhd W(L \cup \{t\delta\}) \lhd C,\, \delta)$$

4. Sinon,
$$(A \lhd K(t) \lhd W(L) \lhd C,\, \delta) \quad \rightarrow_4 \quad (A \lhd T(t\delta) \lhd W(L) \lhd C,\, \delta)$$

5.
$$(E,\, \delta) \quad \rightarrow_T \quad \bigvee_{(E,\delta) \rightharpoonup^! (E',\delta')} (E',\, \delta')$$

Et pour tous :

$t \in L$ avec $t = \{a\}_b^s$ ou $t = \{a\}_{b^*}^p$, et $(\delta',\, h) \in ForgeWith(b,\, t_{ref},\, \delta,\, \emptyset)$

On a :
$$(A \lhd T(t_{ref}) \lhd W(L) \lhd C,\, \delta) \quad \rightharpoonup \quad (A \lhd K(a) \lhd H(h) \lhd T(t_{ref}) \lhd W(L \backslash t) \lhd C,\, \delta')$$
$$\vee (A \lhd W(L) \lhd D(t_{ref}) \lhd C,\, \delta)$$

avec la notation $H(\{h_1, .., h_q\}) = H(h_1) \lhd .. \lhd H(h_q)$

TAB. 8.3: Règles de décomposition de connaissances.

Cependant, cette nouvelle définition d'état n'est utilisée que de manière temporaire, pour les règles $\rightharpoonup$. On pose $(E,\, \delta) \rightharpoonup^{!} (E',\, \delta')$ ssi $(E,\, \delta) \rightharpoonup^{*} (E',\, \delta')$ et aucune règle $\rightharpoonup$ n'est applicable sur $(E',\, \delta')$. Chaque règle $(A \triangleleft T(t_{ref}) \triangleleft W(L) \triangleleft C,\, \delta) \rightharpoonup (E',\, \delta')$ représente une manière de décomposer un terme de L en utilisant un moins une fois t_{ref}. Itérées, elles permettent d'éliminer l'élément $T(..)$ (et donc d'obtenir des contraintes dans $Etat$) : si $(E,\, \delta) \rightharpoonup^{!} (E',\, \delta')$, alors $E' \in Etat$. En conséquence, on pose :

$$\begin{array}{|c|}\hline (E,\, \delta) \;\rightarrow\; (E'_1, \delta'_1) \vee .. \vee (E'_n, \delta'_n) \\ \text{si} \quad (E, \delta) \;\rightarrow_{3 \text{ ou } 4} \rightarrow^{!}_{T}\; (E'_1, \delta'_1) \vee .. \vee (E'_n, \delta'_n) \\ \hline \end{array}$$

Cette présentation de la décomposition de $W(L)$ est loin d'être optimale, notamment par le nombre important de cas redondants qu'elle engendre. En particulier, l'ordre d'élimination des termes de $W(L)$ n'a en réalité aucune importance, et il est donc inutile de tous les tester. Nous discuterons plus loin de quelques autres optimisations réalisées en pratique.

Descriptions des règles de la Table 8.3 :

1. La règle 1 permet d'éliminer une connaissance $K(t)$ ou $D(t)$ déjà présente dans la liste des connaissances C, donc inutile. De plus, cette règle élimine également les connaissances $K(t)$ ou $D(t)$ quand $t\delta = x \in Var$. Ces connaissances sont aussi inutile, car on sait qu'il existe $H(x) \in C$, i.e. la valeur de x est construite (donc déjà connue) par l'intrus.

2. La règle 2 traite le cas d'un couple. Ce cas est très simple, car connaître un couple $\langle a, b\rangle$ ou a et b est strictement équivalent.

3. La règle 3 traite le cas d'une connaissance $t\delta$ de la forme $\{a\}^{s}_{b}$ ou $\{a\}^{p}_{b^*}$. Dans ce cas, on ne peut pas savoir à l'avance si l'intrus sera capable de décomposer effectivement le terme $t\delta$. En fait, on réalise ici une partition (peut-être avec recoupement) de l'ensemble des substitutions closes satisfaisant le couple $(E,\, \delta)$ considéré, selon qu'elles permettent à l'intrus de calculer la clef inverse de $t\delta$ ou pas. Si c'est possible, l'intrus obtient $K(a)$ et doit satisfaire l'hypothèse $H(b)$. Si ce n'est pas possible, l'intrus ajoute $t\delta$ à la liste L des termes pouvant peut-être être décomposés plus tard.

   Optimisations importantes : Si $t\delta$ est une encryption symétrique $\{a\}^{s}_{b}$, alors il n'est pas nécessaire d'ajouter $T(t\delta)$ dans le cas où $t\delta$ est décomposé. En effet, connaissant a et b, l'intrus peut très bien reconstruire $t\delta$. Ceci évite une application de $\rightarrow^{!}_{T}$, ce qui n'est pas négligeable vu le nombre de sous cas qu'il peut générer. Par ailleurs, on remarque que cette règle est appliquée même si $b \in Synth(\bar{C})$, avec $Synth(E) = \{t \mid E \rightarrow^{*}_{L_c} E', t\}$ (selon la notation de Paulson). Comme ceci est facile à tester, on peut ajouter une règle donnant $(A \triangleleft K(a) \triangleleft T(t\delta) \triangleleft W(L) \triangleleft C,\, \delta)$ dans ce cas, ce qui génère moins de cas que la règle 3.

4. La règle 4 termine le filtrage en traitant toutes les autres connaissances, i.e. les atomes $t \in Atomes$ et les inverses $t = ..^{*}$ non présents dans C. Dans ces deux cas, l'intrus n'a aucun moyen de décomposer $t\delta$, et on passe donc directement à $T(..)$ pour décomposer L.

5. Les règles d'élimination de $T(..)$ sont les plus compliquée de toutes. Leurs but est de générer toutes les décompositions possibles des termes de L en utilisant au moins une fois la nouvelle connaissance t. En effet, décomposer un terme de L sans utiliser t n'a aucun intérêt puisque l'on a déjà tenté cette décomposition par le passé. Concrètement,

la propriété (2) de la définition de $|[..]|$ est là pour formaliser de ce fait. Cependant, on ne sait pas à l'avance quels termes de L l'intrus sera capable de décomposer, et on doit donc successivement énumérer tous les cas. On choisi donc $t \in L$ et (δ', h), c'est à dire un terme de L dont $ForgeWith$ a donné au moins une solution potentielle, i.e. une manière pour l'intrus de calculer la clef inverse de t : δ' est une substitution plus restreinte que δ et h est un ensemble d'hypothèses supplémentaires, i.e. un ensemble de termes devant être construits par l'intrus. Pour chaque $t \in L$ et (δ', h), l'intrus obtient le résultat des décompositions des termes t_i (i.e. $K(a)$), mais doit satisfaire les nouvelles hypothèses h. Enfin, on retire de $W(L)$ le terme t que l'on vient de décomposer (inutile de le décomposer à nouveau par la suite). Il est intéressant de remarquer que même si ces règles peuvent potentiellement générer beaucoup de cas, d'une part on ne considère que les solutions potentielles raisonnables, i.e. des solutions où un morceau de la clef à construite est unifiable avec la nouvelle connaissance, et d'autre part on va rapidement éliminer beaucoup de ces solutions potentielles, notamment quand les hypothèses supplémentaires sont trivialement insatisfiables. En pratique, ceci n'explose en nombre de cas que très rarement.

On se rend compte assez facilement que toute application d'une règles 1 à 4 de la Table 8.3 respecte la propriétés A). Pour cela, il suffit d'examiner les règles une par une : on ne change pas la première apparition d'une variable dans une hypothèse, et on n'introduit aucune nouvelle variable. De la même manière, ces règles respectent également la propriété C) : d'une part, on n'ajoute un terme $t\delta$ à $W(L)$ que quand on ajoute aussi $D(t\delta)$. D'autre part, tous les termes ajoutés à $W(L)$ sont des encryptions, symétriques ou asymétriques. Pour la propriété B) on constate assez facilement que pour chaque règle de déduction de connaissances de la forme $(E, \delta) \rightarrow (E_1, \delta_1) \vee ... \vee (E_n, \delta_n)$, on a $\forall i$, $|[(E'_i, \delta_i)]| \subseteq |[(E', \delta)]|$ avec $E' = A \triangleleft K(t_1) \triangleleft .. \triangleleft K(t_p) \triangleleft C$ si $E = A \triangleleft W(L) \triangleleft C$ et $L = \{t_1, .., t_p\}$ (idem pour E_i). Enfin, la propriété D) est satisfaite par construction.

En revanche, montrer que ces règles respectent l'ensemble des solutions $|[..]|$ définie au début de ce chapitre est un peu plus technique. Des preuves complètes dans un système moins optimisé mais fondamentalement équivalent ont été publiées par Y. Chevalier et L. Vigneron dans [26, 25].

En résumé, on peut prouver que :

Proposition 8.1.3.1 *Correction et Complétude.*

Soient deux disjonctions de contraintes I et I' telles que $I \rightarrow I'$ avec I vérifiant les propriétés A) à D). Alors I' vérifie ces mêmes propriétés, et $|[I]| = |[I']|$.

Ainsi, on a définit des ensembles de règles de décomposition de connaissances et d'hypothèses respectant l'ensemble de solutions $|[..]|$ donné au début du chapitre. De plus, on constate aisément que par construction, tout intrus sur lequel aucune de ces règles n'est applicable est nécessairement décomposé. En particulier, toutes les hypothèses sont des variables, et il ne reste plus aucune action à traiter. Arrivé à ce point, on a vu que l'on peut décider trivialement si $|[..]|$ est vide ou non. En outre, il suffit de mesurer le nombre de variables non instanciées et la taille DAG des actions et des connaissances pour constater que ce système de réécriture termine. On a donc bien une procédure de décision pour le problème de l'insécurité de protocoles, en pratique et avec une représentation symbolique des attaques.

8.1.4 Optimisations

Nous avons présenté l'essentiel de la recherche d'attaques telle qu'elle est implémentée dans l'outil Atsé. L'un des avantages de ces règles est de permettre un certain nombre d'optimisations

très intéressantes en pratiques pour avoir un outil efficace.

L'optimisation la plus importante concerne la fonction $ForgeWith(..)$ et les règles $\rightarrow_T^!$. En effet, certaines connaissances $\{a\}_b^s$ ou $\{a\}_{b^*}^p$ de $W(L)$ peuvent être trivialement décomposées, i.e. sans unification supplémentaire, et il est donc inutile de considérer le cas où l'intrus ne peut pas les décomposer. On peut identifier ces connaissances particulières de la manière suivante. Reprenant les notations des $(A \triangleleft T(t) \triangleleft W(L) \triangleleft C, \delta)$, posons $\{r_1, .., r_n\}$ l'ensemble des termes $\{a\}_b^s$ ou $\{a\}_{b^*}^p$ de L tels que $ForgeWith(b, t, \delta, \emptyset)$ contient au moins un couple (δ', h) avec $\delta' = \delta$ et $h \subseteq \bar{C}$. Ce sont donc des termes de L décomposables sans substitution ni hypothèse supplémentaire. On peut alors commencer par décomposer arbitrairement ces termes, c'est à dire considérer éliminer les r_i de L' (i.e. $L' = L \backslash \{r_1, .., r_n\}$, et ajouter $K(a'_1) \triangleleft .. \triangleleft K(a'_n)$ à l'état, avec $r_i = \{a'_i\}_{..}^{s \text{ ou } p}$ pour tout i.

Une seconde optimisation réalisée par Atsé sur ces règles consiste à remarquer qu'il est totalement inutile de parcourir la totalité de la liste pour déterminer le point d'application de la prochaine règle de réécriture. En effet, on se contente tantôt de décomposer des sous termes d'une connaissance ou d'une hypothèse que l'on vient de décomposer, tantôt de passer à la connaissance ou à l'hypothèse suivante. On peut donc éviter de parcourir toute la liste d'actions ou de connaissances à chaque pas à la recherche d'un terme à décomposer. Dans Atsé, ces décompositions d'hypothèse et de connaissances sont réalisées par deux fonction d'ajout d'une hypothèse ou d'une connaissances sur un état de l'intrus, et rendent directement un état de l'intrus entièrement décomposé (les règles de décompositions sont itérées en un seul pas).

On peut également remarquer que dupliquer les actions (presque) à chaque pas n'est pas très heureux. Un outil concret peut factoriser les actions issues des pas de protocole, et ainsi ajouter une même hypothèse ou une même connaissance à plusieurs états de l'intrus en même temps. Un peu de la même manière, on peut aussi n'appliquer les substitutions que par nécessité, en marquant les connaissances déjà instanciées.

8.2 Exemples

Pour donner une idée du comportement de l'outil, la Table 8.4 présente quelques protocoles assez classiques vérifiés avec Atsé. Ces protocoles sont issus de la librairie de J.Clark et J.Jacob (c.f. [28]). Ils doivent vérifier deux types de propriétés : soit le secret d'un message donné (Secret, idem sections précédentes), soit l'authentification d'un principal par un autre sur une connaissance commune. On peut remarque dans cette table que certaines attaques exploitent des confusions de type entre différents opérateurs. La table 8.5 présente les protocoles dont le comportement change quand on interdit les attaques de confusion de type avec un couple.

Nom du protocole	Nb sess	Temps	Type	Description
Andrew	1	$< 0,01s$	Auth.	Confusion de types : un message remplace un nonce.
Denning-Sacco, clef symétrique	2	$< 0,01s$	Auth.	Confusion de types : la clef générée par le serveur est remplacée par un couple.
ISO, clef publique ou symétrique	2, seq.	$< 0,01s$	Auth.	Attaque par rejeu : détournement de session.
Needham-Schröder	2, par.	$< 0,01s$	Auth.	Détournement de messages entre sessions.
Needham-Schröder-Lowe	2, par.	$< 0,01s$	Auth.	Confusion de types : un couple remplace un nonce.
NSPCK, 2 clients & 1 serveur génération d'une clef de session.	1	$< 0,01s$	Secret	Confusion de types : un couple remplace un nonce. Le secret est divulgué à la place d'une autre donnée.
NSPCK, avec authentification au lieu de secret de la clef.	2, par.	$0,02s$	Auth.	L'intrus utilise une session pour en attaquer un autre.
Neumann	1	$< 0,01s$	Secret	Confusion de types : une clef remplace un nonce.
Otway-Rees	1	$< 0,01s$	Secret	Confusion de types : un triplet remplace la clef.
SET, Card Holder Registration.	2, seq.	$0,01s$	Auth.	Rejeu, avec risque d'attaque de dénie de service. Un champ optionnel résout ce problème.
Stubblebine, 2 clients, 1 serveur génération d'une clef de session.	1	$< 0,01s$	Secret	Confusion de types : un nonce connu remplace la clef de session.
TMN Auth. avec un tiers de confiance.	1	$< 0,01s$	Auth.	Le serveur est utilisé comme oracle pour calculer une clef générée par un agent honnête.

TAB. 8.4: Protocoles vérifiés avec Atsé, avec confusion de types.

Nom du protocole	**Nb sess.**	**Temps**	**Type**	**Description**
Denning-Sacco (clef symétrique)	2	$< 0,01s$	Auth.	Confusion entre sessions.
Needham-Schröder-Lowe	8	$2,46s$	–	Aucune attaque pour 8 sessions.
NSPCK, 2 clients & 1 serveur Le serveur crée une clef de session.	2	$< 0,01s$	Auth.	Confusion entre sessions.
Otway-Rees	5	$< 9,55s$	–	Aucune attaque pour 5 sessions.

TAB. 8.5: Protocoles vérifiés avec Atsé, avec confusion de types.

9

Conclusion et perspectives

La volonté naïve de vérifier tous les protocoles cryptographiques possibles se heurtera malheureusement toujours à l'indécidabilité de ce problème de décision. Cependant, plusieurs classes de protocoles rendant ce problème décidable ont été proposées. D'une part, nous avons apporté notre contribution à ce problème sur le plan théorique par la définition d'un modèle adapté au nombre fini de sessions et surtout l'étude de la décidabilité et de la complexité de la vérification de protocoles, face aux propriétés algébriques des opérateurs xor, exponentiation et encryption commutative, ainsi que pour un modèle comportant des sessions en nombre infini. D'autre part, nous avons apporté notre contribution sur le plan concret en créant l'outil Atsé de vérification de protocoles.

Modèle de protocole adapté au cas d'un nombre fini de sessions :

Nous avons défini un modèle de protocoles cryptographiques utilisé adapté à la plupart des preuves de complexité de ce travail (modèle par rôles). Ce modèle de protocoles cryptographiques a été très utile pour toutes les preuves de complexité des chapitres 3 à 7, notamment parce qu'il donne une représentation compacte et directe de chaque pas de protocole, et évite de spécifier individuellement chaque action réalisé par un principal pour traiter un message. De plus, ce modèle est une représentation intuitive du comportement d'un protocole, facile à comprendre. Enfin, ce modèle permet plusieurs extension, dont notamment l'ajout de filtrage de connaissances à chaque pas de protocole et la modélisation de différents canaux de communication.

Complexité de la vérification avec et sans opérateurs algébriques :

Nous avons présenté différents résultats originaux de décidabilité (et de complexité) du problème de l'insécurité de protocoles cryptographiques à nombre de sessions borné. En particulier, nous avons montré que ce problème est NP-complet pour des protocoles n'utilisant pas d'opérateur algébrique. Puis nous avons successivement montré que ce problème reste NP-complet quand on ajoute les opérateurs "ou exclusif" et "exponentiation", avec leurs propriétés algébriques. Dans tous les cas, nous avons ajouté de nouvelles règles de déduction pour l'intrus lui permettant d'utiliser ces opérateurs (et leurs propriétés). En fait, nous avons à chaque fois défini un ensemble de règles d'oracles plus générales (contenant ces règles dédiées aux opérateurs algébriques), et montré que le problème de l'insécurité de protocoles cryptographiques est NP-complet face à tout intrus étendu par des règles d'oracle. Ceci nous a permis par exemple de montrer à moindre coût que l'ajout de règles Préfixes conserve la NP-complétude de ce problème.

Complexité de la vérification avec un opérateur d'encryption commutative :

Nous avons ensuite étendu le résultat précédent sur les protocoles avec exponentiation au cas des protocoles utilisant un opérateur d'encryption commutative. En effet, bien que ces deux opérateurs semblent de prime abord très différents, leur fonctionnement général n'est pas fondamentalement différent, ce qui nous a permis d'adapter les preuves du cas précédent au lieu de réaliser une nouvelle étude complexe. De plus, nous avons montré que le problème de l'insécurité de protocoles est également NP-complet dans le cas des protocoles Ping-Pong avec encryption commutative (et nombre arbitraire de sessions). Totalement disjoint de l'étude précédente, ce résultat montre que même sur des protocoles très simples, l'ajout de propriétés de commutation rend le problème NP-complet. (alors que sans commutation, la vérification des protocoles ping-pong est polynomiale).

Combinaison de modèles à sessions infinies et à messages non bornées :

Nous avons étudié la complexité de la vérification de protocoles cryptographiques lorsque l'on combine deux modèles de protocoles très différents, le premier à nombre de sessions fini mais tailles de messages non bornées, et le second à nombre de sessions non borné mais tailles de messages finies. Ceci permet de construire des attaques utilisant les propriétés des deux modèles. Pour combiner ces deux modèles, nous avons étendu les capacités de l'intrus dans le modèle par rôles (sessions bornées) de manière à modéliser un nombre infini de sessions parallèles à messages bornés. Ceci nous a conduit à un algorithme DEXPTIME de décision de l'insécurité de protocoles cryptographique dans cette combinaison de modèles.

Réalisation logicielle :

Sur le plan concret, nous avons présenté un outil efficace de vérification de protocoles cryptographiques, Atsé, développé dans le cadre du projet européen AVISS. Cet outil n'est pas destiné à rester un prototype. En effet, il a pour but de compléter des outils comme DaTac, et supportera bientôt les opérateurs algébriques xor et exponentiation.

Perspectives :

Ce travail peut être étendu et complété de nombreuses manières :

- Tout d'abord, dans ce travail nous n'avons considéré qu'une seule propriété de sécurité : le secret d'une donnée critique. En effet, il est assez raisonnable de penser que d'autres propriétés de sécurité comme l'authentification peuvent être résolues par les mêmes techniques, modulo des transformations du protocole. De fait, des outils concrets comme Atsé, et d'une manière plus générale les outils existants, sont capables de vérifier des protocoles pour les propriétés de secret et d'authentification présentées dans l'introduction. Cependant, il existe de nombreuses autres propriétés de sécurité intéressantes, comme la disponibilité, la non répudiation, ou l'anonymat, qu'il serait intéressant de vérifier. Ces propriétés nécessitent d'adapter le modèle de protocole ne manière non triviale.
- Une autre extension intéressante du modèle de protocoles cryptographiques par rôles consiste à ajouter des estampillages. Il s'agit d'un marquage de certaines données (en général des nonces) permettant d'en spécifier la durée de vie, comme par exemple pendant combien de temps après sa création un nonce donné sera accepté par un serveur. L'utilisation des estampillages nécessite donc d'ajouter une notion de temps au modèle

de protocoles par rôles, et de spécifier le temps nécessaire à chaque action, que ce soit le traitement d'une donnée par un principal ou un temps de transmission d'une donnée sur le réseau. Selon les contraintes sur les estampillages que l'on veut modéliser, ceci peut conduire à des problèmes de planification de tâches assez compliqués.

- Nous avons présenté une combinaison entre deux modèles pourtant d'aspects très distincts, les modèles à messages bornés et à sessions bornées. Dans la même idée, il pourrait être intéressant de combiner le modèle par rôles avec d'autres modèles décidables de protocoles à nombre de sessions infinies, utilisant d'autres restrictions que la taille des messages. Ceci pourrait permettre, par exemple, d'identifier certains pas du protocole à vérifier, ou certains rôles, dont la structure permet une vérification avec un nombre non borné d'itération. Les autres pas de protocoles, ou rôles, ne possédant pas cette structure particulière seraient quant à eux modélisés avec un nombre borné à priori de sessions.
- La réduction de l'hypothèse de chiffrement parfait est un défi important à l'heure actuelle, et nécessite une description très précise des opérateurs cryptographiques utilisés. Pour cela, les méthodes cryptographiques sont nécessaires mais très difficiles à intégrer dans les méthodes formelles. M. Backes, B. Pfitzmann, M. Waidner ont présenté une librairie de primitives cryptographiques sûrs permettant de faire ce lien [10]. Il serait très intéressant de lier cette librairie à des procédures de décision comme celles présentées ici. Cela faciliterait les preuves de sécurité du point de vue concret.

Index

Bibliographie

[1] M. Abadi and M. Fiore. Computing symbolic models for verifying cryptographic protocols. In Proc. *14th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, June 2001.

[2] M. Abadi and A. Gordon. A calculus for cryptographic protocols : the spi calculus. In *Journal of Information and Computation*, volume 148, 1999.

[3] R.M. Amadio and W. Charatonik. On Name Generation and Set-Based Analysis in the Dolev-Yao Model. In *CONCUR 2002*, LNCS 2421, pages 499–514. Springer-Verlag, 2002.

[4] R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols, In *Proc. CONCUR 2000*, Springer Lecture Notes in Computer Science 2000.

[5] R.M. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Technical Report RR-4147*, INRIA, 2001.

[6] R.M. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1) :695–740, 2002.

[7] A. Armando, D. Basin, M. Bouallagui, Y. Chevalier, L. Compagna, S. Mödersheim, M. Rusinowitch, M. Turuani, L. Viganò, and L. Vigneron. The AVISS Security Protocol Analysis Tool. In *CAV 2002*, LNCS 2404, pages 349–353. Springer, 2002.

[8] D. Basin. Lazy infinite-state analysis of security protocols. In *Secure Networking — CQRE [Secure] '99*, LNCS 1740, pages 30–42. Springer-Verlag, Berlin, 1999.

[9] M. Backes and B. Pfitzmann. A Cryptographically Sound Security Proof of the Needham-Schroeder-Lowe Public-Key Protocol. Cryptology ePrint Archive, Report 2003/121, http ://eprint.iacr.org/, June 2003.

[10] M. Backes, B. Pfitzmann and M. Waidner. Symmetric Authentication Within a Simulatable Cryptographic Library. accepted for ESORICS 2003 (8th European Symposium on Research in Computer Security), Springer-Verlag.

[11] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In proceedings of *Computer Security Foundations Workshop*, 2001.

[12] B. Blanchet. From secrecy to authenticity in Security Protocols. In proceedings of *SAS*, pages 342-359, 2002.

[13] A. Bockmayr and V. Weispfenning. Solving numerical constraints. In *Handbook of Automated Reasoning*, volume I, chapter 12, pages 751–842. Elsevier Science, 2001.

[14] D. Bolignano. Towards the formal verification of electronic commerce protocols. In *IEEE Computer Security Foundations Workshop*, pages 133–146. IEEE Computer Society, 1997.

[15] R. Book and F. Otto. String-rewriting Systems. Springer, New York, 1996.

[16] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proc. of ICALP 2001)*, LNCS 2076, pages 667–681. Springer-Verlag, 2001.

[17] M. Boreale and M.G. Buscemi. On the Symbolic Analysis of Low-Level Cryptographic Primitives : Modular Exponentiation and the Diffie-Hellman Protocol. In *In Proceedings of the Workshop on Foundations of Computer Security (FCS 2003)*, 2003.

[18] I. Borsh and L.B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. In Proc. Amer. Math. Soc. 55, 299-304 (A6), 1976.

[19] M. Bouallagui, Y. Chevalier, M. Rusinowitch, M. Turuani, and L. Vigneron. Analyse Automatique de Protocoles de Sécurité avec CASRUL. Actes de SAR'2002, *Sécurité et Architecture Réseaux*, juillet 2002.

[20] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. A meta-notation for protocol analysis. In P. Syverson, editor *12th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1999.

[21] I. Cervesato and P. Syverson. The Logic of Authentication Protocols. In proceedings of *Foundations of Security Analysis and Design*, pages 63-136, 2001.

[22] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. In Proc. of *LICS 2003*, 2003.

[23] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number of Sessions. Technical Report available at `http://www.inria.fr/rrrt/liste-2003.html`. To appear.

[24] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. Accepted to FSTTCS. Technical Report 0305, CAU Kiel, Germany, 2003. To appear. Available from `http://theory.stanford.edu/~kuesters/publikationen.html`.

[25] Y. Chevalier and L. Vigneron. Towards Efficient Automated Verification of Security Protocols. In *Verification Workshop (VERIFY'01) (in connection with IJCAR'01), Siena, Italy*, June 2001.

[26] Y. Chevalier and L. Vigneron. A Tool for Lazy Verification of Security Protocols. In *16th IEEE International Conference Automated Software Engineering*, November 26-29, 2001 Loews Coronado Bay San Diego, USA.

[27] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In E.Brinksma and K.Guldstrand Larsen, editors, *14th International Conference on Computer Aided Verification*, CAV'02, volume 2404 of *Lectures Notes in Computer Science*, pages 324-337, Copenhagen (Denmark), Jully 2002, Springer.

[28] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature, 1997. Web Draft Version 1.0 available at `http://citeseer.nj.nec.com/`.

[29] E.M. Clarke, and S. Jha, W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In Proceedings of the *IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.

[30] E.M. Clarke, and S. Jha, W. Marrero. Verifying Security Protocols with Brutus. To appear in ACM Transactions in Software Engineering Methodology.

[31] H. Comon-Lundh and V. Cortier. New decidability results for fragments of first-order logic and application to cryptographic protocols. In *Proc. 14th Int. Conf. Rewriting Techniques and Applications (RTA'2003)*, Valencia, Spain, June 2003, volume 2706 of LNCS. To appear.

[32] H. Comon, V. Cortier, and J.C. Mitchell. Tree Automata with one Memory, Set Constraints and Ping-Pong Protocols, ICALP 2001, Crete, Greece, July 8-12, 2001.

[33] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not ? *Journal of Telecommunications and Information Technolog*, 2002.

[34] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of LICS 2003*, 2003.

[35] V. Cortier. Vérification automatique des protocoles cryptographiques. Thèse d'état, Ecole Normale Supérieure de Cachan, 2003.

[36] V. Cortier and J. Millen and Harald Rueß. Proving Secrecy is easy enough. In Proc. *14th IEEE Computer Security Foundations Workshop*, pages 97-108, 2001.

[37] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In *CCC'97*, pages 82–101. IEEE Computer Society, 1997.

[38] G. Delzanno and P. Ganty. Symbolic Methods for Automatically Proving Secrecy and Authentication in Infinite-state Models of Cryptographic Protocols. In proceedings of WISP, *Workshop on Issues in Security and Petri Nets*, Eindhoven, The Netherlands, 2003.

[39] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In Heintze, N. and Wing, J., editors, Proc. of Workshop on Formal Methods and Security Protocols, 25 June 1998, Indianapolis, Indiana,

[40] G. Denker and J. Millen. CAPSL Integrated Protocol Environment. In DARPA Information and Survivability Conference and Exposition (DISCEX'00), Hilton Head, South Carolina, January 25-27, 2000, pp. 207-221, IEEE Computer Society Press

[41] G. Denker, J. Millen, F. Küster Filipe and A. Grau. Optimizing Protocol Rewrite Rules of CIL Specifications. In 13th IEEE Computer Security Foundations Workshop, July 3-5, 2000, Cambridge, England

[42] D. Dolev, S. Even, and R.M. Karp. On the Security of Ping-Pong Protocols. *Information and Control*, 55 :57–68, 1982.

[43] D. Dolev and A.C. Yao. On the Security of Public-Key Protocols. In *IEEE Transactions on Information Theory*, 29(2) :198–208, 1983. Also STAN-CS-81-854, May 1981, Stanford U.

[44] B. Donovan, P. Norris, and G. Lowe, Analyzing a library of protocols using Casper and FDR. In *Proc. of FMSP'99*, `http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/`

[45] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov. Undecidability of Bounded Security Protocols In *Workshop on Formal Methods and Security Protocols,* July 5, 1999, Trento, Italy (part of FLOC'99)

[46] D. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In Proceedings of Advances in Cryptology, CRYPTO'84, pages 10-18, 1985.

[47] S. Even and O. Goldreich. On the Security of Multi-Party Ping-Pong Protocols. In *FOCS'83*, pages 34–39, 1983.

[48] M. R. Garey and D. S. Johnson Computers and Intractability : A guide to the Theory of NP-Completeness. Livre aux éditions W.H. FREEMAN AND COMPANY, 1979.

[49] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In proceedings of *CADE'00*, pages 271-290, 2000.

[50] J. Goubault-Larrecq A Method for Automatic Cryptographic Protocol Verification (Extended Abstract). In Proc. of *Workshop on Formal Methods in Parallel Programming, Theory and Applications (FMPPTA'2000)*, LNCS 1800, pages 977–984, Springer, 2000.

[51] J. Heather and S. Schneider. Towards automatic verification of authentication protocols on an unbounded network *13th IEEE Computer Security Foundations Workshop*, July 2000.

[52] Mark W. Hopkins and Dexter C. Kozen. Parikh's Theorem in Commutative Kleene Algebra In *Logic in Computer Science*, pages 394-401, 1999.

[53] A. Huima. Efficient infinite-state analysis of security protocols. LICS'99, Workshop on Formal Methods and Security Protocols, 1999.

[54] F. Jacquemard, M. Rusinowitch and L. Vigneron. Compiling and Verifying Security Protocols. In *Logic for Programming and Automated Reasoning*, St Gilles, Reunion Island. Springer Verlag, 2000. LNCS. vol 1955. 30 p. M. Parigot and A. Voronkov editors.

[55] D. Kapur, P. Narendran, and L. Wang. Analyzing protocols that use modular exponentiation : Semantic unification techniques. In Proc. of *RTA 2003*, 2003.

[56] R. Küsters. On the Decidability of Cryptographic Protocols with Open-ended Data Structures. In *Proc. of CONCUR 2002*.

[57] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In Book *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS, volume 1055, pages 147–166, 1996.

[58] G. Lowe. Casper : a compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1) :53–84, 1998.

[59] G. Lowe. Towards a completeness result for model checking of security protocols. In Proc. of *11th IEEE Computer Security Foundations Workshop*, pages 96–105. IEEE Computer Society, 1998.

[60] F. Martinelli Languages for the description and the analysis of authentication protocols. In Proc. of *ICTCS'98*. World Scientific Press.

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

[62] C. Meadows. Formal verification of security protocols : A survey. In Proceedings of ASIACRYPT'94, pages 133-150, 1995

[63] C. Meadows. The NRL protocol analyzer : an overview. In *Journal of Logic Programming*, 26(2) :113–131, 1996.

[64] C. Meadows. Invariant Generation Techniques in Cryptographic Protocol Ananysis. In Proceedings of *Computer Security Foundations Workshop*, pages 159-167, 2000.

[65] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In Proc. of *DISCEX 2000*, pages 237–250. IEEE Computer Society Press, 2000.

[66] C. Meadows and P. Narendran. A Unification Algorithm for the Group Diffie-Hellman Protocol. In *Workshop on Issues in the Theory of Security (WITS 2002)*, 2002.

[67] J. Millen. CAPSL : Common Authentication Protocol Specification Language. *Technical Report MP 97B48*, The MITRE Corporation, 1997.

[68] J. Millen and V. Shmatikov. Bounded-Process Cryptographic Protocol Analysis ACM *Conference on Computer and Communication Security*, nov. 2001.

[69] J. Millen and V. Shmatikov. Symbolic Protocol Analysis with Products and Diffie-Hellman Exponentiation. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW 16)*, 2003.

[70] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 166–175. ACM Press, 2001.

[71] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, pages 141–154. IEEE Computer Society, 1997.

[72] J. Mitchell, V. Shmatikov and U. Stern. Finite state analysis of SSL 3.0. In *Seventh USENIX Security Symposium*, pages 201-216, 1998.

[73] D. Monniaux. Abstracting Cryptographic Protocols with Tree Automata. In proceddings of *SAS'99*, pages 149-163, 1999.

[74] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. In *CSL*, number 78-4, 1978.

[75] L.C. Paulson. Mechanized Proofs for a Recursive Authentication Protocol. In 10th *Computer Security Foundations Workshop*, pages 84–95, 1997.

[76] L. Paulson. The inductive approach to verifying cryptographic protocols. In *Journal of Computer Security*, 6(1) :85–128, 1998.

[77] O. Pereira and J.-J. Quisquater. On the perfect encryption assumption. In *Workshop on Issues in the Theory of Security (in conjonction with ICALP'00)*, pages 42-45, 2000.

[78] O. Pereira and J.-J. Quisquater. A Security Analysis of the Cliques Protocols Suites. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 73–81, 2001.

[79] R. Ramanujam and S.P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. FSTTCS'2003.

[80] R.L. Rivest, A. Shamir and L.M. Adleman. A method for Obtaining Digital Signatures and Public-Key Cryptosystems. In *Communications of the ACM*, vol. 21, n°2, page 120-126, 1979.

[81] A. W. Roscoe. Modeling and verifying key exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Society, 1995.

[82] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 174–190, 2001.

[83] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In Journal Theoretical Computer Science, volume 299, pages 451-475, 2003.

[84] Peter Ryan and Steve Schneider. An attack on a recursive authentication protocol : A cautionary tale. *Information Processing Letters 65* (1998), 7-10.

[85] Peter Ryan and Steve Schneider. Modeling and Analysis of Security Protocols. *Livre aux éditions Addison-Wesley*, 2001.

[86] A. Sabefeld and A.C. Myers. Language-based Information-flow Security. In *Journal on Selected Areas in Communications*, 21(1), 2003.

[87] S. Schneider. Verifying Authentication Protocols with CSP. *10th IEEE Computer Security Foundations Workshop*, 1997.

[88] B. Schneier. Cryptographie Appliquée. 2ème édition. Livre aux éditions Vuibert Informatique, 1997.

[89] C.P. Schnorr. Efficient Signature Generation for Smart Cards. In proceedings of *Advances in Cryptology*, CRYPTO'89, pages 239-252, 1990.

[90] D.X. Song. Athena. A new Efficient Automatic Checker for Security Protocol Analysis. In Proceedings of *Conputer Security Foundations Workshop*, pages 192-202, 1999.

[91] Scott D. Stoller. A Bound on Attacks on Payment Protocols. In Proc. *16th Annual IEEE Symposium on Logic in Computer Science* (LICS), pages 61-70, jun 2001.

[92] V. Shmatikov and D.J.D Hughes. Defining Anonymity and Privacy. In Proceedings of the *Workshop on Issues in the Theory of Security*, WITS'02, 2002.