Vers la vérification automatique des services Web

Martin Stéphane

Laboratoire d'Informatique Fondamentale

Encadrant

Denis Lugiez

Résumé

Les services Web sont des programmes qui permettent de réaliser des transactions via internet comme réserver des titres de transport, commander des produits, ou faire des consultations en ligne,... La sécurisation de tels services est essentielle pour assurer leur bon usage et donc leur utilisation. Ce problème de sécurité recouvre des notions très diverses depuis la non-divulgation de données secrètes (numéro de carte bleue par exemple), l'authentification en passant par la qualité de service... La difficulté du problème vient d'une part de la modélisation qui est loin d'être facile et ensuite par l'élaboration de méthodes de vérification automatique. Ce domaine fait l'objet de nombreuses recherches tant au niveau académique que privé (Microsoft, SAP,...).

Dans ce mémoire nous nous intéressons à la préservation du secret pour un modèle de services basés sur le modèle SOAP. Dans ce modèle, les services sont des protocoles qui font intervenir des agents qui communiquent par réception/envoi de messages au format SOAP. Ce modèle est proche de ce qui existe pour les protocoles cryptographiques. Mais la modification essentielle est que les messages sont des données semi-structurées respectant la norme SOAP : un message est une enveloppe contenant un body (obligatoire) et un header (facultatif) qui contiendra les aspects de sécurisation du message. Nous avons baptisé *protocoles XML* les services modélisés ainsi. Un protocole met en oeuvre une politique de sécurité pour s'assurer que le message est conforme. Dans ce rapport, nous avons utilisé le cadre de la WS-SecurityPolicy normalisée par le W3C.

Contribution : Après un état de l'art sur les services et les normes, nous avons étudié trois points :

- Ajout de contraintes de langages réguliers dans la vérification des protocoles cryptographiques (les messages sont des termes et pas des données semi-structurées). Les langages réguliers sont une première approche pour l'étude de politique de sécurité.
- Localité du système de déduction close qui exprime le pouvoir de l'attaquant quand les objets sont des données semi-structurées et non des termes.
- Elaboration et preuve d'un système de déduction permettant de résoudre le problème de la sécurité des protocoles XML.

Chaque résultat est une extension de ce qui a été fait dans le cas classique et a demandé une extension des techniques utilisées dans ce cas.

Mots-clés : Services Web, XML, Politique de sécurité, Vérification de protocoles cryptographiques.

Table des matières

1 Introduction

Internet est un formidable réseau ouvert à tous. Ce qui au départ était simplement pour la consultation de documents, offre maintenant un éventail de services. Par exemple, Monsieur Lussac voudrait utiliser internet pour aller sur le site de la SNCF ouvert 24h/24, 7j/7, afin d'obtenir un billet qu'il pourra imprimer chez lui. Il utilisera le code de sa carte bleue pour payer. Monsieur Lussac utilise sans le savoir des services web.

Qu'est ce qu'un service web ? Un service Web est à la base une technologie de calculs distribués, basée sur l'envoi et la réception de messages asynchrones au format XML. Ces messages répondent à la norme SOAP (Simple Object Access Protocol). Le principe de base est de pouvoir invoquer des objets à distance, comme Java RMI. L'intérêt de cette technologie c'est qu'elle utilise des protocoles bien connus et utilisés partout (comme le HTTP et le SMTP) et elle est entièrement libre. De plus SOAP a été normalisé par le W3C [9].

L'insécurité règne sur internet. Internet est un réseau qui n'a pas été créé pour être sûr. Une personne mal intentionnée peut lire, intercepter, changer les messages circulant sur la toile et cette personne peut changer d'identité et même se faire passer pour un autre. Donc pour protéger les données sensibles et le bon fonctionnement du service, nous devons sécuriser les services web.

Afin d'y arriver, le plus simple serait d'utiliser une couche application sécurisée comme le https. Mais on aimerait bien devoir sécuriser qu'une seule partie des messages pour plusieurs et diverses raisons : par exemple limiter la charge de calcul d'un serveur, ne vouloir que seule une partie du message soit lisible par d'autres personnes, ... Par exemple M. Lussac va chiffrer son numéro de carte bleue et le mettre dans un message destiné au serveur de la SNCF et ce dernier va envoyer la partie chiffrée à la banque qui seule connaît la clef de déchiffrement.

La solution est de mettre en place des politiques de sécurité. Les politiques de sécurité sont des spécifications sur les messages envoyés ou reçus d'un service web. On peut le voir comme si on mettait le service web derrière un pare-feu qui contrôle et sécurise tout les messages entrant et sortant du service. Par exemple : vérifie la forme d'un message, chiffre ou signe les parties du message qui doivent l'être, ... Le langage pour décrire ces politiques est normalisé par le W3C [10].

Mais l'erreur est humaine et peut coûter très cher. Par exemple les failles du protocole WEP (protocole anciennement utilisé pour sécuriser le Wifi), font encore des victimes en entreprise [1]. Il faudrait donc pouvoir vérifier automatiquement et de manière formelle un service web.

Ce mémoire est consacré à la vérification automatique de service web. Les résultats obtenus sont :

- Ajout de contraintes de régularité (section 3)
- Résultat de localité du système de déduction dans le cadre d'échange de données XML (section 4.2)
- Elaboration et preuve d'un système de déductions permettant de résoudre le problème de la sécurité des protocoles XML (section 4.4).

2 Vérification de protocoles cryptographiques

2.1 Objectif

Le but, ici, est, à partir d'un profil d'attaquant et une propriété, de vérifier si le protocole satisfait bien les propriétés attendues. Ces propriétés sont les suivantes :

Le secret : un secret est une donnée (i.e. une suite de bits), formellement un terme, qu'aucune personne non autorisée ne doit connaître. Démontrer que cette propriété n'est pas satisfaite revient à trouver une attaque telle que le secret puisse être connu par une personne non autorisée. Par exemple un numéro de carte bleue ne doit être connu que par M. Lussac et la banque.

L'authentification : l'authentification est la garantie que la personne avec laquelle on échange des messages soit la bonne. Par exemple M. Lussac aimerait bien être sûr de commander ses billets de train à la SNCF et pas à une autre personne se faisant passer pour la SNCF, qui ne rendrait pas service en contre partie du paiement.

L'anonymat : l'anonymat est l'impossibilité, d'une autre personne que lui même, de connaître l'auteur d'une action ou message. Par exemple M. Lussac n'aimerait pas que d'autres sachent que c'est lui qui a écrit ces choses désagréables dans le sondage de la SNCF.

L'équité : l'équité est le fait qu'aucun des participants à un système ne soit désavantagé vis à vis des autres. Par exemple, lorsque M. Lussac imprime ces billets dans son cyber-café favori, il ne voudrait pas que sa demande d'impression soit systématiquement traitée après toute autre impression, y compris celles lancées après.

Dans ce mémoire nous considérons uniquement la propriété du secret.

Il y a deux façons complémentaires de procéder : soit on cherche les attaques, soit on essaye de prouver qu'un protocole est sûr. Malheureusement nous n'avons pas de méthodes automatiques pour démontrer la sûreté d'un protocole sans chercher les attaques, et la preuve à la main peut se montrer fastidieuse.

2.2 Modélisation

Tout d'abord, nous modélisons les messages comme des termes sur un alphabet $\mathcal{F}$. Les services web utilisent des constantes, des noms d'utilisateur, des nonces (souvent générés aléatoirement), ..., et des messages composés par paire $< u, v >$ ou chiffrés avec une clef $\{u\}_v$. Donc $\mathcal{F}$ contient les constantes et les symboles $\{\_\}\_$ (chiffrement) et $< \_, \_ >$ (paire). Soit $\mathcal{X}$ est un ensemble de variables.

2.2.1 Protocoles

Nous définissons les protocoles comme une suite, supposée dénombrable, d'envois et de réceptions de messages entre plusieurs participants (Exemple Fig 1).

Dans le modèle classique de Dolev Yao [8], il y a les personnes honnêtes qui doivent envoyer les messages définis par le protocole dans le bon ordre, et tous

les messages envoyés passent par l'attaquant. Celui-ci peut modifier le contenu ou forger de nouveaux messages à partir de ce qu'il a interprété. Une session est une exécution du protocole.

Par exemple voici le protocole de Denning Sacco :

Tous les $N\_$ sont ici des nombres générés aléatoirement appelés des nonces. Les K_{XY} sont des clefs symétriques connus uniquement de X et de Y

1. $A \rightarrow S :< A, B >$
2. $S \rightarrow A : \{< \{< N_{AB}, < A, N_S >>\}_{K_{BS}}, < B, < N_{AB}, N_S >>>\}_{K_{AS}}$
3. $A \rightarrow B : \{< N_{AB}, < A, N_S >>\}_{K_{BS}}$
4. $B \rightarrow A : \{S_{AB}\}_{N_{AB}}$

FIG. 1 – protocole Denning-Sacco à clef partagée

Ce protocole bien connu est utilisé pour que A puisse s'authentifier à B, afin que B puisse lui donner un secret S_{AB}. A demande à un serveur de confiance S, une clef commune entre A et B, en lui envoyant la paire $< A, B >$ (étape 1 du protocole). A l'étape 2, S envoie un message chiffré avec K_{AS} à A, contenant la clef générée N_{AB} et un paquet chiffré avec la clef K_{BS} que A va renvoyer à B (étape 3) afin que B sache que ce message provient bien de S. Confiant B donne à A le secret S_{AB} chiffré avec la clef commune (étape 4).

2.2.2 Hypothèse sur l'attaquant

Nous écrirons $T \vdash u$ pour dire qu'à partir des connaissances T l'intrus peut déduire u.

$$\textbf{P}\ \frac{T \vdash u \qquad T \vdash v}{T \vdash < u, v >} \qquad \textbf{UL}\ \frac{T \vdash < u, v >}{T \vdash u} \qquad \textbf{UR}\ \frac{T \vdash < u, v >}{T \vdash v}$$

$$\textbf{C}\ \frac{T \vdash u \qquad T \vdash v}{T \vdash \{u\}_v} \qquad \textbf{D}\ \frac{T \vdash \{u\}_v \qquad T \vdash v}{T \vdash u} \qquad \textbf{A}\ \frac{}{T \vdash u}\ \text{Si } u \in T$$

Les règles **P** et **C** sont appelées règles de composition et les règles **UL**, **UR** et **D** sont appelées règles de composition.

FIG. 2 – Règles de déduction de l'attaquant **(DY)**

Il y a deux types d'attaquant :

- l'attaquant passif peut écouter tous les messages transitant sur le réseau et déduire certains secrets à partir de ces messages. Il utilise les règles de déduction de l'intrus **(DY)**. Par exemple, la règle **D** dit qu'à partir d'un message chiffré $\{u\}_v$ et de la clef v, on peut obtenir u.
- l'attaquant actif peut non seulement écouter, mais aussi bloquer les messages, forger de nouveaux messages et les envoyer. Plusieurs sessions du protocole peuvent être exécutées en parallèle, ce qui permet à l'attaquant d'utiliser les informations déduites d'une session dans une autre. Ceci augmente considérablement les difficultés pour vérifier les protocoles.

Dans ce mémoire, nous allons prendre pour hypothèse que l'attaquant est actif. Mais combien de sessions peut jouer ce malotru ?

2.2.3 Hypothèse sur le nombre de sessions

La modélisation des protocoles cryptographiques voudrait que le nombre de sessions jouées par les participants soit non borné. Malheureusement, le problème de sécurité est indécidable [12]. Donc nous nous limiterons à vérifier un nombre borné de sessions.

2.2.4 Hypothèse sur le chiffrement

Le chiffrement peut être aussi une faille dans les protocoles cryptographiques. Il est possible d'attaquer directement le chiffrement : comme le déchiffrement *d'Enigma* pendant la seconde guerre mondiale [11].

Chaque algorithme a des propriétés algébriques, qui peuvent être utilisées pour créer une attaque. Par exemple, l'attaque de Wiener sur RSA [6][p. 198] due à une mauvaise utilisation de l'algorithme. Ou le protocole WEP qui a été cassé à cause d'un manque de protections des vecteurs d'initialisation et d'une mauvaise utilisation du "ou exclusif" [5].

Pour simplifier le problème nous considérerons que le chiffrement est sûr. C'est-à-dire que la seule méthode pour déchiffrer un message est de posséder la clef de déchiffrement. De plus nous considérons aussi que les nombres aléatoires sont correctement générés, ce qui veut dire qu'aucune personne extérieure ne peut deviner le nombre que l'on a généré.

2.3 La vérification pour un nombre borné de sessions

Nous suivons la présentation de Comon-Lundh [3] qui donne un algorithme de résolution du problème du secret avec un intrus possédant les règles de déduction de la figure 2.

2.3.1 La propriété de localité

Nous avons un lemme primordial dans la vérification de protocole cryptographique, le lemme de localité :

Lemme 1 *Si $T \vdash u$ est prouvable à l'aide des règles* **(DY)**, *alors il existe une preuve dont tous les noeuds sont de la forme $T \vdash v$ avec $v \in St(T, u)$. Si la dernière règle appliquée est une décomposition, on a même $v \in St(T)$ pour tous les noeuds $T \vdash v$*

Ce lemme établit que tout ce qui est nécessaire pour obtenir un message à l'aide des règles **(DY)**, est "écrit" dans la connaissance initiale de T_0 et le message m ; puisque seuls des sous-termes sont nécessaires dans la preuve $T_0 \vdash m$. Cela donne immédiatement une procédure en temps polynomial non déterministe pour décider si $T \vdash m$

Proposition 2 *$T \vdash m$ est décidable en temps polynomial non déterministe.*

Cette proposition implique que la vérification de protocole cryptographique avec un attaquant passif est dans NP.

2.3.2 Système de contraintes et sécurité des protocoles

Vérifier la propriété du secret pour un protocole se ramène à résoudre un système de contraintes C qui est une suite d'expressions $T_1 \Vdash u_1, ..., T_n \Vdash u_n$. La résolution d'un système est décrite par des règles de transformation définissant une relation $\rightsquigarrow$. La relation $\rightsquigarrow$ est indicée par un substitution σ dont le domaine est contenu dans l'ensemble des variables libres de la contrainte. $(C, \theta) \rightsquigarrow (C', \theta\sigma)$ si $C \rightsquigarrow_\sigma C'$, la deuxième composante de la paire n'est là que pour garder en mémoire la séquence de substitutions appliquée à la contrainte. A l'origine, θ est l'identité. Lorsque la substitution σ n'est pas précisée, c'est la substitution identité.

Une substitution close σ est solution d'une contrainte C si et seulement si, pour tout $T \Vdash u \in C, T\sigma \vdash u\sigma$ est dérivable par les règles de déduction de l'intrus.

L'algorithme est le suivant : on fixe le nombre n d'instance de chaque rôle du protocole et on devine un entrelacement d'actions de chaque rôle en respectant la séquence du protocole. $u_1 \Rightarrow v_1, ..., u_n \Rightarrow v_n$. Il existe une attaque sur le protocole si et seulement si le système de contraintes suivant admet une solution :

$$\begin{array}{rcl} T_0 & \Vdash & u_1 \\ T_0, v_1 & \Vdash & u_2 \\ & \vdots & \\ T_0, v_1, ..., v_{n-1} & \Vdash & u_n \\ T_0, v_1, ..., v_n & \Vdash & s \end{array}$$

Avec T_0 comme l'ensemble des connaissances initiales de l'intrus (nom des participants, clefs publiques, les messages initiaux, ...) et de plus $var(T_0, v_1, ..., v_i) \subseteq var(u_j), j < i$

Nous résolvons ce système avec les règles de simplification de contraintes de la figure 3.

$$\begin{array}{lrll} R_1 : & C, T \Vdash u & \rightsquigarrow \top & \text{Si } T \cup \{x | T' \Vdash x \in C, T' \subsetneq T\} \vdash u \\ R_2 : & C, T \Vdash u & \rightsquigarrow_\sigma C\sigma, T\sigma \Vdash u\sigma & \text{Si } \sigma = mgu(t, u), t \in St(T), t \neq u; t, u \notin \mathcal{X} \\ R_3 : & C, T \Vdash u & \rightsquigarrow_\sigma C\sigma, T\sigma \Vdash u\sigma & \text{Si } \sigma = mgu(t_1, t_2), t_1, t_2 \in St(T), t_1 \neq t_2; t_1, t_2 \notin \mathcal{X} \\ R_4 : & C, T \Vdash \{u\}_v & \rightsquigarrow C, T \Vdash u \wedge C, T \Vdash v & \\ R_5 : & C, T \Vdash < u, v > & \rightsquigarrow C, T \Vdash u \wedge C, T \Vdash v & \\ R_6 : & C, T \Vdash u & \rightsquigarrow \bot & \text{Si } T = \emptyset \text{ ou } [Var(T, u) = \emptyset \text{ et } T \nvdash u] \end{array}$$

FIG. 3 – Règles de simplification de contraintes

2.3.3 Exemple

Voici l'exemple de résolution du protocole Donning-Sacco décrit page 5 figure 1. Nous avons 3 rôles : le serveur de clef s, le client Alice a et le client Bob b. On obtient donc :

$$\begin{array}{lrcl} s. & < u, v > & \Rightarrow & \{<< \{< N_{uv}, < u, N_s >>\}_{K_{vs}}, v >, < N_{uv}, N_s >>\}_{K_{us}} \\ a. & \{<< z, b >, < x, y >>\}_{K_{as}} & \Rightarrow & z \\ b. & \{< x', < a, y' >>\}_{K_{bs}} & \Rightarrow & \{S_{ab}\}_{x'} \end{array}$$

Nous avons choisi l'entrelacement suivant : (s,a,b). Ce qui donne le système suivant :

$$\begin{array}{rcl} a, b, s = T_0 & \Vdash & < u, v > \\ T_0, (\{<< \{< N_{uv}, < u, N_s >>\}_{K_{vs}}, v >, < N_{uv}, N_s >>\}_{K_{us}}) = T_1 & \Vdash & \{<< z, b >, < x, y >>\}_{K_{as}} \\ T_0, T_1, z = T_2 & \Vdash & \{< x', < a, y' >>\}_{K_{bs}} \\ T_0, T_1, T_2, \{S_{ab}\}_{x'} = T_3 & \Vdash & S_{ab} \end{array}$$

dont $\{u, v, x, x', y, y', z\} \in \mathcal{X}$ et le reste ce sont des constantes.

On le résout à l'aide des règles de simplifications figure 3 de la page 7.

3 Les politiques de sécurité

La norme du W3C définit un langage basé sur XML pour décrire les politiques de sécurité. Ce langage exprime des contraintes sur la forme du message, plus précisément les parties qui sont chiffrées, signées ou requises et les algorithmes utilisés. Par exemple, la balise *encryptedElement* permettant de chiffrer un élément du message. Cet élément est sélectionné avec XPATH qui est un langage de requête pour les documents XML.

Les politiques de sécurité peuvent devenir facilement très complexes. Devant le grand nombre d'assertion que propose la norme WS-SecurityPolicy, nous avons décidé pour commencer de modéliser les politiques de sécurité par des langages réguliers d'arbres [7]. Ce modèle est peu expressif vis à vis de la norme du W3C, mais permet de comprendre l'interaction entre la politique de sécurité et le comportement du protocole. Dans ce cadre un message respecte la politique de sécurité s'il appartient à un langage régulier d'arbre donné.

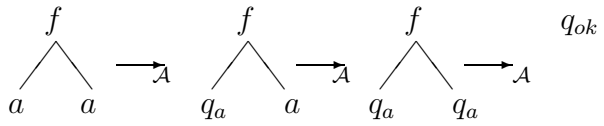
3.1 Qu'est ce qu'un automate d'arbre ?

Un automate d'arbre non déterministe dans $\mathcal{F}$ est un quadruplet $A = (Q, \mathcal{F}, Q_f, \Delta)$ Q est un ensemble d'états, $Q_f \subseteq Q$ est un ensemble d'états finaux, et Δ est un ensemble de règles de transition de type : $f(q_1, ..., q_n) \to q$ avec $q, q_i \in Q$ et $f \in \mathcal{F}$. Le principe est de partir des feuilles et d'arriver à la racine avec un état final.

Par exemple : Soit $\mathcal{F} = \{f(\_,\_), a, b\}$ et l'automate $\mathcal{A} = \{Q, \mathcal{F}, Q_f, \Delta\}$, définis par : $Q = \{q_a, q_b, q_{ok}\}, Q_f = \{q_a, q_b, q_{ok}\}$ et Δ est l'ensemble des transitions suivantes :

$$a \to q_a, b \to q_b, f(q_{ok}, q_{ok}) \to q_{ok}, f(q_a, q_a) \to q_{ok}, f(q_b, q_b) \to q_{ok}$$

Soit $\longrightarrow_{\mathcal{A}}$ représente une des transitions de $\mathcal{A}$



Reconnaît les arbres binaires complets où tous les fils d'une même racine ont la même étiquette.

Pourquoi ce choix ? Les termes sont des arbres et les langages réguliers sont classiquement utilisés comme base de système de typage.

3.2 Sémantique

Soit $\mathbb{T}$ l'ensemble des contraintes de typages et τ un type, en l'occurrence un langage d'arbre régulier. Un terme t est de type τ si et seulement si $t \in \tau$, noté $t : \tau$. Une *contrainte de typage* $\mathbb{T}$ est un ensemble d'expressions de la forme $t : \tau$ avec τ un langage régulier et t un terme. Une substitution close σ est solution d'une contrainte de typage si et seulement si $\forall t : \tau \in \mathbb{T}$ on a $t\sigma \in \tau$. L'ensemble des solutions de $\mathbb{T}$ est noté $Sol(\mathbb{T})$.

3.3 Résolution des Types

Nous donnons des règles de typage permettant de déterminer les solutions d'une contrainte de typage. $\mathbb{T} = \bigwedge_{i=1}^{n} t_i : \tau_i$ est en forme résolue si et seulement si $t_i \in \mathcal{X}$ ou $\mathbb{T} = \bot$ et les variables n'apparaissent qu'une seule fois.

τ étant un langage régulier d'arbre, on peut lui associer l'automate $A_\tau = (Q, \Sigma, Q_f, \Delta)$ et $L_\tau(q)$ le langage associé avec pour état d'acceptation q.
Les règles de typages sont :

R'_1 $\quad f(t_1, ..., t_n) : L_\tau(q) \not\hookrightarrow \bigwedge_{i=1}^{n} t_i : L_\tau(q_i)$ Si $\exists f(q_1, ..., q_n) \to q \in A_\tau \wedge q \in Q_f$

R'_2 $\quad t : L_{\tau_1}(q_1) \wedge t : L_{\tau_2}(q_2) \not\hookrightarrow t : L_{\tau_1 \cap \tau_2}(q')$, $q' = q_1 \times q_2$ de l'automate $A_{\tau_1} \times A_{\tau_2}$

R'_3 $\quad \Phi \wedge t : L_\tau(q) \not\hookrightarrow \Phi$ Si $Var(t) = \emptyset$ et $t \in L_\tau(q)$

R'_4 $\quad t : L_\tau(q) \not\hookrightarrow \bot$ Si $Var(t) = \emptyset$ et $t \notin L_\tau(q)$

R'_5 $\quad t : \emptyset \not\hookrightarrow \bot$

R'_6 $\quad \Phi \wedge \bot \not\hookrightarrow \bot$

R'_7 $\quad f(t_1, ..., t_n) : L_\tau(q) \not\hookrightarrow \bot$ Si $\nexists f(q_1, ..., q_n) \to q \in A_\tau \wedge q \in Q_f$

Nous considérons les fonctions de cryptographie et de couple comme des fonctions d'arité deux.

Nous remarquons que la première règle n'est pas déterministe. c'est-à-dire qu'elle peut donner plusieurs résultats différents selon la règle de l'automate choisie, s'il y en a plusieurs correspondantes. Ainsi l'ensemble des solutions est l'union des solutions de tous les chemins possibles.

Proposition 3 *La résolution de typage termine sur un système en forme résolue*

Preuve : Pour la terminaison, nous allons prendre la mesure $C(\mathbb{T}) = \Sigma|t_i|$, $|t_i|$ étant la taille de l'arbre t_i . Les règles R'_2 à R'_7 font décroître strictement $C(\mathbb{T})$ car elles éliminent toutes un élément de la conjonction. R'_1 fait aussi décroître strictement $C(\mathbb{T})$ car elle élimine une fonction et met les feuilles dans $\mathbb{T}$. De plus le test d'appartenance à un langage régulier termine.

Enfin l'algorithme de simplification de type s'arrête avec $\mathbb{T} = \bot, \mathbb{T} = \emptyset$ ou aucune règle n'est applicable. Si aucune règle n'est applicable cela implique que tout $t : \tau$ de $\mathbb{T}$ n'est pas une fonction, sinon nous pourrions appliquer R'_1 ou R'_7. Il n'y a pas de terme non variable sinon nous pourrions appliquer R'_3 ou

R'_4, et ces variables n'apparaissent qu'une seule fois, sinon nous pourrions appliquer R'_2. □

Proposition 4 *Si* $\mathbb{T} \rightarrowtail \mathbb{T}' \Rightarrow Sol(\mathbb{T}') \subseteq Sol(\mathbb{T})$.

Preuve : La preuve se fait sur les règles :
R'_1, R'_7 : l'algorithme d'appartenance à un langage d'arbre régulier nous donne la preuve.
R'_2 : Si $t : L_{\tau_1}(q_1)$ et $t : L_{\tau_2}(q_2)$ alors par définition $t : L_{\tau_1}(q_1) \cap L_{\tau_2}(q_2)$.
R'_3, R'_4 et R'_5 par définitions. □

Proposition 5 *Soit* $\sigma \in Sol(\mathbb{T})$ *alors* $\exists \mathbb{T}' | \mathbb{T} \rightarrowtail \mathbb{T}'$ *et* $\sigma \in Sol(\mathbb{T}')$.

Preuve :
R'_1 : $f(t_1, ..., t_n)\sigma \in L_\tau(q)$ et $\exists f(q_1, ..., q_n) \rightarrow q \in A_\tau \wedge q \in Q_f \Rightarrow \sigma t_i \in L_\tau(q_i)$.
R'_2 : Par définition si $t\sigma \in \tau_1$ et $t\sigma \in \tau_2$ alors $t\sigma \in \tau_1 \cup \tau_2$
R'_3 : $var(t) = \emptyset$ donc $t\sigma = t$ et donc le lemme est prouvé.
R'_4 **à** R'_7 : Contredit l'hypothèse que σ est une solution de $\mathbb{T}$.

□

3.4 Règles de simplification

Nous avons ajouté aux règles de Comon-Lundh des contraintes de typage, qui seront vérifiées à la fin (avec les règles R_1 et R_6). Les règles de simplification des contraintes du système sont :

R_1 :	$\mathbb{T} \wedge T \Vdash u : \tau \rightsquigarrow \top$	**Si** $T \cup \{x \mid T' \Vdash x \in C, T' \subset T\} \vdash u$ et $\mathbb{T}, u : \tau \not\overset{*}{\rightarrowtail} \bot$
R_2 :	$\mathbb{T} \wedge C \wedge T \Vdash u : \tau \rightsquigarrow_\sigma \mathbb{T}\sigma \wedge C\sigma \wedge T\sigma \Vdash u\sigma : \tau$	**Si** $\sigma = mgu(t, u), t \in St(T)$, $t \neq u; t, u \notin \mathcal{X}$
R_3 :	$\mathbb{T} \wedge C \wedge T \Vdash u : \tau \rightsquigarrow_\sigma \mathbb{T}\sigma \wedge C\sigma \wedge T\sigma \Vdash u\sigma : \tau$	**Si** $\sigma = mgu(t_1, t_2), t_1, t_2 \in St(T)$, $t_1 \neq t_2; t_1, t_2 \notin \mathcal{X}$
R_4 :	$\mathbb{T} \wedge T \Vdash \{u\}_v : \tau \rightsquigarrow (\{u\}_v : \tau \wedge \mathbb{T}) \wedge T \Vdash u \wedge T \Vdash v$	
R_5 :	$\mathbb{T} \wedge T \Vdash < u, v > : \tau \rightsquigarrow (< u, v > : \tau \wedge \mathbb{T}) \wedge T \Vdash u \wedge T \Vdash v$	
R_6 :	$\mathbb{T} \wedge T \Vdash u : \tau \rightsquigarrow \bot$	**Si** $T = \emptyset$ ou $[Var(T, u) = \emptyset$ et $T \not\vdash u]$ ou $\mathbb{T}, u : \tau \overset{*}{\rightarrowtail} \bot$

Les termes non typés sont de type générique, c'est-à-dire l'automate associé à ce type accepte tout. Dans la section suivante nous allons exprimer les règles de typage.

3.5 Terminaison

Proposition 6 *Les Règles de simplification terminent.*

Preuve : Ces règles de simplifications sont similaires à celles de Comon-Lundh [3], suivies de la vérification du typage (qui termine - proposition 3). □

3.6 Correction

Proposition 7 *Si* $(\mathbb{T}, C) \rightsquigarrow_\sigma (\mathbb{T}', C')$*, alors pour toute solution* θ *de* $(\mathbb{T}', C')$*,* $\sigma\theta$ *est une solution de* $(\mathbb{T}, C)$

Preuve :

R_1 : D'après le lemme 3 de [3] : Toute solution de C est aussi solution de $T \Vdash u$. D'après la proposition 4, les règles de simplification sont correctes. On a fini ce cas.

R_2, R_3 : Résulte des définitions et du fait qu'un type n'as pas de variable donc $u\sigma : \tau\sigma = u\sigma : \tau$

R_4, R_5 : Résulte du pouvoir de construction de l'intrus.

□

3.7 Complétude

Proposition 8 *Toute contrainte qui n'est pas en forme résolue est réductible ou bien n'a pas de solution.*

Preuve : Ceci résulte directement des résultats de complétude du cas classique [3, Section 3.3, Lemme 9] et de la complétude des règles de typage : proposition 5. □

Ceci nous donne un des résultats de ce mémoire :

Proposition 9 *La vérification de protocoles cryptographiques sur un nombre borné de sessions avec des contraintes de régularité sur les messages est décidable.*

Ainsi nous pouvons essayer de formaliser des messages XML, et nous pensons nous rapprocher de la décidabilité de la fiabilité d'un Service web.

4 Modélisation XML

4.1 Abstraction XML

Dans cette modélisation nous allons essayer de nous rapprocher le plus possible des documents semi-structurés, XML. Ces documents sont vus comme des arbres d'arité non borné et nous avons choisi le point de vue que les fils sont non ordonnés, car des attaques utilisent le fait que certains interpreteurs ne vérifient pas l'ordre. Par exemple ils vérifient qu'un des fils a été signé et choisissent la première occurrence. Un attaquant peut déplacer le fils signé et ajouter un fils similaire. Les fils d'un noeud seront donc considérés comme éléments d'un multi-ensemble, et nous décrivons dans la suite l'algorithme d'unification sur ces termes.

4.1.1 Termes

Nous avons deux sortes de termes $\mathcal{M}$ (multi-ensemble) et $\mathcal{U}$(Universel). Tous les termes de type $\mathcal{M}$ sont aussi de type $\mathcal{U}$. Soit $\mathcal{F}$ un ensemble de fonctions. Soit $\mathcal{X}$ un ensemble de variables de sorte $\mathcal{M}$ ou $\mathcal{U}$.

L'ensemble des termes $T_{\mathcal{M}}(\mathcal{X})$ est le plus petit ensemble tel que :

- $\{||\} \in T_{\mathcal{M}}(\mathcal{X})$ est de sorte $\mathcal{M}$.
- $x \in \mathcal{X} \Rightarrow x \in T_{\mathcal{M}}(\mathcal{X})$.
- $\forall f, \forall t_1, ..., t_n \in T_{\mathcal{M}}(\mathcal{X}), n \geq 0, f(t_1, ..., t_n) \in T_{\mathcal{M}}$ et est de sorte $\mathcal{U}$.
- $s, t \in T_{\mathcal{M}}(\mathcal{X})$ et s de sorte $\mathcal{M}$ alors $\{|t|s|\} \in T_{\mathcal{M}}(\mathcal{X})$ et est de sorte $\mathcal{M}$.

Notation

- $\{|t_1|\{|t_2|...\{|t_n|\{||\}|\}...|\}$ se note $\{|t_1, ..., t_n|\}$.
- $\{|t_1, ..., \{|t_n|M|\}|\}$ se note $\{|t_1, ..., tn|M|\}$ (M étant de sorte $\mathcal{M}$).

On ne peut pas écrire d'autres termes comme $\{|x|f(a)|\}$.

On note aussi $\sqsubseteq$ l'ordre partiel sur $T_{\mathcal{M}}(\mathcal{X})$, tel que $\forall s, t \in T_{\mathcal{M}}(\mathcal{X}), s \sqsubseteq y$, si t est de sorte $\mathcal{U}$, ou s et t sont de même sorte. Deux termes s et t sont de sortes incompatibles si $s \not\sqsubseteq t$ et $t \not\sqsubseteq s$.

L'égalité sur des termes clos dans $T_{\mathcal{M}}(\mathcal{X})$ L'égalité est définie par induction.

1. $t = t'$ si t et t' sont les mêmes constantes.
2. $f(s_1, ..., s_n) = g(t_1, ..., t_m)$, si $n = m, g = f$ et si $\forall i \in [1..n], s_i = t_i$
3. $\{|s_1, ..., s_n|\} = \{|t_1, ..., t_m|\}$ si n=m et s'il existe un permutation ρ tel que $\forall i \in [1..n], s_{\rho(i)} = t_i$

Par exemple $\{|a, a, b, d, c|\} = \{|a, b, c, d, a|\} \neq \{|a, b, c, d|\}$. Il ne peut y avoir d'autre égalité comme $\{|x|\} = f(x)$.

Définition de sous termes

$$
\begin{aligned}
St(t) &= t, \text{ si } t \text{ est une constante} \\
St(f(t_1, ..., t_n)) &= f(t_1, ..., t_n) \cup St(t_1, ..., t_n) \\
St(\{|t_1, ..., t_{n-1}|t_n|\}) &= \{|t_1, ..., t_{n-1}|t_n|\} \cup St(t_1, ..., t_{n-1}, t_n) \\
St(\{|t_1, ..., t_n|\}) &= \{|t_1, ..., t_n|\} \cup St(t_1, ..., t_n)
\end{aligned}
$$

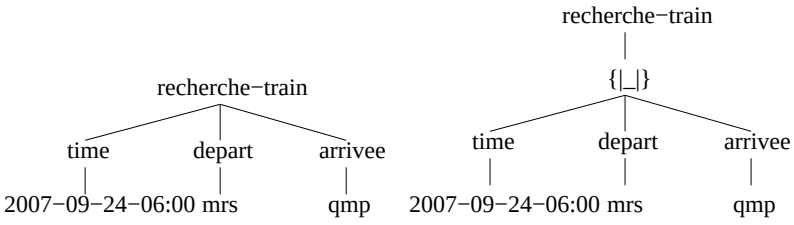
4.1.2 Exemple d'utilisation

Voici un exemple montrant comment transcrire des messages XML dans notre formalisme.

Soit un message au format XML :

```
1      <recherche-train>
2        <time>2007-09-24-06:00</time>
3        <depart>mrs</depart>
4        <arrivee>qmp</arrivee>
5      </recherche-train>
```

Nous allons formaliser ce message comme ceci :



(a) Arbre du message (b) Arbre de la modélisation

```
recherche-train({|time(2007-09-24-06:00),depart(
    mrs),arrivee(qmp)|})
```

4.2 Les règles de déduction de l'intrus et la localité

Nous adaptons les règles de déduction de l'intrus dans ce cadre et un résultat de notre mémoire est de montrer que le résultat de localité reste vrai.

$$\mathbf{P}\ \frac{T\vdash u_1 \quad ... \quad T\vdash u_n}{T\vdash f(u_1,...,u_n)} \qquad \mathbf{PM}\ \frac{T\vdash u_1 \quad ... \quad T\vdash u_n}{T\vdash \{|u_1,...,u_{n-1}|u_n|\}} \qquad \mathbf{C}\ \frac{T\vdash u \quad T\vdash v}{T\vdash \{u\}_v}$$

$$\mathbf{U}\ \frac{T\vdash f(u_1,...,u_n)}{T\vdash u_{i\in[1..n]}} \qquad \mathbf{UM}\ \frac{T\vdash \{|u_1,...,u_{n-1}|u_n|\}}{T\vdash u_{i\in[1..n]}} \qquad \mathbf{D}\ \frac{T\vdash \{u\}_v \quad T\vdash v}{T\vdash u}$$

$$\mathbf{A}\ \frac{}{T\vdash u}\ \text{Si } u\in T$$

Les règles **P**, **PM**, **C** sont appelées règles de composition et les règles **U**, **UM** et **D** sont appelées règles de décomposition.

FIG. 4 – Règles de déduction de l'attaquant dans le cas multi-ensemble

Proposition 10 *Si $T\vdash u$ est prouvable à l'aide des règles fig 4, alors il existe une preuve dont tous les noeuds sont de la forme $T\vdash v$ avec $v\in St(T,u)$. Si la dernière règle appliquée est une décomposition, on a même $v\in St(T)$ pour tous les noeuds $T\vdash v$.*

Preuve : Nous procédons par induction sur la structure de la preuve.

Première étape : L'axiome **A** est trivial.

Deuxième étape : nous allons utiliser la récurrence pour les multi-ensembles.

– Si c'est la composition :

$$\Pi' = \{\frac{\Pi_1=\{\frac{...}{T\vdash t_1} \quad ... \quad \Pi_{n-1}=\{\frac{...}{T\vdash t_{n-1}} \quad \Pi_n=\{\frac{...}{T\vdash t_n}}{\Pi''=\{\frac{T\vdash\{|t_1,...,t_{n-1}|t_n|\}}{...}}$$

Il est clair que $t_1,...,t_n\in St(\{|t_1,...,t_{n-1}|t_n|\})$ le lemme est démontré pour Π''

– Si c'est une décomposition : pour un $i \in [1..n]$ donné, $\dfrac{T \vdash \{|t_1, ..., t_{n-1}|t_n|\}}{T \vdash t_i}$

Nous nous mettons dans le cas de la preuve minimale. Nous n'avons pas pu utiliser une règle de composition à la précédente étape, car il existerait une preuve plus minimale. Donc nous avons une structure de ce type :

$$\Pi = \{\frac{\dots}{T \vdash \{|t_1, ..., t_{n-1}|t_n|\}}$$
$$\Pi' = \{\frac{T \vdash t_i}{\dots}$$
$$\Pi'' = \{$$

Par hypothèse de récurrence $\{|t_1, ..., t_{n-1}|t_n|\} \in St(T)$ car Π' est une décomposition. Nous avons : $\{|t_1, ..., t_{n-1}|t_n|\} \in St(T) \Rightarrow t_1, ..., t_n \in St(T)$. Donc Π'' vérifie aussi le lemme.

Les autres cas sont traités de manière similaire.

□

4.3 Algorithme d'unification

La procédure pour trouver les attaques dans le cas actif utilise l'unification entre deux termes. Nous devons donc étendre l'unification dans notre cadre. Un exemple complet est donné en section 4.3.3 Nous avons choisi un algorithme déjà existant [2] que nous vous résumons ci-dessous. Soit S le système d'équations associé. Nous considérerons que " = " n'est pas un symbole commutatif.

4.3.1 Définitions

Graphe de multi-ensemble associé : Chaque variable de multi-ensemble est représentée par un noeud et toute équation de type $x = \{|Y|u|\}$ donne une arête étiquetée Y, d'origine x et de destination u.

Variables finales de multi-ensemble : une variable de multi-ensemble est dite finale : Dans le graphe de multi-ensemble, il n'y a pas d'arête partant du noeud associé.

Graphe d'équation : Chaque variable de S donne un noeud et chaque équation donne une arête.

Variable finale : une variable est dite finale : si aucune arête part du noeud associé à cette variable. Elles sont notées $V_f(S)$.

Reduction du système par $x = y$: enlevez toutes équations $x = y$ et $y = x$

1. Si x et y sont la même variable ne rien faire.
2. Si x et y sont de sortes incompatibles, S devient $\perp$
3. Si $x \sqsubseteq y$ alors remplacer tous les x par des y et ajouter l'équation $x = y$
4. Si $y \sqsubseteq x$ alors remplacer tous les y par des x et ajouter l'équation $y = x$

une correspondance de multi-ensemble : Soit deux séquences $X = x_1, ..., x_n$ et $Y = y_1, ..., y_m$.

Une correspondance de multi-ensemble est définie comme une relation d'équivalence sur $X \cup Y$ tel que chaque classe d'équivalence vérifie que : le nombre total d'occurrences d'une variable de R dans X est égale au nombre total d'occurrences d'une variable de R dans Y.

Par exemple $X = x, x, z, y$ et $Y = x, y, y, u$. On peut trouver ces classes $\{x, y\}, \{u, z\}$ ou celles là $\{x, u\}, \{y, z\}$ ou celles-ci $\{x, y, z, u\}$. Une correspondance de multi-ensemble est dite minimale si aucune classe d'équivalence ne peut être coupée en deux, tel que les deux sous ensembles en résultant soient eux même une correspondance de multi-ensemble pour nos deux séquences. La première et la seconde correspondance sont minimales.

4.3.2 Algorithme

Etape 1 : L'abstraction

Chaque élément est décomposé de manière suivante :

(a) Enlever toutes les équations $s = t$ où s et t ne sont pas des variables. Créer x nouvelle variable et ajouter les équations $x = s$ et $x = t$.

(b) Enlever toutes les équations $s = t$ où $s \in \mathcal{X}$ et $t \notin \mathcal{X}$, et ajouter l'équation $t = s$.

(c) Enlever toutes les équations de type $s = t$ où $t \in F$ et on crée une variable par paramètre de t et ajouter les équations correspondantes. Exemple : $x = \{|f(a), \{||\}, b|\}$ devient :

$$x = \{|z, y, v|y|\}, z = f(u), y = \{||\}, v = b, u = a$$

x, y, z, u, v sont des nouvelles variables. y est une variable de multi-ensemble.

Etape 2 : Utiliser les règles suivantes autant de fois que possible :

Règle 1 : Si il existe $z = f(x_1, ..., x_n)$ et $z = f(y_1, ..., y_n), f \in F$ et $n \geq 0$, Enlevez $z = f(x_1, ..., x_n)$ et réduire S en utilisant $x_i = y_i, i \in [1..n]$.

Règle 2 : Si on a $z = t$ et $z = s$ ou s et t sont des Multi-ensembles on enlève ces deux équations et Il y a plusieurs cas :

Cas 1 : s ou t n'ont pas d'extension S devient $\perp$

Cas 2 : Si on a la forme $z = \{|X|u|\}$ et $z = \{|Y|u|\}$:

Si : $|X| \neq |Y|, S$ devient $\perp$

Sinon : Ajouter $z = \{|Y|u|\}$ et générer de façon non déterministe la plus petite correspondance de multi-ensemble entre X et Y et réduire S avec.

Cas 3 : Si on a la forme $z = \{|X|u|\}$ et $z = \{|Y|v|\}$, où $u \neq v$, découper de manière non déterministe X et Y en X_1 et X_2, respectivement en Y_1 et $Y_2(X_1, Y_1$ peuvent être vides), tel que $|X_1| = |Y_1|$. Et faire :

(a) Ajouter l'équation $z = \{|X_2, Y_1, Y_2|w|\}, w$ étant une nouvelle variable de multi-ensemble.

(b) Si X_2 est non-vide, ajouter l'équation $u = \{|X_2|w|\}$. Sinon réduire S avec $v = w$

(c) Si Y_2 est non-vide, ajouter l'équation $u = \{|Y_2|w|\}$. Sinon réduire S avec $u = w$

(d) Si X_1 et Y_1 sont non vides, générer de façon non déterministe la plus petite correspondance de multi-ensemble entre X_1 et Y_1 et réduire S avec.

Etape 3 : Verifier qu'il n'y ait pas de cycle dans le graphe de variable.

Etape 4 : Génération d'un unificateur. Pour chaque équation, remplacer " = " par " $\leftarrow$ ", on obtient ainsi $\sigma = \{x_1 \leftarrow t_1, ..., x_n \leftarrow t_n\}$, Remplacer chaque x_i apparaissant dans les t_j par t_i.

Soit $\sigma = U(u, v); u, v \in T_{\mathcal{M}}(\mathcal{X})$ l'unificateur calculé par l'algorithme ci-dessus. Donc σ est de la forme $\{x_1 \leftarrow t_1, ..., x_n \leftarrow t_n\}$ où $x_1, ..., x_n \in \mathcal{X}$ et $t_1, ..., t_n \in T_{\mathcal{M}}(\mathcal{X})$.

Proposition 11 *Soit S un système d'équations et S' une forme résolue de S calculée à l'aide de l'algorithme d'unification.*
- *Si l'algorithme introduit de nouvelles variables alors $|V_f(S)| > |V_f(S')|$*
- *Sinon $|V_f(S)| \geq |V_f(S')|$*

Preuve : La preuve va se faire par induction sur le nombre de règles utilisées. Soit $S_0 = S$, S_1 est S_0 après abstraction et S_{n+1} est S_n après application d'une des règles :

Rang 0 : Pour chaque variable x créée durant l'abstraction une équation $x = t$ est créée donc x n'est pas finale et donc $V_f(S_0) = V_f(S_1)$.

Rang n+1 : Admettons la proposition au rang n, et vérifions la au rang n+1 selon la règle utilisée :

Règle 1 : Supprime que des équations de type $f(y_1, ..., y_n) = f(z_1, ..., z_n)$ et ne crée pas de variable donc $|V_f(S_n)| \geq |V_f(S_{n+1})|$.

Règle 2 :

cas 1 : trivial.

cas 2 : ne crée pas de variable finale et ne supprime pas la dernière équation de type $x_i = t'$ donc $|V_f(S_n)| \geq |V_f(S_{n+1})|$.

cas 3 : on fait ce qui suit :

a) une nouvelle variable finale est créé w.

b) v devient non finale ou disparaît du système.

c) u devient non finale ou disparaît du système.

d) réduit ou ne change pas le nombre de variables finales.

En résumé on crée une variable finale mais on en enlève deux. Donc $|V_f(S_n)| > |V_f(S_{n+1})|$.

De plus si la règle 2 dans le cas 3 est utilisée, on a $\exists i \in [1..n] || V_f(S_0)| \geq ... \geq |V_f(S_i)| > |V_f(S_{i+1})| \geq ... \geq |V_f(S_n)| \Rightarrow |V_f(S_0)| > |V_f(S_n)|$.

□

Proposition 12 *Soit E un ensemble de termes, $u, v \in E$ et $\sigma = U(u, v)$ alors $nbv(E) > nbv(E\sigma)$*

Preuve : Il y a plusieurs cas :

Cas 1 : u ou v est une variable : alors le système est sous forme résolue ou il n'y a pas de solutions (occur check). Donc si σ est généré : $\sigma = [u \leftarrow v]$ ou $\sigma = [v \leftarrow u]$ et donc $var(E\sigma) = var(E)/\{u\}$ ou respectivement $var(E\sigma) = var(E)/\{v\}$ car il n'y a pas de nouvelles variables et $var(u) \cap var(v) = \emptyset$ (occur check). Donc $nbv(E) > nbv(E\sigma)$.

Cas 2 : $V_f(S) = var(u, v)$. Soit S', le système S sous forme résolue.
D'après la proposition 11 nous avons deux cas :

Cas 2.1 : Il n'y a pas de nouvelles variables créées : Comme le système est sous forme résolue nous avons :$(\bigcup_{i=1}^{n} var(x_i)) \cap \bigcup_{i=1}^{n} var(t_i)) = \emptyset$
Et comme $\forall i | var(x_i) \in var(E)$ et comme on ne créé pas de variable finale on a aussi que $var(t_i) \in E$. On a bien $nbv(E) > nbv(E\sigma)$.

Cas 2.2 : D'après la proposition 11 : $|V_f(S)| > |V_f(S')|$, or dans le cas $V_f(S) = var(u, v) \in var(E)$. Et par définition on a $var(u\sigma, v\sigma) \subseteq V_f(S')$. On a $|var(u, v)| = |V_f(S)| > |V_f(S')| \geq |var(u\sigma, v\sigma)|$.
Donc $nbv(E) > nbv(E\sigma)$.

□

4.3.3 Exemple d'unification

Nous allons essayer d'unifier les termes $= \{|\{x\}_{K_b}, y|A|\} = \{|\{a\}_{K_b}, time(t)|B|\}$

Etape 1 :

$$z = \{|u, y|A|\}, z = \{|v, w|B|\}, u = \{x\}_m, v = \{n\}_m, w = time(g), m = K_b, g = tn = a$$

Etape 2

– Utilisation de la Règle 2, Cas 3 : Prenons

$$X_1 = u, y; Y_1 = u, w \text{ et } X_2 = Y_2 = \emptyset$$

Nous choisissons la correspondance de multi-ensemble $\{u, v\}$ et $\{y, w\}$, ce qui donne le système suivant :

$$z = \{|u, y|W\}, u = \{n\}_m, u = \{x\}_m, y = time(g), n = a, m = K_b, A = W, B = W, g = t$$

– Utilisation de la Règle 1 pour u on obtient $x = n$

$$z = \{|u, y|W\}, u = \{n\}_m, y = time(g), n = a, m = K_b, A = W, B = W, g = t$$

nous ne pouvons plus utiliser de règles, il n'y a pas de cycle et le résultat après quelques étapes de calcule est :

$$\sigma = \{x \leftarrow a, y \leftarrow time(t), A \leftarrow W, B \leftarrow W\}$$

4.4 Règles de simplification

Soit $\sigma = U(u, v)$ désigne un unificateur calculé par l'algorithme de [2].
Voici les règles de simplification des contraintes du système :

R_1 : $C, T \Vdash u \rightsquigarrow \top$ Si $T \cup \{x | T' \Vdash x \in C, T' \subsetneq T\} \vdash u$

R_2 : $C, T \Vdash u \rightsquigarrow_\sigma C\sigma, T\sigma \Vdash u\sigma$ Si $\sigma = U(t, u), t \in St(T), t \neq u; t, u \notin \mathcal{X}$

R_3 : $C, T \Vdash u \rightsquigarrow_\sigma C\sigma, T\sigma \Vdash u\sigma$ Si $\sigma = U(t_1, t_2), t_1, t_2 \in St(T), t_1 \neq t_2; t_1, t_2 \notin \mathcal{X}$

R_4 : $C, T \Vdash \{u\}_v \rightsquigarrow C, T \Vdash u \wedge C, T \Vdash v$

R_5 : $C, T \Vdash f(u_1, ..., u_n) \rightsquigarrow \bigwedge_{i=1}^{n} C, T \Vdash u_i$

R'_5 : $C, T \Vdash \{|u_1, ..., u_{n-1}|u_n|\} \rightsquigarrow \bigwedge_{i=1}^{n} C, T \Vdash u_i$

R_6 : $C, T \Vdash u \rightsquigarrow \bot$ Si $T = \emptyset$ ou $[Var(T, u) = \emptyset$ et $T \nvdash u]$

4.5 Terminaison

Lemme 13 *Les règles de simplification terminent.*

Preuve : il suffit de remarquer que la mesure suivante de C décroît strictement par l'application des règles. $I(C) = (nbv(C), |C|)$ où nbv(C) est le nombre de variables et $|C|$ est la somme des tailles des membres droits de la contrainte. Les paires sont ordonnées lexicographiquement. R_1, R_4, R_5, R'_5 et R_6 font diminuer strictement $|C|$. Quand à R_2 et R_3 elles font diminuer $nbv(C)$ strictement. d'après la Proposition 12. □

4.6 Correction

Lemme 14 *Si $C \rightsquigarrow_\sigma C'$, alors pour toute solution θ de $C', \sigma\theta$ est solution de C.*

Preuve : Pour les règles R_4, R_5 et R'_5 : ceci résulte des capacités de construction de l'intrus.

Pour les règles R_2 et R_3 : cela résulte des définitions et que l'unificateur est correct [2][A.1.2]

Pour la règle R_1 : Il s'agit de montrer que toute solution de C est aussi solution de $T \Vdash u$. Par condition d'application de la règle, si σ est une solution de $C, T\sigma \cup \{x\sigma | T' \Vdash x \in C, T' \subsetneq T\} \vdash u\sigma$ est dérivable. Mais $T' \Vdash x \in C$ entraîne que $T'\sigma \vdash u\sigma$ est dérivable et, comme $T' \subsetneq T, T\sigma \vdash x\sigma$ est dérivable : pour tout $z \in \{x | T' \Vdash x \in C, T' \subsetneq T\}$, $T\sigma \vdash z\sigma$ est dérivable. Donc $T\sigma \cup \{x\sigma | T' \Vdash x \in C, T' \subsetneq T\} \vdash u\sigma \Rightarrow T\sigma \vdash u\sigma$.

□

4.7 Complétude

Minimal gauche : $T_i \vdash u$ est dite minimale à gauche si pour tout $j \leq i$ tel que $T_j \vdash u$

Preuve simple : Une preuve $T_i \vdash u$ est simple si :

1. Aucune branche ne contient deux fois le même noeud.
2. Toutes les sous-preuves sont minimales à gauche.
3. Le lemme de localité fonctionne.

Notation Si C est une contrainte, on dira que le membre de gauche T_i est minimal non résolu si, pour tout T_j tel que $T_j \subsetneq T_i$ et $T_j \Vdash u_j \in C, u_j$ est une variable. On notera $T_i' = T_i \cup \{x | T_j \Vdash x \in C, T_j \subsetneq T_i\}$

Lemme 15 *Pour toute séquence $T_1 \subseteq ... \subseteq T_k$, tout indice i et tout terme u, si $T_i \vdash u$ est prouvable, alors il en existe un preuve simple.*

Lemme 16 *Soit C une contrainte et σ une solution de C. Soit T_i un membre gauche minimal non résolu de C.*

Si $T_i\sigma \vdash u$ a une preuve simple dont la dernière règle est une décomposition ou un axiome alors il existe $t \in St(T_i), t$ non variable, tel que $u = t\sigma$.

Les preuves des lemmes 16 à 18 sont en annexe A.

Lemme 17 *: Soit C une contrainte et σ une solution de C. Soit T_i un membre gauche minimal non résolu de C.*

Si T_i ne contient pas deux sous-termes distincts non variables qui sont unifiables, alors : $\left.\begin{array}{l} T_i\sigma \vdash u\sigma \text{ prouvable} \\ u \in St(T_i), u \text{ non variable} \end{array}\right\} \Rightarrow T_i' \vdash u$ *est prouvable*

Lemme 18 *Si σ est une solution de C et C n'est pas en forme résolue, alors il existe une contrainte C' et des substitutions θ, τ telles que $C \rightsquigarrow_\theta C', \tau$ est une solution de C' et $\sigma = \theta\tau$.*

Lemme 19 *Toute contrainte qui n'est pas en forme résolue est réductible ou bien n'a pas de solution.*

Preuve : C'est un conséquence du lemme 18 : Si C n'est pas sous forme résolue et possède une solution, alors il est possible d'appliquer une règle à C. □

Ce qui conduit au dernier résultat de ce mémoire :

Théorème 20 *La vérification de protocoles XML contre un attaquant actif sur un nombre borné de sessions est décidable.*

5 Conclusion

Nous avons démontré que la vérification classique de protocoles cryptographique avec des contraintes de régularité sur les messages et la vérification de protocoles cryptographiques XML étaient toutes les deux décidables. La prochaine étape est de vérifier qu'un protocole cryptographique XML avec des contraintes de régularité est décidable. Nous pensons que c'est décidable, car il existe une notion d'automate d'arbre pour notre modélisation XML : les automates de Pressburger [7, Chapitre 8]. Puis nous essayerons de modéliser la vérification des protocoles XML avec des contraintes aussi expressives que le WS-SecurityPolicy du W3C.

Tout ceci est dans le but de vraiment vérifier les services web, et d'arriver à une implémentation de vérificateur automatique qui permet de vérifier l'existence d'une attaque. Contrairement, à Tulafale[13] qui utilise les clauses de Horn pour vérifier un nombre quelconque de sessions, notre système sera limité a un nombre borné, mais donnera une procédure de décision (et non de semi-décision comme Tulafale).

ANNEXE

A Preuves de complétude

Lemme 16 *Soit C une contrainte et σ une solution de C. Soit T_i un membre gauche minimal non résolu de C.*

Si $T_i\sigma \vdash u$ a une preuve simple dont la dernière règle est une décomposition ou un axiome alors il existe $t \in St(T_i), t$ non variable, tel que $u = t\sigma$.

Preuve : On procède par récurrence sur la preuve.

La dernière règle est un axiome : Dans ce cas, $u \in T_i\sigma$. donc il existe $t \in T_i$ tel que $t\sigma = u$.
Montrons que t n'est pas une variable $^{(1)}$: Si t est une variable, $T_t \subsetneq T_i$ (par définition du système de contraintes) et que $T_t \vdash v \in C$ avec $x \in var(v)$. On aurait par hypothèse de lemme, $T_t \Vdash t \in C$. et donc $T_{t\sigma} \vdash t\sigma$ prouvable soit $T_{t\sigma} \vdash u$. Contradiction avec la minimalité.

La dernière règle est un déchiffrement :

$$\frac{T_i\sigma \vdash \{u\}_v \qquad T_i\sigma \vdash v}{T_i\sigma \vdash u}$$

Par simplicité de la preuve, la dernière règle appliquée pour obtenir $T_i\sigma \vdash \{u\}_v$ est une règle de décomposition ou un axiome. Donc par hypothèse de récurrence, il existe un terme non variable $t \in St(T_i)$ tel que $t\sigma = \{u\}_v$. Comme t n'est pas une variable, $t = \{t_1\}_{t_2}$. De plus $t_1\sigma = u$ (car $\{u\}_v = \{t_1\sigma\}_{t_2\sigma}$). Comme pour $^{(1)}$ t_1 ne peut pas être une variable, $t_1 \in St(T_i)$ et $t_1\sigma = u$.

La dernière règle est une projection :

$$\frac{T_i\sigma \vdash f(x_1, ..., u, ..., x_n)}{T_i\sigma \vdash u}$$

Comme dans le cas précédent, la dernière règle appliquée pour obtenir $T_i\sigma \vdash f(x_1, ..., u, , ..., x_n)$ ne peut être qu'une décomposition ou un axiome. Par hypothèse de récurrence, il existe $t \in St(T_i)$ non variable tel que $t\sigma = f(x_1, ..., u, ..., x_n)$. Donc $t = f(t1, ..., t_{n+1})$ et donc $\exists j \in [1..n+1]|t_j = u\sigma$. Comme pour $^{(1)}, t_j$ ne peut pas être une variable. De plus $t_j \in St(T_i)$ et $t_j\sigma = u$.

La dernière règle est une décomposition multiset : Tout d'abord u ne peut pas être une variable car il existerait une preuve plus petite. Donc par notation, u ne peut être une extension.

$$\frac{T_i\sigma \vdash \{|x_1, ..., u, ..., x_{n-1}|x_n|\}}{T_i\sigma \vdash u}$$

Comme dans le cas précédent, la dernière règle appliquée pour obtenir $T_i\sigma \vdash \{|x_1, ..., u, ..., x_{n-1}|x_n|\}$ ne peut être qu'une décomposition ou un axiome. Par hypothèse de récurrence, il existe $t \in St(T_i)$ non variable tel que $t\sigma = \{|x_1, ..., u, ..., x_{n-1}|x_n|\}$. Donc $t = \{|t_1, ..., t_n|t_{n+1}|\}$ et donc $\exists j \in [1..n]|t_j = u\sigma$. Comme pour $^{(1)}, t_j$ ne peut pas être une variable. De plus $t_j \in St(T_i)$ et $t_j\sigma = u$

□

Lemme 17 *: Soit C une contrainte et σ une solution de C. Soit T_i un membre gauche minimal non résolu de C.*

Si T_i ne contient pas deux sous-termes distincts non variables qui sont unifiables, alors : $\left.\begin{array}{l} T_i\sigma \vdash u\sigma \text{ prouvable} \\ u \in St(T_i), u \text{ non variable} \end{array}\right\} \Rightarrow T_i' \vdash u$ *est prouvable*

Preuve : D'après le lemme 15 on sait qu'il existe une preuve simple de type $T_i\sigma \vdash u\sigma$. On prouve en faisant un récurrence sur la preuve.

Si la dernière règle est un Axiome : Par définition on a deux cas

- $u \in T_i$: Ce qui termine la preuve
- $\exists t \in T_i, t \neq u$, **et** $t\sigma = u\sigma$: Si t est une variable alors, $T_t \Vdash t \in C$ et donc $T_t\sigma \vdash u\sigma$ prouvable donc il existe une preuve plus simple. Et donc il y a deux sous-termes distincts non variables unifiables (t et u).

Si la dernière règle est un déchiffrement :

$$\frac{T_i\sigma \vdash \{u\sigma\}_v \qquad T_i\sigma \vdash v}{T_i\sigma \vdash u\sigma}$$

Par simplicité de la preuve, la dernière règle appliquée pour obtenir $T_i\sigma \vdash \{u\sigma\}_v$ ne peut être qu'un axiome ou une décomposition.
On peut donc appliquer le lemme 16 : il existe $t \in St(T_i), t$ non variable tel que $t\sigma = \{u\sigma\}_v$.
Comme t n'est pas une variable : $t = \{t_1\}_{t_2}$, et donc $t\sigma = \{t_1\sigma\}_{t_2\sigma}$
On sait que $t_1\sigma = u\sigma$ encore par simplicité de la preuve on sait que t_1 n'est pas variable. Or $u, t_1 \in St(T_i)$ d'après les hypothèses on a u et t_1 non unifiable et non variables comme $u\sigma = t_1\sigma$ on a donc $u = t_1$. d'après les hypothèses de récurrences : $T_i' \vdash t = \{u\}_{t_2}$.
Il reste deux cas :

Cas 1 Si t_2 n'est pas une variable, comme $t_2\sigma = v$ et $T_i\sigma \vdash t_2\sigma$ d'après l'hypothèse de récurrence $T_i' \vdash t_2$ on applique :

$$\frac{T_i' \vdash \{u\}_{t_2} \qquad T_i' \vdash t_2}{T_i' \vdash u}$$

Cas 2 Si t_2 variable,$T_i' \vdash t_2$ est prouvable.

Si la dernière règle est une décomposition multi-set : Tout d'abord $u\sigma$ ne peut pas être une variable car il existerait une preuve plus petite. Donc par notation, u ne peut être une extension.

$$\frac{T_i\sigma \vdash \{|u_1, ..., u\sigma, ..., u_{n-1}|u_n|\}}{T_i\sigma \vdash u\sigma}$$

Par simplicité de la preuve, la dernière règle appliquée pour obtenir $T_i\sigma \vdash \{|u_1, ..., u\sigma, ..., u_{n-1}|u_n|\}$ ne peut être qu'un axiome ou une décomposition.
On peut donc appliquer le lemme 16 : il existe $t \in St(T_i), t$ non variable

tel que $t\sigma = \{|u_1, ..., u\sigma, ..., u_{n-1}|u_n|\}$.
Comme t n'est pas une variable : $t = \{|t_1, ..., t_n|t_{n+1}|\}$. et donc $t\sigma = \{|t_1\sigma, ..., t_n\sigma|t_{n+1}\sigma|\}$
On sait que $\exists j \in [1..n]|u\sigma = t_j\sigma$ encore par simplicité de la preuve on sait que t_j n'est pas variable. Or $u, t_j \in St(T_i)$ d'après les hypothèses on a u et t_j non unifiables et non variables comme $u\sigma = t_j\sigma$ on a donc $u = t_j$.
d'après les hypothèses de récurrences : $T'_i \vdash t = \{|t_1, ..., u, ..., t_{n-1}|t_n|\}$ donc $T'_i \vdash u$.

Si la dernière règle est une projection : Même démonstration que la décomposition multi-ensemble.

Si la dernière règle est une composition : Exemple la composition de multi-ensemble.

$$\frac{T_i\sigma \vdash u_1 \qquad ... \qquad T_i\sigma \vdash u_n}{T_i\sigma \vdash u\sigma}$$

u n'est pas une variable et appartient à $St(T_i)$, donc $u = \{|t_1, ..., t_{n-1}|t_n|\}$ avec $t_\rho(j)\sigma = u_j, j \in [1..n-1], t_n\sigma = u_n$ et ρ une permutation. Pour chaque termes, il y a deux cas :

Si t_j n'est pas une variable : On a que $t_j \in St(T_i)$ ($u \in St(T_i)$) et $T_i\sigma \vdash t_j\sigma$ ($T_i\sigma \vdash u$). Par hypothèse de récurrence on a $T'_i \vdash t_j$.

Si t_j est une variable : $T'_i \vdash t_j$ prouvable.

Dans tous les cas, par composition $T'_i \vdash u$.

□

Lemme 18 *Si σ est une solution de C et C n'est pas en forme résolue, alors il existe une contrainte C' et des substitutions θ, τ telles que $C \rightsquigarrow_\theta C', \tau$ est une solution de C' et $\sigma = \theta\tau$.*

Preuve : Comme C n'est pas en forme résolue, il existe un membre droit minmal T_i tel que $T_i \Vdash u \in C$ et $u \notin \mathcal{X}$.
Comme σ est une solution de $C, T_i\sigma \vdash u\sigma$ est prouvable, et par une preuve simple (lemme 15)
On procède par récurrence sur cette preuve.

Cas 1 : la dernière règle est une décomposition Dans ce cas, par le lemme 16, $\exists t \in St(T_i)$ non variable et $t\sigma(= t\sigma\sigma) = u\sigma$. On a deux cas :

Si $t \neq u$, alors on peut appliquer la règle R_2 à $T_i \vdash u$, obtenant une contrainte qui satisfait aux conditions du lemme. Car l'unificateur est complet[2][A.1.2].

Sinon , $u \in St(T_i)$. Et, d'après le lemme 17, si u et t sont unifiables on applique la règle R_3, sinon on applique la règle R_3.

Cas 2 : la dernière règle est une composition Par exemple un multi-ensemble.

$$\frac{T_i\sigma \vdash u_1 \qquad ... \qquad T_i\sigma \vdash u_n}{T_i\sigma \vdash u\sigma}$$

et $u\sigma = \{|u_1, ..., u_{n-1}|u_n|\}$. comme u n'est pas une variable, $u = \{|t_1, ..., t_{n-1}|t_n|\}$ et $t_{\rho(j)}\sigma = u_j, j \in [1..n-1], t_n\sigma = u_n$ et ρ une permutation. En appliquant la règle R'_5 à C, on obtient $\bigwedge_{i=1}^{n} T_i \vdash t_j$, et on a bien σ solution.

Même démonstration sans la permutation pour la composition de fonctions et le chiffrement.

□

Références

[1] www.journaldunet.com/solutions/0705/070521-wep-tjx-failles-wifi.shtml

[2] Evgeny Dantsin & Andrei Voronkov, Bag and Set Unification, UPMAIL Technical Report, 1997.

[3] Hubert Comon-Lundh, Résolution de contraintes et recherche d'attaque pour un nombre borné de sessions, 5 février 2004.

[4] Pascal Lafourcade, Vérification de protocoles cryptographique en présence de théorie équationnelles, 25 septembre 2006.

[5] Fluhrer, Mantin et Shamir, Weaknesses in the key scheduling algorithm of RC4, English Annual Workshop on Selected Areas in Cryptography, aout 2001.

[6] Douglas Stinson, Cryptographie, Théorie et pratique,Vuibert, 2003.

[7] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison & Marc Tommasi, Tree Automata Technique and Applications, www.grappa.univ-lille3.fr/tata/.

[8] D. Dolev & A.C. Yao. On the security of public key protocols. In Proceeding of the 22nd Symp. on Fondations of ComputerScience, Pages 350-357, Nashville, Tenessee, USA, 1981. IEEE Computer Society Press.

[9] www.w3.org/2002/ws/ et www.w3.org/TR/soap/

[10] Web Services Security Policy Langage (WS-SecurityPolicy) , specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf , 11 juillet 2005.

[11] Histoire des codes secrets, S. Singh, Livre de Poche ,3 septembre 2001.

[12] N.Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols, In Proc. Workshop on Formal Methods and Security protocols, Trento, Italy 1999.

[13] Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Riccardo Pucella, TulaFale : A Security Tool for Web Services, Microsoft.