Machine Translated by Google

Disponible en ligne sur www.sciencedirect.com

ScienceDirect

Information et calcul 205 (2007) 1741–1776

Information and Computation

www.elsevier.com/locate/ic

Analyse des protocoles cryptographiques à l'aide de transducteurs

Ralf Küsters a,*, Thomas Wilke b

aETH Zurich, Institut d'informatique théorique, IFW E 48.3, Haldeneggsteig 4, CH-8092 Zurich, Suisse bInstitut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Allemagne

Reçu le 14 mars 2006; révisé le 31 juillet 2007
Disponible en ligne le 14 septembre 2007

Abstrait

Les protocoles cryptographiques peuvent être divisés en (1) protocoles où les étapes du protocole sont simples d'un point de vue informatique et peuvent donc être modélisées par des moyens simples, par exemple, par des règles de réécriture uniques - nous appelons ces protocoles non-bouclants - et (2) protocoles, tels que les protocoles de groupe, où les étapes du protocole sont complexes et impliquent généralement un calcul itératif ou récursif - nous les appelons récursifs. Alors que l'on sait beaucoup de choses sur la décidabilité de la sécurité pour les protocoles non-bouclants, on en sait peu sur les protocoles récursifs. Dans cet article, nous prouvons la décidabilité de la sécurité (par rapport à l'intrus Dolev-Yao standard) pour une classe de base de protocoles récursifs et l'indécidabilité pour plusieurs extensions. L'ingrédient clé de notre modèle de protocole est des transducteurs d'arbres spécifiquement conçus qui fonctionnent sur des signatures infinies et ont la capacité de générer de nouvelles constantes (qui nous permettent d'imiter la génération de clés). Le résultat de décidabilité est basé sur une construction théorique des automates qui implique une nouvelle notion de régularité, conçue pour fonctionner correctement avec les signatures infinies que nous utilisons. © 2007 Elsevier Inc. Tous droits réservés.

Mots clés : Protocoles cryptographiques ; Analyse automatique ; Décidabilité ; Transducteurs

1. Introduction

Dans la plupart des protocoles cryptographiques, les principaux sont décrits par une séquence fixe de ce que nous appelons des actions de réception-envoi . Lorsqu'il exécute une telle action, un principal reçoit un message de l'environnement et, après un calcul interne, réagit en renvoyant un message à l'environnement. Les recherches sur l'analyse automatique des protocoles [34,2,5,26] se sont concentrées sur les protocoles où une action de réception-envoi peut être décrite par une seule règle de réécriture de la forme $t \to t$: lors de la réception d'un message m, le message (t) est renvoyé en sortie à condition que soit le matcher pour t et m, c'est-à-dire $(t) = m$. En d'autres termes, un message d'entrée est traité en appliquant la règle de réécriture une fois au niveau supérieur. Nous appelons les actions de réception-envoi de ce type et les protocoles basés sur de telles actions de réception-envoi non-bouclables. Il a été prouvé que pour les protocoles non-bouclables, lorsqu'ils sont analysés par rapport à

Un résumé détaillé de cet article est paru dans STACS 2004 (Küsters et Wilke, 2004). Ce travail a été partiellement financé par la « Deutsche Forschungsgemeinschaft (DFG) » au titre de la bourse KU 1434/4-1.

Auteur correspondant.
Adresses e-mail : ralf.kuesters@inf.ethz.ch (R. Küsters), wilke@ti.informatik.uni-kiel.de (T. Wilke).

0890-5401/$ - voir page de garde © 2007 Elsevier Inc. Tous droits réservés.
doi:10.1016/j.ic.2007.08.003

Machine Translated by Google

à un nombre fini d'actions de réception-envoi et à l'intrus Dolev-Yao standard où la taille du message n'est pas limitée, la sécurité (plus précisément, la confidentialité) est décidable même lorsque les principaux peuvent effectuer des tests d'égalité sur des messages arbitraires [34,2,5,26], les clés complexes sont autorisées [34,5,26], et l'hypothèse d'algèbre à termes libres est assouplie par les propriétés algébriques de divers opérateurs, tels que le ou exclusif (XOR), l'exponentiation Diffie-Hellman et le cryptage RSA [8,13,9,36,10].

La question principale qui nous intéresse dans cet article est de savoir dans quelle mesure la sécurité est décidable pour les protocoles où les actions de réception-envoi sont complexes et impliquent généralement un calcul itératif ou récursif ; nous appelons ces actions de réception-envoi et les protocoles contenant de telles actions récursifs.

Pour illustrer le type d'actions de réception-envoi effectuées dans les protocoles récursifs, considérons le serveur de distribution de clés S du protocole d'authentification récursive (RA) [7]. Dans ce protocole, le serveur S doit effectuer l' action de réception-envoi récursive suivante : le serveur S reçoit d'abord une séquence a priori illimitée de requêtes de paires de principaux qui souhaitent partager des clés de session. Ensuite, S génère des clés de session et envoie enfin une séquence de certificats (correspondant aux requêtes) contenant les clés de session. Les actions de réception-envoi de ce type sont typiques des protocoles de groupe, mais se produisent également dans des protocoles tels que le protocole Internet Key Exchange (IKE) - voir [25] pour une description de certains protocoles récursifs. Comme souligné par [25] et illustré dans [39,17], la modélisation de la récursivité est pertinente pour la sécurité.

Une façon naturelle de décrire les actions de réception-envoi récursives est d'utiliser des transducteurs d'arbre, qui étendent la classe des transductions exprimables par des règles de réécriture simples (avec un côté gauche linéaire). Plus précisément, pour étudier la décidabilité, nous introduisons dans la section 2 des transducteurs d'arbre descendants non déterministes (TTAC) avec des transitions anticipées et ε qui fonctionnent avec une signature contenant un ensemble infini de ce que nous appelons des constantes anonymes (AC), sur lesquelles les TTAC n'ont qu'un contrôle très limité. Les TTAC peuvent générer de nouvelles constantes (anonymes), une fonctionnalité souvent nécessaire pour modéliser les actions de réception-envoi récursives ; dans le protocole RA, par exemple, le serveur de distribution de clés doit générer (un nombre a priori illimité de) clés de session.

Le principal résultat de cet article est que (1) la sécurité (pour un nombre fini d'actions de réception–envoi, de clés atomiques et de l'intrus Dolev–Yao standard où la taille du message n'est pas limitée) est décidable si les actions de réception–envoi sont modélisées par des TTAC (section 5), et que (2) certaines caractéristiques des modèles pour les protocoles non-bouclables ne peuvent pas être ajoutées sans perdre la décidabilité : dès que les TTAC sont équipés de la capacité d'effectuer des tests d'égalité entre des messages arbitraires, dès que des clés complexes sont autorisées, ou dès que l'hypothèse d'algèbre à termes libres est assouplie en ajoutant XOR ou l'exponentiation Diffie-Hellman, la sécurité est indécidable (section 6).

Les résultats d'indécidabilité sont obtenus par des réductions du problème de correspondance de Post. Le résultat de décidabilité est obtenu en deux étapes. Tout d'abord, nous montrons que les TTAC sont suffisamment puissants pour simuler l'intrus. Cela nous permet de décrire les attaques comme la composition de transducteurs. Nous pouvons ensuite réduire le problème de sécurité au problème du mot de pré-image itéré pour les TTAC, dont nous montrons qu'ils sont décidables (section 3). Étant donné un terme t, un « ensemble régulier » R de termes et une séquence de TTAC, le problème du mot de pré-image itéré demande si sur l'entrée t la composition des TTAC peut produire une sortie dans R. Ici, « ensemble régulier » signifie un ensemble de termes reconnaissables par un nouveau type d'automates d'arbres, les automates d'arbres sur signatures avec constantes anonymes (TAAC), qui peuvent comparer des constantes anonymes pour l'égalité.

1.1. Travaux connexes

Les protocoles récursifs, tels que le protocole RA et le protocole A-GDH.2 [3], ont été analysés manuellement [33] et semi-automatiquement à l'aide de démonstrateurs de théorèmes ou d'outils spéciaux [32,6,24].

La décidabilité des protocoles récursifs a été initialement étudiée dans [20]. Cependant, il existe des différences significatives par rapport au présent article. Tout d'abord, dans [20], des transducteurs de mots sont utilisés, qui sont moins puissants que les transducteurs d'arbre. En conséquence, les transducteurs d'arbre fournissent une image plus claire des différences entre les protocoles récursifs et non-bouclés, et permettent également de tracer une limite plus étroite de décidabilité. Deuxièmement, la génération de nouvelles constantes (par exemple, des clés de session) n'a pas été prise en compte dans [20]. Troisièmement, les modèles basés sur TTAC de protocoles (récursifs) sont en général beaucoup plus précis que les modèles basés sur des transducteurs de mots car les clés de session peuvent être générées et les nonces n'ont pas nécessairement besoin d'être typés ; dans la section 7, cela est illustré pour le protocole RA. Quatrièmement, les techniques de preuve employées sont différentes. Dans [20], un argument de pompage assez complexe et technique est utilisé pour obtenir la décidabilité puisque les transducteurs de mots ne sont pas assez puissants pour simuler l'intrus. Dans le présent article,

Machine Translated by Google

la caractérisation des attaques en termes de composition des transducteurs permet une preuve plus élégante, et les constantes anonymes présentent un défi complètement nouveau.

Récemment, Truderung [38] a proposé un modèle alternatif pour les protocoles récursifs dans lequel il montre que la sécurité est NEXPTIME-complète. Au lieu de transducteurs, il utilise ce qu'il appelle des théories de sélection, qui sont certaines classes de clauses de Horn, pour modéliser les actions récursives de réception-envoi. L'expressivité de ces théories est orthogonale à l'expressivité de nos transducteurs. D'une part, la sélection des théories ne peut produire que des séquences de messages structurés, tandis que les transducteurs peuvent produire des messages beaucoup plus complexes. Par exemple, les transducteurs peut produire une sortie de taille indépendante de la taille de l'entrée. Dans le modèle de Truderung, la taille de sortie est linéaire la taille de l'entrée. De plus, de nouvelles constantes ne peuvent pas être générées et, par conséquent, la génération de clés de session ne peut être qu'approximative. D'autre part, la sélection de théories permet de tester l'égalité de messages arbitraires. Les techniques de preuve appliquées par Truderung sont très différentes de celles employées dans le présent article. Alors que nous, comme expliqué, utiliser des techniques théoriques d'automates, les techniques employées par Truderung sont plus proches de celles de la non-bouclage protocoles. Dans [21], le modèle de Truderung a été étendu pour inclure le OU exclusif (XOR) et la décidabilité et des résultats d'indécidabilité ont été montrés dans ce modèle étendu.

Dans divers articles, des techniques de théorie des automates ont été appliquées à l'analyse de protocoles cryptographiques (voir, par exemple, [27,18,19,12]). Cependant, ces travaux visent à analyser les protocoles non-bouclants par rapport à un nombre illimité de sessions et ne semblent pas s'appliquer de manière évidente aux protocoles récursifs. à notre connaissance, le travail de [20] et le présent travail sont les premiers à utiliser des transducteurs (sur une plage infinie) signatures) pour l'analyse de protocole. Automates et transducteurs sur des signatures infinies (bien que très différentes (parmi ceux considérés ici) ont été étudiés dans le contexte de la vérification de type et de l'inférence de type pour XML requêtes avec des valeurs de données (voir, par exemple, [1,29]).

1.2. Structure du document

Dans la section 2, nous introduisons les automates d'arbres mentionnés (TAAC) et les transducteurs (TTAC) sur les signatures avec des constantes anonymes et prouver les propriétés de base. La section 3 fournit la définition de l'image préalable itérée problème de mots et la preuve que ce problème est décidable pour les TTAC. Notre protocole basé sur un transducteur d'arbre Le modèle est présenté dans la section 4. Dans la section 5, nous montrons que la sécurité dans ce modèle est décidable. Le but de La section 6 consiste (1) à discuter brièvement de la relation entre notre modèle et les modèles pour les protocoles sans boucle, et (2) pour prouver les résultats d'indécidabilité susmentionnés. La section 7 contient des modèles formels basés sur TTAC de le protocole d'authentification récursive et le protocole d'authentification à clé publique Needham Schroeder. Nous conclure à la section 8.

1.3. Définitions et notations de base

Un symbole est un objet auquel est attribuée une arité . Un symbole d'arité 0 est appelé une constante (symbole). Une signature est un ensemble de symboles. Lorsque désigne une signature, alors $_n$ désigne l'ensemble des symboles de avec arité n.

L'ensemble des termes sur une signature est noté T. Pour un ensemble C de symboles constants disjoints d'une signature , nous posons T(C) = T C.

On fixe une infinité X de variables parmi lesquelles on trouve x0, x1, x2, ... Pour n 0, on écrit T n pour l'ensemble de tous les termes de T({x0, ... , xn−1}). Un terme t T n est linéaire si chaque xi avec i<n apparaît au plus une fois dans t. Lorsque t T n et t0, ... , tn−1 sont des termes arbitraires, on écrit t[t0, ... en , tn−1] pour le terme qui est obtenu à partir de t par simul- substituant tanément ti à xi, pour tout i<n. Une substitution sur est une fonction : T(X) →T(X) telle que pour chaque terme t, (t) est obtenu à partir de t en substituant simultanément (x) à x, pour tout x X .

On désigne par ε l'ensemble des chaînes finies sur les entiers non négatifs. La chaîne vide est notée ε . d'habitude, v tel que est appelé préfixe de v s'il existe v Comme concaténation de v et v . Un ensemble S est dit préfixe fermé si avec v S tout préfixe de v appartient à S.

Nous utilisons les notions de « terme » et d'« arbre » de manière interchangeable puisqu'un terme peut être vu comme un arbre. Formellement, un arbre est une application à partir d'un ensemble fermé non vide, fini et préfixé S tel que si t() pour un certain n 0 $_n$ et S, alors {i | n − 1}, et si t() S={0, variable, alors {i | i S}= . On appelle S l' ensemble des positions de t et désignons cet ensemble par P(t).

Machine Translated by Google

Pour un terme t et P(t), t| désignera le sous-terme de t à la position , c'est-à-dire P(t|) = { t|( ) = t( ) pour | P(t)} et tout P(t|).

Un sous-ensemble de T × T est appelé une transduction sur . Pour un terme t, on définit (t) = {t | (t, t) }. Si et sont des transductions sur alors leur composition ◦ définit la transduction {(t, t) | t ((t, t) (t, t))}, c'est-à-dire que la composition se lit de droite à gauche. Étant donné une transduction sur et un ensemble R T, la préimage de R sous est l'ensemble (R) = {t | t R avec (t, t) }.
−1

2. Automates d'arbres et transducteurs d'arbres à constantes anonymes

Dans cette section, nous décrivons les modèles d'automates et de transducteurs d'arbres que nous utilisons, de manière totalement indépendante de l'application que nous avons en tête, car ils sont d'intérêt général. Avant de définir nos automates et transducteurs d'arbres, nous introduisons les signatures avec des constantes anonymes.

2.1. Signatures et constantes anonymes

Un couple (,C) constitué d'une signature finie et d'un ensemble arbitraire infini C de symboles constants disjoints de est appelé une signature à constantes anonymes ; les éléments de et C sont appelés respectivement symboles réguliers et constantes anonymes . A une telle signature, on associe la signature C, notée C.
Autrement dit, lorsque nous parlons d'un terme sur (,C), nous entendons un terme sur C. Dans ce qui suit, soit occC(t) l'ensemble des éléments de C qui apparaissent dans le terme t ; de même, soit occC(S) l'ensemble des éléments de C qui apparaissent dans tout terme d'un ensemble de termes S.

2.2. Automates d'arbres sur signatures avec constantes anonymes

Nos automates d'arbres sont des automates d'arbres non déterministes de bas en haut qui acceptent des arbres sur des signatures avec des constantes anonymes ; ils ont un contrôle total sur les symboles réguliers mais seulement un contrôle très limité sur les constantes anonymes. Par exemple, il se produira qu'avec chaque arbre qu'un tel automate accepte, il accepte chaque arbre qui est obtenu à partir de celui-ci simplement en permutant (en renommant de manière cohérente) les constantes anonymes.

Nos automates d'arbres ont deux états distincts, qd et qs , qui sont utilisés comme états initiaux pour l'anonyme - l'automate anonymes : Dans chaque exécution sur un arbre t , attribue d'abord de manière non déterministe qd et qs à des constantes TC, les constantes anonymes qui apparaissent dans t de manière arbitraire sous la restriction qu'au plus une constante anonyme, qui est alors appelée la constante sélectionnée, se voit attribuer qs et que toutes les autres se voient attribuer qd , la valeur par défaut. Notez que différentes occurrences de la même constante anonyme dans t se voient attribuer le même état, qd ou qs . Une fois ces états attribués aux constantes anonymes, le reste de l'exécution de l'automate sur t se déroule de manière ascendante standard.

Formellement, un automate d'arbre (TAAC) sur une signature avec des constantes anonymes (,C) est un tuple

$$A = (Q, q_d, q_s, \ldots, F) \tag{1}$$

où Q est un ensemble fini non vide d'états, qd Q est l' état par défaut, qs Q est l' état de sélection, est un ensemble fini de transitions comme spécifié ci-dessous, et F Q est un ensemble d'états finaux. Si l'ensemble des états finaux est omis, on parle de semi-TAAC.

Formellement, les transitions sont des paires d'un certain type, mais pour une meilleure lecture, nous écrivons une transition (t, q) comme t →q. Il existe deux types de transitions : Une transition consommatrice est de la forme

$$f(q_0, \ldots, q_{n-1}) \to q$$

où f n, q, q0, ... , qn−1 Q ; une ε -transition est de la forme

$$q \to q$$

où q , q Q.

Machine Translated by Google

On dit qu'un TAAC est déterministe s'il ne contient pas de ε -transitions et si pour tout f il existe au n et q0, ... , qn−1 Q plus un q Q tel que f(q0, ... , qn−1) →q .

Chaque TAAC sur une signature à constantes anonymes (,C) définit un ensemble d'arbres issus de T(C). Pour décrire cet ensemble, on considère l'ensemble Q comme un ensemble de constantes et on définit pour chaque terme t T(C Q) l'ensemble [t]A des états que l'automate atteint après avoir lu le terme t.

Nous donnons d'abord une définition inductive pour les termes t T(Q) sans constantes anonymes. Dans ce cas, [t]A est le plus petit ensemble satisfaisant les règles suivantes :

– Si t Q, alors t [t]A.
– Si t = f(t0, ... , tn−1) et qu'il existe q0, ... , qn−1 avec f(q0, ... , qn−1) →q et qi [ti]A pour tout i<n, alors q [t]A.

– Si q [t]A et q →q A la , alors q [t]A.

substitution permise est une fonction :C →{qd , qs } où au plus un élément de C se voit attribuer qs . Maintenant, pour un t T(C Q) arbitraire, on pose

$$[t]A = \bigcup_{\text{autorisé}} [(t)]A.$$

Le langage arborescent reconnu par A est le langage

$$T(A) = \{t \in T(C) \mid F \cap [t]A \neq \emptyset\}.$$

On dit qu'un langage arborescent est C est TAAC-reconnaissable sur (,C) s'il est reconnu par un TAAC sur sur (,C).

Avant de fournir quelques exemples de langues (non) reconnaissables par TAAC, nous introduisons une notation qui nous l'utiliserons plus tard. Étant donné un terme t T^n_C et définit S0, ... , Sn−1 Q, on écrit

$$[t[S_0, \dots, S_{n-1}]]A = \bigcup_{(q_0,\dots,q_{n-1}) \in S_0 \times \dots \times S_{n-1}} [t[q_0, \dots, q_{n-1}]]A.$$

Exemple 1. Supposons que 2 = {f } et = pour tout i /= 2. Soit T= = {f(c, c) | c C}. Ce langage est reconnu par un TAAC avec seulement trois états, disons q0, q1 et q2. Nous choisissons qd = q0 et qs = q1, F = {q2} et n'avons qu'une seule transition, à savoir f(q1, q1) →q2.

Exemple 2. Fixer une signature (,C) quelconque avec des constantes anonymes. Pour tout i 3, soit Ti le langage arborescent sur C qui contient un arbre t ssi en t au moins i constantes anonymes distinctes deux à deux apparaissent. Il est alors facile de voir que Ti est reconnaissable par TAAC sur (,C) pour i 2, mais pas pour i = 3 : pour i = 0, le TAAC doit accepter tous les arbres. Pour i = 1, on définit un TAAC avec un état distinct q = qd = qs auquel sont mappées les constantes anonymes. Le TAAC vérifie si cet état apparaît dans une exécution sur un arbre au moins une fois pour s'assurer que l'arbre contient au moins une constante anonyme. Pour i = 2, le TAAC a des états qd et qs avec qd = qs et vérifie si qd et qs apparaissent tous les deux dans une exécution sur un arbre. Le résultat négatif pour i = 3 découle immédiatement du lemme 3, qui implique en particulier que si un TAAC accepte f(c, g(c , c)) pour les constantes anonymes c, c , c, alors il accepte également l'une des constantes f(c, g(c, c)), f(c, g(c , c)) et f(c , g(c , c)).

Pour énoncer le lemme mentionné, nous avons besoin de la notion de termes c-équivalents. Soient t et t des termes arbitraires et c une constante anonyme. Or, t et t sont c-équivalents s'ils coïncident sauf pour les occurrences de constantes anonymes autres que c, c'est-à-dire que t et t sont c-équivalents s'il existe un terme u et des substitutions et avec (X), (X) C \ {c} tels que (u) = t et (u) = t . Par exemple, f(c, g(c , c)) et f(c, g(c, c)) sont c-équivalents mais f(c, g(c , c)) et f(c , g(c, c)) ne le sont pas.

Lemme 3. Soit T un langage reconnaissable par TAAC et t T. Alors il existe une constante anonyme c telle que t T pour tout arbre t qui est c-équivalent à t.

Machine Translated by Google

Preuve (esquisse). Soit A un TAAC reconnaissant T . Prenons pour c la constante à laquelle est assignée qs dans une exécution acceptante de A sur T . Maintenant, l'affirmation se déroule facilement.

Un autre exemple de langues reconnaissables par TAAC est la classe de langues suivante :

Exemple 4. Fixer une signature (,C) quelconque avec des constantes anonymes. Pour tout i, soit Ti le langage arborescent sur C qui contient un arbre t ssi dans t il y a au moins i occurrences de constantes anonymes (pas nécessairement différentes). Alors, Ti est reconnaissable par le TAAC sur (,C) pour tout i. En effet, Ti est reconnu par le TAAC A = ({0, ... , i, q}, q, q, ,{i}) où pour tout f et états q1, ... , qn {0, ... , i, q}, contient la transition f(q1, ... , qn) →j où j = min(i, #{l | ql =nq} + l {1,...,n},ql=q ql). C'est-à-dire que des constantes anonymes sont assignées à q et A compte le nombre de q jusqu'à i. Si i est atteint à la racine de l'arbre, l'arbre est accepté.

Le TAAC ci-dessus est ce que nous appelons un TAAC faible. Un TAAC est appelé faible (WTAAC) si l'état par défaut et l'état de sélection sont identiques, c'est-à-dire, qd = qs . Cela signifie qu'il n'y a en fait aucun état de sélection. Les WTAAC sont vraiment plus faibles car il est facile de voir que, par exemple, T= n'est pas reconnaissable par WTAAC sur (,C).

Nous concluons cette section en résumant les propriétés de base des TAAC et des WTAAC. Nous commençons par un exemple simple observation, qui peut être prouvée en utilisant une construction d'ensemble de puissance simple.

Lemme 5. Tout TAAC est équivalent à un TAAC déterministe. Il en va de même pour les WTAAC.

Mais observez que pour les TAAC, il y a toujours un non-déterminisme impliqué en raison de la liberté de choisir lequel la constante anonyme est attribuée à qs .

Dans le lemme suivant, nous considérons les propriétés de fermeture des TAAC ainsi que des WTAAC. Comme nous le verrons, le comportement des TAAC est assez différent de celui des automates d'arbres sur des signatures finies.

Lemme 6. Soit (,C) une signature à constantes anonymes. Alors :

(1) L'ensemble des langages arborescents sur (,C) reconnus par WTAAC sur (,C) est fermé par union, intersection, et la complémentation.
(2) L'ensemble des langages arborescents sur (,C) reconnus par les TAAC sur (,C) est fermé sous l'union.
(3) L'ensemble des langages arborescents sur (,C) reconnus par les TAAC sur (,C) est fermé sous complémentation ssi = 0 1. Il en va de même pour l'intersection.

Preuve. (1) Ceci peut être prouvé de la même manière que pour les transducteurs d'arbres ascendants : la fermeture sous union découle de la fermeture sous intersection et complémentation ; la fermeture sous intersection est montrée en utilisant une construction de produit standard ; la fermeture sous complémentation découle du lemme 5, car la complémentation d'un WTAAC déterministe peut être obtenue en complémentant son ensemble d'états finaux.

(2) Soit A0 = (Q0, qd 0, 0, F0) et A1 = (Q1, qd 0 , qs 1 , qs 1, 1, F1) soient des TAAC. Nous voulons montrer que T(A0) T(A1) est reconnu par un TAAC. Fondamentalement, nous construisons l'automate produit de A0 et A1 et utilisons l'équivalence suivante : Étant donné un arbre t, il existe une substitution permise telle que A0 ou A1 accepte (t) ssi il existe une substitution permise 0 telle que A0 accepte (t) ou il existe une substitution permise 1 telle que A1 accepte 1(t).

Plus précisément, pour construire un TAAC qui reconnaît T(A0) T(A1), nous rendons d'abord A0 et A1 complets comme suit : (i) Nous ajoutons de nouveaux états s0 et s1 à A0 et A1, respectivement, et ii) pour tout f , qn−1 Qi {si} n, q0, ... , qn−1) →si à l'ensemble des transitions de Ai pour i < 2. Dans la 1. Notons les automates résultants par A2 nous ajoutons f(q0, ... et A3, respectivement. Nous avons clairement T(Ai) = T(Ai+2) deuxième étape, nous construisons l'automate produit suivant :

$$(Q2 \times Q3, (qd\ 2, qd\ ), \{qs\ 2, qs\ 3), (F0 \times Q3) \quad (Q2 \times F1))$$

avec (q2, q3) →(q 2, q 3) si q2 →q 2 2 et q3 = q 3, qn 3)) →(q, q)si f(q1 , qn 2) →q 2 et f(q1 q2 = q 2 , qn et q3 →q 3 ou 3, et f((q1 2, q1 3), ... , (qn 2, reconnaît T(A2) T(A3) 2, T(A0) T(A1). 3, ... 3) →q 3 pour f n. De toute évidence, le TAAC résultant

Machine Translated by Google

(3) Tout d'abord, si = 0 1 etAest un TAAC sur(,C), alors T(A) = T((Q, qd , qd , , F)) T((Q, qs , qs , , F)) puisque les termes sur (,C) ne peuvent contenir au plus qu'une constante anonyme, et donc, dans une exécution sur un tel terme, cette constante est soit assignée à qd soit à qs . Ces deux automates sont des WTAAC, donc par (1). nous savons que T(A) est reconnu par un WTAAC, et, encore par 1, son complément l'est aussi. Cela prouve une direction de l'implication.

Nous montrons maintenant que les TAAC ne sont pas fermés par complémentation et intersection s'il existe un symbole binaire dans , disons f · Il est simple d'étendre cela à toute signature avec au moins un symbole d'arité 2. Dans ce qui suit, soit #c(t) le nombre d'occurrences de c dans t.

Pour chaque n 1, nous définissons le langage arborescent

$$L_n = \{f(t, t') \in T_C \mid c(c \in C \quad \#c(t) = \#c(t') \bmod n)\}.$$

Il est facile de voir que Ln est reconnaissable par TAAC pour tout n 1 grâce à une construction similaire à l'exemple 4.
Cependant, nous montrons que ni $L_2 = T_C \setminus L_2$ ni $L_2 \cap L_3$ ne sont reconnaissables par TAAC.

Par contradiction, supposons que $T = T_C \setminus L_2$ soit reconnu par un TAAC A comme dans (1). Alors cet automate accepterait le terme t = f(f(c0, c1), f(c0, c1)) pour deux constantes anonymes distinctes c0 et c1. Soit c une constante telle que l'affectation qs →c donne une suite acceptante pour t. On procède par une distinction de cas. Si c / {c0, c1}, alors A a une suite acceptante sur t qui ne fait pas de distinction entre c0 et c1, et donc, f(f(c0, c0), f(c0, c1)) T — une contradiction. Si c = c0, on a f(f(c0, c1), f(c0, c2)) T pour une nouvelle constante c2 — une contradiction. Et si c = c1, nous avons f(f(c0, c1), f(c2, c1)) T pour une nouvelle constante c2 — encore une contradiction.

Supposons maintenant qu'il existe un TAACA reconnaissant $L_2 \cap L_3$. Pour c C et n 2, let $n = f(c, f(c, \cdots f(c, c))$), $f(t_c^6)$) = n. Soient c0 et c1 deux constantes anonymes distinctes. De toute évidence, t = f(f(t3 tel que #c(t_{c0}^{n}, t_{c1}^{2} t_{c0}^{4}, t_{c1}^{4})) $L_2 \cap L_3$. Comme ci-dessus, en considérant différents cas pour c, on montre que des variantes de t sont reconnues par A bien qu'elles n'appartiennent pas à $L_2 \cap L_3$, ce qui conduit à une contradiction.

Nous notons enfin que pour les TAAC le problème du mot et du vide sont décidables :

Lemme 7. Le mot et le problème du vide sont décidables pour les TAAC, et donc pour les WTAAC.

Preuve. Pour le problème du mot, qui demande si étant donné un terme et un TAAC, le TAAC reconnaît le terme, c'est évident. Pour le problème de la vacuité, qui demande si étant donné un TAAC, le langage reconnu par le TAAC est vide, on peut montrer par l'argument de pompage habituel sur les automates ascendants à arbres que si un TAAC reconnaît un arbre, alors aussi un arbre de profondeur limitée par le nombre d'états du TAAC, et, clairement, seules deux constantes anonymes différentes fixes doivent être considérées (une pour qs et une pour qd). Maintenant, la décidabilité du problème de la vacuité s'ensuit facilement.

2.3. Transducteurs d'arbres sur des signatures avec des constantes anonymes

Les transducteurs d'arbres sont de différentes sortes. Notre modèle est conçu de telle manière que (1) l'image préliminaire d'un langage reconnaissable par TAAC soit à nouveau reconnaissable par TAAC et (2) nous pouvons (facilement) modéliser les protocoles cryptographiques et l'adversaire que nous voulons. Ces deux objectifs sont opposés l'un à l'autre : pour atteindre (1), le modèle doit être faible, pour atteindre (2), il doit être fort. Un aspect important de (2) est qu'il sera nécessaire qu'un nombre illimité de constantes anonymes puissent être introduites par un transducteur d'arbre, mais seulement de manière très faible.

Notre modèle est un transducteur d'arbre de haut en bas, c'est-à-dire qu'un arbre donné est transformé en un nouvel arbre selon certaines règles de réécriture, qui s'appliquent de la racine de l'arbre à ses feuilles. Elle présente plusieurs particularités :

– un look-ahead WTAAC, – un
mécanisme de génération de nouvelles constantes anonymes, et – un registre
pour une constante anonyme.

De plus, nos transducteurs d'arbres peuvent être non déterministes et peuvent contenir des transitions ε .

Pour définir nos transducteurs, nous avons besoin de quelques notations supplémentaires. Nous fixons une signature (,C) avec des constantes anonymes et un ensemble fini S d'états, dont nous considérons les éléments comme des symboles binaires. Nous supposons que l'on nous donne un ensemble

Machine Translated by Google

V = {vR, vN } de deux variables pour les constantes anonymes : vR représente le registre susmentionné, vN fait référence à une constante anonyme nouvellement générée.

Un terme d'état est de la forme s(z, t) pour s S, z C { , vR, vN } et t T(C X). Le terme t est appelé le noyau terme du terme d'état. Si z appartient à un ensemble D C { , vR, vN }, alors nous disons que s(z, t) est un terme d'état D.

Intuitivement, un terme d'état de la forme s(, t) ou s(c, t) avec c C fait partie d'une configuration d'un transducteur et signifie que le transducteur est sur le point de lire t en commençant dans l'état s où le registre ne stocke pas de valeur ou stocke la constante anonyme c, respectivement. Pour décrire les transitions, nous utilisons des termes d'état de la forme s(vR, t), s(vN , t), et encore s(, t), mais pas s(c, t). Nous définissons maintenant formellement nos transducteurs d'arbres et leur calcul, ainsi que exemples.

Formellement, un transducteur d'arbre (TTAC) sur une signature avec des constantes anonymes (,C) est un tuple

$$T = (S, I, A,) \tag{2}$$

où S est un ensemble fini d' états, I S est un ensemble d' états initiaux, A est un semi WTAAC sur (,C), le look-ahead automate, et est un ensemble fini de transitions comme décrit ci-dessous.

Une transition est de la forme

$$s(z, t) \to q\ t[vR, vN , t_{0, \dots} , t_{r-1}] \tag{3}$$

où

(1) q Q est l'anticipation, avec Q l'ensemble des états de A,
(2) s(z, t) est un terme d'état {vR, } (rappelons que cela signifie que z = vR ou z =) avec t T n et t linéaire,
(3) t T^{r+2} (pas nécessairement linéaire), où vR n'apparaît pas en t [vR, vN est , $t_{0, \dots}$, t_{r-1}] si z = , et
(4) chaque t_i soit une variable apparaissant en t, soit un terme d'état {z, vN , } dont le terme central est un sous-terme de t.

On remarque que la condition de (3) n'est pas une restriction réelle, elle sert juste à bien définir le formalisme : si z = , le registre n'est pas défini, ce qui signifie qu'il n'a aucun sens d'utiliser son contenu (vR) sur le côté droit de la règle, donc vR ne devrait pas y apparaître.

Lorsque vN se produit dans $t[vR, t_0, \dots, t_{r-1}]$, alors la transition est dite générative, et non générative dans le cas contraire. Parfois, nous omettons le regard vers l'avant q lorsque nous écrivons des transitions. Cela revient à supposer que le regard vers l'avant est un état q dans lequel A accepte tous les termes. Si A ne contient pas un tel état, alors A peut être étendu en conséquence. Pour s S, on note T(s) le transducteur T avec s comme seul état initial.

Avant de définir formellement le calcul des TTAC, considérons un exemple simple de TTAC et de ses calcul (voir l'exemple 9 pour un cadre plus complexe).

Exemple 8. Soit 0 = {d}, 1 = {f }, 2 = {g} et = 0 1 2. On considère un TTAC T sur (,C) avec un seul état, pas d'anticipation et les transitions suivantes :

$$s(, f(x)) \to g(s(vN , x), vN), \tag{4}$$
$$s(vR, f(x)) \to g(s(vR, x), vR), s(vR, \tag{5}$$
$$d) \to vR. \tag{6}$$

Supposons que t = f(f(d))est donné en entrée à T. Ensuite, la configuration initiale de Tiss(, t), ce qui signifie que Tis dans l'état s, est sur le point de lire t, et ne contient pas de constante anonyme dans son registre. La seule transition qui peut être appliqué à cette configuration est (4), où x est remplacé par f(d) et vN est remplacé par un constante, disons c. Le résultat de l'application de (4) à s(, t) est la nouvelle configuration g(s(c, f(d)), c). Maintenant, (5) peut être appliqué à s(c, f(d)) où vR, la variable de registre, est remplacée par c et x par d. (En général, l'intermédiaire la sortie produite par un TTAC peut contenir plusieurs termes d'état et des transitions sont appliquées à ces termes d'état indépendamment.) Le résultat de l'application de (5) à s(c, f(d)) est g(s(c, d), c), et la sortie globale produite par T jusqu'à présent est g(g(s(c, d), c), c). À ce stade, (6) peut être appliqué à s(c, d) donnant la sortie globale g(g(c, c), c). Ce terme ne contient pas de terme d'état et est considéré comme une sortie de T sur l'entrée t. Notez que puisque T aurait pu générer

Machine Translated by Google

autres constantes anonymes que c dans la première étape du calcul, tous les termes de la forme g(g(c , c), c) avec c C sont des sorties possibles de T.

Le fait que les TTAC puissent produire plusieurs sorties différentes pour la même entrée n'est pas seulement dû au fait que les constantes anonymes ne peuvent pas être distinguées. Même modulo le renommage des constantes anonymes, étant donné un terme d'entrée, plusieurs sorties peuvent être produites par un TTAC car les TTAC peuvent être non déterministes. De plus, comme les TTAC peuvent contenir des transitions qui ne consomment pas de symboles d'entrée, l'ensemble des sorties peut être infini même modulo le renommage des constantes anonymes. Considérons, par exemple, la transition

$$s(vR, f(x)) \to g(s(vR, f(x)), vR). \tag{7}$$

Cette transition fonctionne comme (5), mais avec une différence : elle ne consomme pas le symbole d'entrée f . En ajoutant cette transition à T ci-dessus et en connaissant le terme d'entrée t ci-dessus, tous les termes de la forme

$$\underbrace{g(g(\cdots g(c, c)\cdots), c), c)}_{n \text{ occurrences de } g}$$

pour n 2 sont des sorties possibles de T.

Formellement, le calcul effectué par un TTAC est décrit par une séquence d'étapes de réécriture. La relation de réécriture correspondante est définie par rapport à un sous-ensemble U C de constantes anonymes pour garantir que chaque fois que le TTAC est censé générer une nouvelle constante, cette constante n'appartient pas à U. Plus tard, U sera l'ensemble des constantes anonymes dans le terme d'entrée, ce qui garantit alors que les constantes anonymes générées par le TTAC sont différentes de celles présentes dans l'entrée.

Pour définir U , supposons que l'on nous donne un terme u0 = u1[s(c, u2)] où u2 = t[t0, ... , tn−1] TC , et donc, t TC ({x0, ... , xn−1}) (voir p. 5), et u1 est linéaire, et une transition comme dans (3) avec z = vR. Soit la substitution définie par (xi) = ti. Alors, si q [u2]A (c'est-à-dire, l'entrée courante satisfait le look-ahead),

$$u0 \ U \ u1[t\,[c, c\,, (t_0), \ldots, (t_{r-1})]]$$

pour tout c C \ (occC(u0) U). Observez que si est non génératif, c et U ne sont pas pertinents. Notez également que la constante anonyme nouvellement générée n'apparaît pas dans U et dans le terme de sortie calculé jusqu'à présent. L'étape de réécriture dans le cas u0 = u1[s(, u2)] est définie de la même manière, mais il est nécessaire que z = .

Une séquence s(, t) U t1 U t2 U ... U t avec t et t termes sur (,C) est appelée un calcul.

Soit t une forme abrégée de forme. La relation $\to_{T,U}$ désigne la fermeture réflexive transitive de U . On pose $t \in T_U(t)$ occC(t)

sur T(C) défini par le TTAC T est

$$_T = \{(t, t\,) \quad TC \times TC \mid \quad s(s \quad l \quad s(\ , t) \qquad t\,)\}.$$

On dit qu'une transduction sur (,C) est TTAC-réalisable s'il existe un TTAC T tel que $= {}_T$. Nous appelons deux TTAC équivalents s'ils réalisent la même transduction.

Regardons un autre exemple un peu plus complexe.

Exemple 9. Soit 0 = {d}, 1 = {f }, 2 = {g} et C un ensemble infini de constantes anonymes. Considérons la transduction sur (,C) où (t, t) si t ne contient pas f et t est obtenu à partir de t en remplaçant tout sous-terme maximal qui ne contient pas de constantes anonymes par un terme de la forme g(f(f(. . . f(c) . . .)), f(f(. . . f(c) . . .))) pour une nouvelle constante anonyme c, où les arguments de g peuvent être de n'importe quelle profondeur et n'ont pas besoin d'être égaux, seule la constante anonyme c doit être la même dans les deux arguments de g. Les sous-termes de t qui contiennent une constante anonyme sont simplement copiés. Par exemple, une transduction possible de g(g(d, d), g(g(c , c ,), c)) pour une constante anonyme c est g(g(f(f(f(c))), f(c)), g(g(c , c), c)) où c est une nouvelleconstante anonyme, en particulier c = c . Nous montrons que est réalisable par TTAC.

Machine Translated by Google

Soit A le semi WTAAC avec les états qC, qR, qM , qf où qC est l'état par défaut et les transitions sont :

$$\begin{aligned} d &\to qR,\\ g(qR, qR) &\to qR,\\ qC &\to qM ,\\ g(qR, qM) &\to qM ,\\ g(qM , qR) &\to qM ,\\ g(qM , qM) &\to qM ,\\ d &\to qf ,\\ qC &\to qf ,\\ g(qf , qf) &\to qf . \end{aligned}$$

Alors qf [t]A ssi f n'apparaît pas dans t ; qR [t]A ssi t ne contient pas f ni constantes anonymes ; et qM [t]A ssi t ne contient pas f mais une constante anonyme.

Il est maintenant facile de construire le TTAC souhaité. Nous choisissons sI comme état initial et utilisons le suivant transitions:

$$sI(, x0) \to qf\ s0(, x0), \tag{8}$$

$$s0(, x0) \to qC\ x0, \tag{9}$$

$$s0(, g(x0, x1)) \to qM\ g(s0(, x0), s0(, x1)), \tag{10}$$

$$s0(, x0) \to qR\ g(sf(vN , x0), sf(vN , x0)), sf \tag{11}$$

$$(vR, x0) \to f(sf(vR, x0)), sf(vR, \tag{12}$$

$$x0) \to vR. \tag{13}$$

La transition (8) est utilisée pour vérifier si le terme d'entrée ne contient pas f . La transition (9) est appliquée si l'entrée term est une constante anonyme. Cette constante est simplement copiée dans le terme de sortie. La transition (10) est appliquée si le terme d'entrée commence par g mais contient une constante anonyme. Par conséquent, ce terme n'est pas remplacé à ce stade et le calcul se poursuit avec les arguments de g. La transition (11) est appliquée si le terme d'entrée courant ne ne contient pas de constante anonyme et, par conséquent, puisque le TTAC traite les termes de haut en bas, l'entrée actuelle terme est un sous-terme maximal du terme d'entrée d'origine sans constantes anonymes et, par conséquent, est supposé être remplacé par g(f(··· f(c)···), f(··· f(c)···)) pour une nouvelle constante anonyme c, qui dans (11) est créée en utilisant vN . Les transitions (12) et (13) sont utilisées pour produire des termes de la forme f(··· f(c)···) où c est l'anonyme constante qui a été générée lorsque la transition (11) a été appliquée.

Une classe C de transductions est fermée par composition si pour toutes les transductions et nous avons que C , implique ◦ C. Il est bien connu [16] que l'ensemble des transductions réalisées par un arbre descendant non déterministe transducteurs sur des signatures finies n'est pas fermé sous composition. Il est facile de voir que cela est également vrai pour TTAC. Un exemple illustrant ce fait est le suivant : évidemment, pour g 1, on peut définir un TTAC T1 que sur l'entrée a 0 génère gn(a) = g(g(··· g(a))) pour tout n 0. De plus, il est facile de construire un TTAC T2 que lorsqu'on lui donne un terme t en entrée, il produit f(t, t). Maintenant, sur l'entrée a, la transduction produit l'arbre $T_2 \circ T_1$ langue {f(gn(a), gn(a)) | n 0}. Par des arguments de pompage standard, il est facile de prouver qu'aucun TTAC unique ne peut produire une telle sortie sur l'entrée a.

Lemme 10. L'ensemble des transductions TTAC-réalisables n'est pas fermé par composition.

3. Le problème du mot pré-image itéré

L'objectif de cette section est de prouver que le problème du mot pré-image itéré est décidable. Ce problème est défini comme suit :

Machine Translated by Google

IteratedPreImage. Étant donné un terme t sur (,C), un TAAC B sur (,C) et une séquence de TTAC T0, ... ,Tl−1 sur (,C) avec = T0 ◦ ··· ◦ Tl−1, décider si t $^{-1}$ (T(B)).

La clé pour prouver la décidabilité de ce problème est :

Théorème 11. L'image préalable d'un langage arborescent reconnaissable par TAAC sous une transduction réalisable par TTAC est une Langage arborescent reconnaissable par TAAC. De plus, un TAAC approprié peut être construit efficacement.

En utilisant ce théorème, nous obtenons immédiatement :

Corollaire 12. IteratedPreImage est décidable.

Preuve. Étant donné t, B, T0, ... ,Tl−1, et comme ci-dessus, par le théorème 11 il s'ensuit qu'un TAAC A reconnaissant (T(B)) $^{-1}$ peut être construit efficacement. Puisque, d'après le lemme 7, le problème verbal pour les TAAC est décidable, nous pouvons décider si t T(A), et donc, si t $^{-1}$ (T(B)).

Le temps d'exécution de la procédure de décision décrite dans la preuve du corollaire peut être non élémentaire puisque, comme nous le verrons, le nombre d'états du TAAC construit dans la preuve du théorème 11 peut être exponentiel dans la taille de l'entrée.

La preuve du théorème 11 est réalisée en trois étapes. Nous montrons d'abord comment les TTAC peuvent être transformés en ce que nous appelons des TTAC simples (section 3.1). Nous construisons ensuite un TAAC reconnaissant la pré-image d'un TAAC reconnaissable langage arborescent sous un TTAC simple (Section 3.2) et enfin prouver l'exactitude de cette construction (Section 3.3).

3.1. TTAC simples

On dit qu'une transition de la forme (3) est simple si

(1) le terme t est soit
   (a) une variable — dans ce cas, nous appelons la transition ε -transition — ou
   (b) de la forme f(x0, ... , xn−1) pour un certain f n — dans ce cas nous appelons la transition -transition —, et
(2) pour tout i<r, le terme t est soit une variable, soit un terme d'état de la forme s(z , x) où x est une variable survenant dans t.

Nous appelons un TTAC simple s'il ne contient que des transitions simples. Le principal avantage des TTAC simples est que le le côté gauche des transitions des TTAC simples est plat et, par conséquent, leur application est plus locale et leurs applications des différentes transitions ne se chevauchent pas trop. Cela rend ces transitions plus faciles à gérer et donc aide à la preuve du théorème 11.

Avant de prouver que chaque TTAC peut être transformé en un TTAC simple équivalent, nous observons :

Lemme 13. Pour tout terme linéaire t T qt ^{n} C, il existe un WTAAC At sur (,C) et un état, disons qt, dans At tel que est le seul état final de At et T(At) = {t | il existe une substitution telle que (t) = t }.

Preuve. La preuve peut être réalisée par une simple induction structurelle sur t.

Notez que le lemme n'est pas valable pour les termes non linéaires.

Nous montrons ensuite :

Lemme 14. Tout TTAC est équivalent à un TTAC simple, qui peut être construit en temps polynomial.

Preuve. Soit T un TTAC comme dans (2) et soit contenant une transition non simple de la forme

$$s(z, t) \to_q t[vR, vN, t_{0,\dots}, t_{r-1, xi0}, \cdots, xil-1] \tag{14}$$

où le t sont des termes d'état et les xij sont des variables apparaissant dans t.

Nous expliquons comment (14) peut être remplacé par plusieurs transitions simples ; en itérant cet argument toutes les transitions non simples les transitions de T peuvent être remplacées par des transitions simples.

Machine Translated by Google

L'idée est la suivante : nous utilisons d'abord une transition ε pour vérifier le q anticipé et rien d'autre. Ensuite, nous étendons A par At (lemme 13) pour pouvoir vérifier si le terme d'entrée correspond à t. Cela se fait dans une transition ε avec qt comme look-ahead. Ensuite, nous nous occupons du problème le plus difficile, à savoir éliminer les occurrences de sous-arbres de t sur le côté droit. Nous codons simplement les positions de ces sous-arbres dans des états appropriés. Enfin, nous ajoutons des transitions pour les états introduits à l'étape précédente, qui garantissent que les sous-arbres des arbres qu'ils Les encodages sont traités de manière appropriée.

Dans la première étape, nous ajoutons simplement la transition ε

$$s(z, x) \to_q s(z, x), \tag{15}$$

où s est un nouvel état. Cette transition ne change pas l'entrée, mais elle ne peut être prise que si l'entrée satisfait la condition d'anticipation.

Dans la deuxième étape, nous vérifions que le terme d'entrée correspond à t. Par conséquent, nous ajoutons les états et les transitions de At , en supposant des ensembles d'états disjoints. De plus, nous ajoutons la transition ε

$$s(z, x) \to_{qt} s(z, x), \tag{16}$$

où s est un autre nouvel état et qt est comme dans le lemme 13. Cette transition ne change pas non plus l'entrée, mais elle ne peut être pris que si l'entrée correspond à t.

Dans la troisième étape, nous codons les références aux sous-termes de t dans de nouveaux états. Pour cela, nous ajoutons des états p^q de la forme p et p pour P(t) et q S. Nous ajouterons ensuite des transitions de telle sorte que ces états satisfassent les conditions suivantes propriétés:

(1) $p(z, t) \in_U$ t ssi t | = t , et, de même,
(2) $p^q(z, t) \in_{Tu}$ t ssi q(z, t |) $\in_{Tu}$ t ,

pour tout choix de termes t et t , tout choix de U C, et tout choix de z. Étant donné ces propriétés, il est maintenant facile d'ajouter la bonne transition. Nous supposons que chaque t_{je} est de la forme si(zi, t_{je}) et pour chaque i nous choisissons une position $_{je}$ tel que t_{je} = t|i . (S'il y a plusieurs positions qui satisfont cette condition, n'importe laquelle sera bonne.) Nous avons également choisir pour chaque position ja $_j$ tel que xij = t| $_j$. Nous pouvons maintenant ajouter la transition ε suivante :

$$s(z, x) \to_t [vR, vN , ps_0(z_0, x), \dots , ps_{r-1}(z_{r-1}, x), p_{0}(, x), \dots , p_{l-1}(, x)].$$

Dans la quatrième étape, nous ajoutons les transitions nécessaires pour garantir (1) et (2) ci-dessus. Tout d'abord, pour chaque f n, q Q, i<n, z {vR, }, et avec i P(t) nous ajoutons

$$_i^q(z, f(x_0, \dots , x_{n-1})) \to_{pp} {}^q(z, x_i),$$

où, par convention, p_ε^q = q. De même, pour chaque f n, i<n, et avec i P(t) on ajoute

$$pi(, f(x_0, \dots , x_{n-1})) \to p(, x_i),$$

où, par convention, p(, xi) est remplacé par xi dans le cas = ε .

On peut maintenant facilement vérifier que le nouveau TTAC est équivalent à l'original, par induction sur la longueur d'un calcul.

3.2. Construction du TAAC reconnaissant la pré-image

Étant donné un TAAC I (l'automate image) et un TTAC T, nous construisons un TAAC P (l'automate pré-image) tel que $T(P) = (T(I))$. Dans la section 3.3, nous prouvons ensuite que notre construction est correcte.

Dans ce qui suit, laissez

$$I = (QI , qd , qs , _{je}, FI)$$

soit un TAAC sur (,C) et

$$T = (QT , IT , A, T)$$

Machine Translated by Google

être un TTAC sur (,C) avec

$$A = (QA, qd_{UN}, qd_{UN}, {}_{UN})$$

comme son automate d'anticipation. Comme mentionné, nous voulons construire un TAAC

$$P = (QP, qd_{P}, qs_{P}, {}_{P}, PF)$$

sur (,C) tel que

$$T(P) = {}_{T}^{-1}(T(Je)).$$

En raison du lemme 14, nous pouvons supposer que T est simple. Ainsi, nous pouvons supposer que T est constitué de -transitions de la forme

$$q(z, f(x0, \dots, xn-1)) \to_{qA} t[vR, \dots, vR, vN, \dots, vN, t_{0,\dots}, t_{r-1, xi0}, \dots, xil-1] \tag{17}$$

où t est linéaire, ij {0, ... est un terme d'état {vR, } de la forme qi(zi, xji) avec ji {0, ... n,− 1}, et vR ne peuvent apparaître que sur le côté droit de (17) si z = vR, et les ε -transitions de la forme

$$q(z, x) \to_{qA} t[vR, \dots, vR, vN, \dots, vN, t_{0,\dots}, t_{r-1, x, \dots}, x] \tag{18}$$

où t est linéaire, t est un terme d'état {vR, } de la forme qi(zi, x), et vR ne peut apparaître que sur le côté droit de (18) si z = vR. Notez que supposer que t est linéaire est wlog puisque nous pouvons dupliquer les entrées vR, vN, $t_{i, x}$ et x_{ij}.

En gros, l'idée derrière la construction de P est que dans une série de P sur t TC , P simule les courses de I sur toutes les sorties possibles t de T sur l'entrée t simultanément. Le problème est que les exécutions de I sont nécessaires pour satisfaire une condition globale, à savoir que les états par défaut et de sélection qd et qs de I sont attribués à des constantes dans un manière cohérente - par une substitution autorisée. Pour capturer cette condition globale dans P, nous utilisons que les exécutions de P aussi satisfaire à une telle condition globale. Plus précisément, la substitution autorisée dans une série de P sur t déterminera la substitutions autorisées qui sont considérées dans les exécutions de I sur les arbres t de la manière suivante :

(1) (Cas « Oui ») Si dans une exécution de P sur t l'état qs_P est assigné à une constante anonyme c (apparaissant dans t), alors dans cette exécution, seules les exécutions de I où qs est assigné à la même constante anonyme c et qd est attribué à toutes les autres constantes anonymes, en particulier à toutes les constantes anonymes générées par T.

(2) (Cas « Non ») Si dans une exécution de P sur t l'état qd_P est attribué à chaque constante anonyme apparaissant dans t, puis dans cette exécution ne prendra en compte que les exécutions de I où qd est attribué à chaque constante anonyme apparaissant dans t sauf pour au plus une constante anonyme générée par T, à laquelle qs peut être attribué .

Afin de pouvoir distinguer entre (1) et (2) et de s'assurer que dans (2) au plus un anonyme la constante est attribuée à qs , le TAAC P fait une certaine comptabilité.

Nous fournissons maintenant la définition formelle de P. Dans la section 3.3, nous montrons que P reconnaît en fait la pré-image.

Espace d'état de P. L'espace d'état de P est défini par

$$QP = 2QI \times 2QA \times \{oui, non\} \times 2QT \times \{qd, qs, \} \times QI \times 2QT \times \{qd, \} \times QI$$

où nous utilisons la notation suivante pour accéder aux composants individuels d'un état. Pour un état b = (S, L, , Md , Ms) de P, on définit :

– Iset(b) = S,
– LA(b) = L,
– vu(b) = ,
– D(q,s)(b) = {a | (q,s, a) Md } pour tout q QT et s {qd , qs , }, et

Machine Translated by Google

– S(q,s)(b) = {a | (q,s, a) Ms} pour tout q QT et s {qd , }.

La signification intuitive des différentes composantes d'un état b est la suivante. L'ensemble Iset(b) rassemble les états atteignables par I sur l'arbre d'entrée t vers P. L'ensemble LA(b) rassemble les valeurs du look-ahead de T pour l'arbre d'entrée donné t. La valeur seen(b) distingue (1) et (2) ci-dessus : si P a observé (vu) que nous sommes dans le cas (1), alors seen(b) = oui, et seen(b) = non sinon. Les deux autres composantes sont plus difficiles à décrire.

Dans D(q,s)(b), nous collectons tous les états accessibles dans une exécution de I sur un arbre de sortie t obtenu en exécutant T sur t en commençant par l'état q où le registre est indéfini (s =), contient une constante anonyme à laquelle qd est assignée (s = qd), ou contient une constante anonyme à laquelle qs est assignée (s = qs), où, dans les deux derniers cas, la constante est supposée ne pas appartenir à t. Les exécutions de I sur t sont simulées par rapport à une substitution autorisée qui mappe toutes les constantes générées par T à l'état par défaut qd — le D majuscule dans D(q,s)(b) rappelant cela — et coïncide avec la substitution autorisée utilisée dans l'exécution de P sur toutes les constantes apparaissant dans t.

L'interprétation de S(q,s)(b) est similaire : ici, nous supposons que toutes les constantes de t sont affectées à qd et que dans les séries de I, toutes les substitutions autorisées sont prises en compte, lesquelles mappent toutes les constantes de t à qd et au plus une nouvelle constante à l'état de sélection qs — le S majuscule dans S(q,s)(b) rappelant cela. Le cas où le registre est affecté à qs n'a pas besoin d'être pris en compte.

Pour donner plus d'intuition à ces composantes, regardons un exemple. Supposons que T ne contienne pas de ε -transitions et exactement une transition avec f 2 sur le côté gauche, à savoir la transition q(vR, f(x, y)) →qA g(vR, x, q (vN , y)), où g 3, q, q sont des états de T, et x et y sont des variables. Nous désignons cette transition par (*). De plus, supposons que P vient de lire les termes t0 et t1, ce qui donne respectivement les états b0 et b1, et que P est sur le point de lire f 2, c'est-à-dire que la configuration actuelle de P est f(b0, b1). Nous considérons les composantes D(q,s)(b) et S(q,s)(b) dans b. De toute évidence, nous avons que D(q ,s)(b) = et S(q ,s)(b) = pour tout q = q et tout s {qs , qd , } puisqu'il n'y a pas de transition dans T avec q et f du côté gauche. En d'autres termes, lorsqu'il est appliqué à un terme de tête f dans l'état q , T ne produit pas de sortie. De plus, D(q,s)(b) = et S(q,s)(b) = si qA / LA(b) puisque (*) est la seule transition dans T dont le côté gauche contient f , mais T ne peut pas s'appliquer (*) si l'automate d'anticipation de T n'accepte pas le terme t = f(t0, t1) dans l'état qA. Supposons maintenant que qA LA(b). Nous avons D(q,)(b) = et S(q,)(b) = puisque le côté gauche de la transition (*) contient vR au lieu de '*'. Pour D(q,qd)(b) nous obtenons l'ensemble [g(qd , Iset(b0), D(q,qd)(b1)]I. C'est l'ensemble des états dans lesquels I peut éventuellement se trouver après avoir lu un terme t produit par T sur l'entrée t dans l'état q où dans l'exécution de I sur un certain t toutes les constantes anonymes générées par T sont affectées à qd et la valeur du registre de T au début de la lecture de t est également affectée à qd . De même, nous avons que D(q,qs)(b) = [g(qs , Iset(b0), D(q,qd)(b1)]I. La seule différence ici est que dans l'exécution de I la valeur du registre de au lieu de qd . Le composant S(q,qd)(b) contient également les états dans lesquels I T au début de la lecture de t est affectée à qs , peut éventuellement se trouver après à qs . Par conséquent, nous avons que S(q,qd) avoir lu les sorties t de T où une constante anonyme générée par T peut être affectée (b) = [g(qd , Iset(b0), D(q ,qs)(b1)]I [g(qd , Iset(b0), S(q ,qd)(b1)]I. Notez qu'en faisant la distinction entre les composantes D et S, nous pouvons garder une trace des endroits où nous autorisons l'affectation de constantes anonymes générées par T à qs dans les exécutions de I et des endroits où nous n'autorisons pas de telles affectations. Nous notons enfin que si T avait des ε -transitions, les composantes qui viennent d'être décrites devraient être augmentées puisque T pourrait également appliquer des ε -transitions et donc produire davantage de termes de sortie. Ci-dessous, nous allons donc considérer les ε -fermetures d'états de P.

L'intuition derrière les composantes des états de P est formellement capturée dans le lemme 17.

La fermeture ε . Il nous faut une définition supplémentaire avant de pouvoir continuer.

Étant donnée une transition comme dans (18), nous écrivons $t \xrightarrow{\varepsilon} [q, \cdots q, \ldots q, D(q0,s0)(b), \ldots, q, , D(qr-1,sr-1)(b), Iset(b), \ldots , Iset(b)]$,

$\xrightarrow{\varepsilon,k}$ et, pour k<r, t comme abréviation du terme q,q,b

$t [q, \cdots q, q, \ldots , q, D(q0,s0)(b), \ldots , D(qk-1,sk-1)(b), S(qk,sk)(b), D(qk+1,sk+1)(b), ,$

Machine Translated by Google

··· , D(qr−1,sr−1)(b), Iset(b), ... , Jeset(b)]

où dans les deux cas si = si zi = , si = q si zi = vR, et si = q si zi = vN .

Étant donné une transition comme dans (17) et des états b0, ... , bn−1 (ce seront les états assignés aux arguments de f dans une série de P), nous écrivons $t_{q\,,\,q}$ comme abréviation du terme

t [q , ··· q , q, ... , q, D(q0,s0)(bj0), ... , D(qr−1,sr−1)(bjr−1), Iset(bi0), ... , Iset(bil−1)] ,

et, pour k<r, $t_{,kq\,,q}$ comme abréviation du terme

t [q , ··· q, ... , q, D(q0,s0)(bj0), ... , D(qk−1,sk−1)(bjk−1), S(qk ,sk)(bjk), D(qk+1,sk+1)(bjk+1), ... , q ,
··· , D(qr−1,sr−1)(bjr−1), Iset(bi0), ... , Iset(bil−1)]

où encore dans les deux cas si = si zi = , si = q si zi = vR, et si = q si zi = vN .

Nous pouvons maintenant définir la ε -clôture d'un état de P. Soit b QP . Nous définissons une suite b0, b1, ... par récurrence, où, dans le cas de base, nous posons b0 = b, et pour i 0 nous définissons les composantes de bi+1 comme suit :

– Iset(bi+1) = Iset(b), LA(bi+1) = LA(b), vu(bi+1) = vu(b).
– Pour tout q QT et s {qd , qs , }, l'ensemble D(q,s)(bi+1) est obtenu à partir de D(q,s)(bi) en additionnant tous les états a pour dans laquelle il existe une ε -transition comme dans (18) telle que qA LA(bi), z = ssi s = , et a [t]I. $s,\varepsilon qd\,,bi$
– Pour tout q QT et s {qd , }, l'ensemble S(q,s)(bi+1) est obtenu à partir de S(q,s)(bi) en additionnant tous les états a pour lesquels il existe une ε -transition comme dans (18) telle que qA LA(bi), z = ssi s = , et a $[t_{s,qs,\varepsilon bi}]$I ou a $[t_{s,qd,\varepsilon,k bi}]$I pour un certain k<r.

De toute évidence, il existe un indice j tel que bj = bj+1 = ... Nous définissons l' ε -fermeture de b comme étant bj pour un tel aj et la notons b.

Les états par défaut et de sélection de P. L'état par défaut qd de P est défini comme la ε -fermeture bd de l'état suivant bd :

– Iset(bd) = [qd]I, LA(bd) = [qd A]A, vu(bd) = non, – D(q,s)
(bd) = pour tout q QT et s {qd , qs , }, – S(q,s)(bd)
= pour tout q QT et s {qd , }.

De même, l'état de sélection qs – P de P est défini comme la ε -fermeture bs de l'état suivant bs :

Iset(bs) = [qs]I, LA(bs) = [qd A]A, seen(bs) = yes, – D(q,s)
(bs) = pour chaque q QT et s {qd , qs , }, – S(q,s)
(bs) = pour chaque q QT et s {qd , }.

Transitions de P. Dans ce qui suit, par abus de notation nous écrivons qd au lieu de qd et qs au lieu de qs . De cette façon, P . Dans ce nous pouvons utiliser les mêmes substitutions permises pour P et I. Rappelons que qd et qs sont respectivement l'état par défaut et l'état de sélection de I. l'automate P

Pour tout f bn−1 n et b0, ..., bn−1 contient la transition f(b0, ...
est défini comme étant la ε -fermeture b de l'état suivant b déterminé par les trois conditions suivantes.

– Iset(b) = [f(Iset(b0), ... , Iset(bn−1))]I, LA(b)=[f(LA(b0), ... , LA(bn−1))]A, vu(b) = oui s'il existe i tel que vu(bi) = oui, et vu(b) = non sinon.
– Pour tout q et s, on a a D(q,s)(b) s'il existe une -transition comme dans (17) telle que qA LA(b), z = ssi s = , et a [t s,qd]I pour tout q QT et s {qd , qs , }.
– Pour tout q et s, on a a S(q,s)(b) s'il existe une -transition comme dans (17) telle que qA LA(b), (z = ssi s,qd]I pour un [t s,qs]I ou a [t ,k certain k pour tout q QT et s {qd , }. s =), et a

Machine Translated by Google

États finaux de P. L'ensemble des états finaux FP de P contient un état b s'il existe q IT et a FI tels que – vu(b) = oui et a D(q,)(b), ou – vu(b) = non et a S(q,) (b).

3.3. Rectitude de la construction

La proposition suivante énonce que notre construction est correcte.

Proposition 15. $T(P) = \overset{-1}{T}(T(Je))$.

Pour prouver cette proposition, nous devons prouver deux lemmes. Le premier lemme, qui découle immédiatement de la construction stipule que P est complet et déterministe, et que les états atteignables sont ε -fermés.

Lemme 16. Pour tout t et toute substitution permise ε - , il existe un état b tel que {b}=[(t)]P. Cet état b est fermée, c'est -à-dire $b = \overline{b}$.

Le deuxième lemme est plus complexe et constitue la clé pour prouver la proposition 15. Il formalise toute l'intuition derrière notre construction.

Lemme 17. Pour tout terme t TC où , et la substitution autorisée telle que b [(t)]P ce qui suit est vrai, b QP , on écrit =t pour dire que est une substitution permise qui coïncide avec sur occC(t).

(1) Iset(b) = [(t)]I.
(1) LA(b) = [t]A.
(3) Les éléments suivants sont équivalents : (A) vu(b) = oui.
(B) Il existe c occC(t) tel que (c) = qs .
(4) Pour chaque a QI et q QT , les éléments suivants sont équivalents :
(A) a D(q,)(b).
(B) Il existe t TC et tel que =t , (c) = qd pour tout c occC(t) \ occC(t), a [(t)]I, et q(, t) t .
(5) Pour chaque a QI , q QT et c / occC (t), les éléments suivants sont équivalents :
(A) a D(q,qd)(b).
(B) Il existe t TC et =t st (c) = qd pour tout c occC(t) \ occC(t), (c) = qd , a [(t)]I, et q(c, t) t .
(6) Si vu(b) = non, alors pour tout a QI , q QT , c / occC(t), les éléments suivants sont équivalents :
(A) a D(q,qs)(b).
(B) Il existe t TC et st =t , (c) = qs , a [(t)]I, et q(c, t) t .
Notez que (c) = qd pour tout c occC(t) puisque vu(b) = no.
(7) Si vu(b) = non, alors pour tout a QI , q QT , les éléments suivants sont équivalents :
(A) a S(q,)(b).
(B) Il existe t TC et st =t (8) Si vu(b) = non, , a [(t)]I, et q(, t) t .
alors pour tout a QI , q QT , c / occC(t), les élémentssuivants sont équivalents : (A) a S(q,qd)(b).
(B) Il existe t TC et st =t , (c) = qd , a [(t)]I, et q(c, t) t .

Avant de prouver ce lemme, nous l'utilisons pour prouver la proposition 15.

Preuve (Proposition 15). -1 T(P) T (T(I)) : Supposons que t T(P). Il s'ensuit qu'il existe une substitution permise et un état final b FP tels que b [(t)]P. Considérons deux cas. Supposons d'abord que seen(b) = yes. Alors, il existe q IT et a FI tels que a D(q,)(b). Le lemme 17, (4) implique qu'il existe t et une substitution permise telle que q(, t) t et a [(t)]I, et donc, t T(I) puisque a FI . Cela signifie que $t \overset{-1}{T}(T(I))$.

Machine Translated by Google

Deuxièmement, supposons que vu(b) = no. Alors, il existe q IT et a FI tels que a S(q,)(b). D'après le lemme 17, (7) il existe t et une substitution permise telle que q(, t) t et a [(t)]I. Ainsi, t T(I) et t T^{-1}(T(Je)).

T(P) T^{-1}(T(I)) : supposons que t T^{-1}(T(I)). Cela signifie qu'il existe t , un FI, et une substitution permise telle que q(, t) t pour un q IT et a [(t)]I. Soit b [(t)]P. Un tel ab existe et est unique déterminé en raison du lemme 16. Nous montrons que b , et donc, t T(P). Tout d'abord, supposons que vu(b) = oui. Ensuite, FP par le lemme 17, (3) et comme est une substitution permise, nous avons que (c) = qd pour tout c / occC(t). Par le lemme 17, (4), nous pouvons conclure que a D(q,)(b). Deuxièmement, si seen(b) = no, le lemme 17, (7). implique que a S(q,)(b). Dans dans les deux cas, on obtient b FP .

Preuve (lemme 17). Soit t TC (2), , b QP , et être une substitution autorisée telle que b [(t)]P. Énoncés (1), et (3) sont faciles à vérifier par la construction de P. Nous prouvons (4)–(8) simultanément. Nous montrons d'abord la implications de gauche à droite par induction structurelle sur t puis établir l'autre sens par induction sur la longueur des calculs.

« » : Cas de base. Supposons que t C. Nous savons que b = $\overline{bd}$ ou b = $\overline{bs}$. Soit b0 = bd ou b0 = bs ; les deux cas peuvent être traitées de la même manière. Pour b0 les implications sont trivialement valables. Par induction sur i, nous montrons qu'elles sont valables pour bi+1. Nous nous concentrons sur (8) car c'est l'un des cas les plus intéressants. Les autres inclusions peuvent être montrées de manière analogue. Nous supposons que seen(b) = no et a S(q,qd)(bi+1). Nous devons montrer qu'il existe t et a permis substitution telle que =t et a [(t)]I. Si a (c) = qd, q(c, t) S(q,qd)(bi), cela suit par la hypothèse d'induction. Sinon, par définition de S(q,qd)(bi+1), on sait qu'il existe une ε -transition comme dans (18) tel que qA LA(bi), z = vR et a [$t^{\varepsilon}_{qd\,,\,qs,\,bi}$]Je ou un [$t^{\varepsilon\,,k}_{qd\,,\,qd\,,\,bi}$]I pour un certain k. Supposons d'abord que a [$t^{\varepsilon}_{qd\,,\,qs,\,bi}$]JE. Alors, il existe un j D(qj,sj)(bi) tel que a [t [qd , , qd , qs , , Iset(bi)]]I. Soit ··· , qs , a0, ... , ar−1, Iset(bi), ... c / occC(t) {c} et ci = si zi = , ci = c si zi = vR, et ci = c si zi = vN . On définit (c) = qs et (c) = qd . Notez que =t et (c) = qd . En utilisant l'hypothèse de récurrence sur i, il est facile de vérifier pour tout c = c si il existe t , t_{r-1} tel que qj(cj, t) 0, ... t_j, aj [(t j)]I, et le t j sont choisis de telle manière que nouvelles constantes générées dans le calcul qj(cj, t) les constantes apparaissant t_j sont différents de c, c , en t, et ceux qui sont générés dans qj(cj , t) pour j = j. Notez que pour établir l'existence de la t_j avec les propriétés ci-dessus, nous pouvons en fait utiliser comme substitution autorisée pour chaque j. Ainsi, nous avons q(c, t) r−1 , t, [c, ... k, c c t_j , t] := t et a [(t)]I. Le cas où a [$t^{\varepsilon\,,k}_{qd\,,\,qd\,,\,bi}$]Je pour certains peut être traité de manière analogue. Ceci conclut la preuve du cas de base.

Ce que nous avons essentiellement montré ici, c'est que si les implications sont valables pour un État, elles sont valables pour l'autre. ε -fermeture de cet état. Le fait que t soit une constante anonyme n'a été utilisé que pour montrer l'implication pour b0.

Étape d'induction. Supposons que t = f(t0, ... , tn−1) et que les inclusions sont vraies pour les sous-termes ti de t. Soit bi l'unique élément avec bi [(ti)]P. De la même manière que dans le cas de base, on peut montrer que les inclusions pour b où b est défini comme dans la définition des transitions de P. De plus, à partir de la preuve de la base cas, il s'ensuit que les inclusions restent vraies lorsque l'on prend la fermeture ε d'un état. Ainsi, elles restent vraies pour b = $\overline{b}$.

« » : Nous prouvons (4)–(8) simultanément par induction sur la longueur des calculs. Pour les calculs de longueur zéro rien n'est à montrer puisque les conditions de (B) sont nulles. Dans l'étape d'induction, nous nous concentrons à nouveau sur (8) ; les autres cas peuvent être montrés de manière analogue. Supposons que seen(b) = no et soit c / occC(t). Pour tout t TC t nous a QI , et substitution autorisée telle que =t , (c) = qd , a [(t)]I, et q(c, t) doivent montrer que a S(q,qd)(b). Nous distinguons deux cas selon que la première transition T s'applique dans q(c, t) t est une transition - ou ε .

T est une -transition. Supposons que T est une -transition de la forme (17) où z = vR. On utilise c C \ (occC(t) {c}) comme la nouvelle constante générée par T (dans le cas où T est génératif). Nous avons que t = f(t0, ... , tn−1) pour un certain ti TC . Soit bi l'élément déterminé de manière unique dans [(ti)]P = [(ti)]P. Alors, nous savons que b est b où b est défini comme dans la définition des transitions. Après avoir appliqué T à q(c, t) on obtient

Machine Translated by Google

t [c, ... , c, c , ··· , c , q0(c0, tj0), ... , qr−1(cr−1, tjr−1), ti0 , ··· , jusqu'à−1]

où ci = si zi = , ci = c si zi = vR, et ci = c si zi = vN . Soit t 0, ... et , t r−1 soient les termes tels que qi(ci, tji) t je

t = t [c, ... , c, c , ··· , c , t 0, ... , t r−1, ti0 , ··· , jusqu'à−1].

Puisque a [(t)]I, il existe ai QI tel que ai [(t i)]I et

a [(t [c, ... , c, c , ··· , c , a0, ... , ar−1, [(ti0)]Je, ... , [(jusqu'à−1)]I]]I.

D'après le lemme 17, (1) et puisque et coïncident sur occC(t), on a Iset(bij) = [(tij)]I. Le lemme 17, (2) assure que qA LA(b) = LA(b) = [t]A. On distingue deux cas.

Tout d'abord, supposons que (c) = et ainsi, toutes les autres constantes anonymes sont mappées sur qd . Il est facile de vérifier. qs , qui satisfait les conditions pour D(qi,si)(bji) par rapport à tji . Ainsi, l'hypothèse de récurrence sur la longueur des calculs donne ai D(qi,si)(bji). Maintenant, il s'ensuit que a [t qd ,qs]I, et donc, a S(q,qd)(b) S(q,qd)(b) = S(q,qd)(b).

Deuxièmement, supposons que soit une substitution permise telle que (c) = qd pour tout c occC(t) {c, c }. Nous) tel que (c) = considérons deux sous-cas. Premièrement, supposons qu'il existe k et c occC(t k que c / occC(t) {c, c }, et donc, c a été nouvellement généré dans qk (ck , tjk) cur in t i qs . Il s'ensuit que k . Par conséquent, c t n'est pas oc-pour i = k. Il est à nouveau facile de vérifier que satisfait les conditions pour D(qi,si)(bji) par rapport à . tji pour tout i = k et S(qk ,sk)(bjk) par rapport à . tjk . Maintenant, l'hypothèse de récurrence sur la longueur des calculs donne que ai D(qi,si)(bji) pour tout i = k et ak S(qk ,sk)(bjk). Ainsi, a [t qd ,qs]I. Par conséquent, ,k

a S(q,qd)(b) S(q,qd)(b) = S(q,qd)(b). S'il n'y a pas de k et que c occC(t, k) tel que (c) = qs , alors on peut montrer de la même manière que a [t qd ,qs]I est pair pour tout k, et donc a S(q,qd)(b). ,k

T est une ε -transition. Ce cas peut être traité de manière très similaire au cas des ε -transitions. Au lieu d'utiliser la définition des transitions de P, on utilise la définition de la ε -fermeture et le fait que b est ε -fermé par le lemme 16.

4. Le modèle de protocole basé sur un transducteur d'arbre

Dans cette section, nous présentons notre modèle de protocole et d'intrus. Les hypothèses de base de notre modèle coïncident avec celles des modèles décidables de protocoles non-bouclables : tout d'abord, nous analysons les protocoles par rapport à un nombre fini d'actions de réception-envoi, et en particulier, un nombre fini de sessions. Ensuite, l'intrus est basé sur l'intrus Dolev-Yao. Il peut dériver de nouveaux messages à partir de messages connus par décomposition, décryptage, composition, cryptage et hachage. Nous ne mettons pas de limite à la taille des messages. Comme dans [2], nous supposons que les clés sont des messages atomiques ; dans [34,26,5], il peut s'agir de messages complexes.

La principale différence entre le modèle présenté ici et les modèles pour les protocoles non-bouclés réside dans la manière dont les actions de réception-envoi sont décrites : au lieu de règles de réécriture simples, nous utilisons des TTAC. Ces transducteurs ont deux caractéristiques importantes nécessaires pour modéliser des actions de réception-envoi récursives, mais manquantes dans les modèles décidables pour les protocoles non-bouclés : tout d'abord, ils permettent d'appliquer un ensemble de règles de réécriture de manière récursive à un terme. Ensuite, ils permettent de générer de nouvelles constantes, une caractéristique qui n'est pas nécessaire pour les protocoles non-bouclés lorsqu'ils sont analysés par rapport à un nombre fini d'actions de réception-envoi.

Nous fournissons maintenant la définition formelle de notre modèle basé sur un transducteur d'arbre en définissant des messages, l'intrus, protocoles et attaques.

4.1. Messages

La définition des messages que nous utilisons ici est plutôt standard, sauf que nous autorisons un nombre infini de messages (anonymes). ymous). Comme mentionné, nous supposons que les clés sont atomiques.

Machine Translated by Google

Plus précisément, les messages sont définis comme des termes sur la signature (A, C) avec des constantes anonymes. L'ensemble C est un ensemble dénombrable infini de constantes anonymes, qui dans cet article sera utilisé pour modéliser les clés de session (Section 7.1). La signature finie A est définie relativement à un ensemble fini A de constantes, l'ensemble des messages atomiques, qui peut par exemple contenir des noms principaux et des clés (à long terme). Il contient également un sous-ensemble K de clés publiques et privées qui est équipé d'une application bijective · attribuant à une clé publique (privée) $\overline{k}^{1}$ K sa clé privée (publique) correspondante k−1 K. Maintenant, A désigne la signature (finie) composée des constantes de A, des symboles unaires hasha (hachage à clé) et encs (chiffrement symétrique) pour tout a A, du symbole unaire enca (chiffrement asymétrique) pour tout k K, et du symbole binaire (appariement). Au lieu de $(t, t)_k$ nous écrivons $\{t, t\}_k$. Le terme hasha(m) doit représenter le hachage de m sous la clé a plus m lui-même. On pourrait rendre cela explicite en écrivant m, hash(a, m) au lieu de hasha(m). Cependant, vérifier si un message est de la forme m, hash(a, m) pour un message m nécessite le terme non linéaire x, hash(a, x), où x est une variable. Pour éviter de tels termes non linéaires, nous utilisons hasha(m), mais permettons à l'intrus (voir ci-dessous) de dériver m de hasha(m). L'ensemble des messages sur (A, C) est noté M = TA (C).

Notez que les constantes anonymes ne sont pas autorisées comme clés. La question de savoir si le problème de décision Attack (voir Section 4.4) serait toujours décidable autrement reste un problème ouvert .

4.2. L'intrus

Comme dans le cas des modèles pour les protocoles non-bouclables, notre modèle d'intrus est basé sur l'intrus de Dolev-Yao [14]. Autrement dit, un intrus a un contrôle total sur le réseau et peut dériver de nouveaux messages à partir de ses connaissances actuelles en composant, décomposant, cryptant, décryptant et hachant des messages. Nous n'imposons aucune restriction sur la taille des messages.

L'ensemble (éventuellement infini) de messages d(S) que l'intrus peut dériver d'un ensemble S M est le plus petit ensemble satisfaisant aux conditions suivantes :

(1) S d(S); (2) si m, m d(S), alors m, m d(S) (décomposition); (3) si encs (m) d(S) et a d(S), alors m d(S) (déchiffrement symétrique); (m) d(S) et k−1 d(S), alors $\{m\}_k$ d(S) (déchiffrement asymétrique); (4) si enca (5) si hasha(m) d(S), alors m (obtention de messages hachés); (6) si m, m d(S), alors m, m d(S) (composition); (7) si m d(S) et a A ∩ d(S), alors encs (m) d(S) (chiffrement symétrique) ; (8) si m d(S) et k K ∩ d(S), alors enca (m) d(S) (chiffrement asymétrique) ; (9) si m d(S) et a A ∩ d(S), alors $\{\text{hasha}(m)\}_k$ d(S) (hachage à clé).

Soit an(S) la fermeture de S sous (2)–(5), et syn(S) la fermeture de S sous (6)–(9)

Il est bien connu que d(S) peut être obtenu en appliquant d'abord an à S et au résultat en appliquant syn. C'est parce que nous utilisons des clés atomiques ; pour les clés complexes, cela n'est pas vrai (voir, par exemple, [32,28]) :

Lemme 18. Pour tout S M,

$$d(S) = \text{syn}(\text{an}(S)).$$

Nous notons que bien que les principaux aient la capacité de générer de nouvelles constantes (anonymes), telles qu'elles sont définies en termes de TTAC, pour l'intrus, l'ajout de cette capacité n'est pas nécessaire car cela n'augmenterait pas son pouvoir d'attaquer les protocoles (voir également la section 4.4).

4.3. Protocoles

Les protocoles sont décrits par des ensembles de principes et chaque principe est défini par une séquence d'actions de réception-envoi, qui dans une exécution de protocole sont exécutées l'une après l'autre. Chaque action de réception-envoi est spécifiée par un certain TTAC, que nous appelons transducteur de message.

Machine Translated by Google

Définition 19. Un transducteur de message T est un TTAC sur (A, C).

En gros, un principal est défini comme une séquence de transducteurs de messages.

Définition 20. Un principal (basé sur TTAC) est un tuple

((T0, ... ,Tn−1), I)

constitué d'une séquence (T0, ... ,Tn−1) de transducteurs de messages et d'une relation n-aire I I0 ×···× In−1 où Ii désigne l'ensemble des états initiaux de Ti .

Les transducteurs de message unique Ti dans la définition de sont appelés actions de réception-envoi. Dans une exécution de protocole, exécute les actions de réception-envoi l'une après l'autre. Plus précisément, au début d'une exécution de protocole, un tuple (q0, ... , qn−1) I est choisi de manière non déterministe où qi sera l'état initial de Ti dans l'exécution en cours. Maintenant, si dans le protocole exécuté le premier message reçu est m0, alors renvoie un message m 0 avec (m0, m 0) T0(q0). Ensuite, à la réception du deuxième message, disons m1, renvoie m 1 avec (m1, m 1) T1(q1), et ainsi de suite. En fixant la valeur initiale indique au début, que nous modélisons que nous pouvons transmettre (une quantité finie d') informations à partir d'un récepteur-émetteur action à une autre. Par exemple, si q0 code que l'on s'attend à parler à Bob, alors q1 pourrait décrire cela dans le le deuxième message s'attend à revoir le nom de Bob.

Un protocole est défini comme une famille finie de principaux plus, puisque nous nous intéressons aux attaques sur les protocoles, une connaissance initiale de l'intrus et des informations sur les actions de réception/envoi qui doivent être considérées comme telles. être des actions de « défi » :

Définition 21. Un protocole P (basé sur TTAC) est un tuple ({i}i<n, S,C) où

–{i}i<n est une famille de n principaux (basés sur TTAC), et
–S M est un ensemble fini, la connaissance initiale de l'intrus,
–C {0, ... , n − 1} est l'ensemble des indices de défi.

La dernière action du principal i sera appelée une action de sortie de défi si i C. (L'utilisation d' actions de sortie de défi sera expliqué dans la section suivante.)

La classe de protocoles qui peuvent être spécifiés selon la définition 21 peut être grossièrement caractérisée comme suit : dans chaque action de réception-envoi, les principaux peuvent effectuer des calculs récursifs, qui, puisqu'ils sont décrits comme TTAC, permet d' effectuer de manière récursive des vérifications linéaires sur les messages d'entrée (c'est-à-dire des vérifications qui peuvent être exprimées en termes de correspondance avec des termes linéaires). Par exemple, dans une action de réception-envoi, un principal peut parcourir une liste des requêtes et vérifie le format de chaque requête en faisant correspondre un terme linéaire. Dans le processus récursif les principaux peuvent également produire un nombre illimité de nouveaux nonces (constantes anonymes) et créer des messages de sortie, par exemple, une liste de certificats contenant des clés de session (constantes anonymes fraîchement générées) ou un message contenant des chiffrements profondément imbriqués. De plus, comme expliqué ci-dessus, les principaux peuvent transmettre un quantité d'informations d'une action de réception-envoi à la suivante. Dans la section 7, nous illustrons le type de protocoles qui peut être modélisé par des exemples.

4.4. Attaques

Lors d'une attaque sur un protocole, l'intrus, qui a le contrôle total du réseau de communication, entrelace les actions de réception-envoi des principaux d'une manière ou d'une autre (c'est-à-dire qu'il détermine un ordre total sur la réception-envoi). actions), et essaie de produire des entrées pour les mandants de telle sorte qu'à partir des sorties correspondantes et de son initial de cette connaissance, il peut tirer un secret, c'est-à-dire un message qui n'est pas censé tomber entre les mains de l'intrus. Un tel secret peut par exemple être une clé de session ou un message secret. Ainsi, on peut vérifier si un protocole préserve le secret. On peut également vérifier un type d' authentification faible. Par exemple, le secret peut être quelque message auxiliaire indiquant qu'un principal, disons P, a terminé une session avec une instance d'un autre principal, disons P, qui n'existe pas dans le modèle de protocole spécifié. Maintenant, si l'intrus parvient à voir le message secret, cela signifie que la propriété d'authentification est violée. Des formes d'authentification plus fortes pourraient également être vérifiées

Machine Translated by Google

si l'absence de certaines actions de réception-envoi était testée. Cependant, l'authentification n'est pas l'objet principal du présent travail et nous n'étudierons donc pas plus en détail l'authentification ici.

Dans la définition des attaques, nous utilisons des actions de sortie de défi (voir définition 21). Dans l'entrelacement des actions de réception–envoi déterminées par l'intrus, nous exigeons que la dernière action de réception–envoi (et seulement cette action) soit une action de sortie de défi. Cette action détermine le secret que l'intrus essaie de déduire. C'est-à-dire que la sortie de cette action n'est pas ajoutée aux connaissances de l'intrus mais lui est présentée comme un défi, c'est-à-dire un message à déduire.

L'utilisation d'actions de sortie de défi permet de déterminer les secrets de manière dynamique, en fonction du protocole exécuté. Ceci est par exemple nécessaire lorsqu'on demande si l'intrus est capable de dériver une clé de session (une constante anonyme, qui peut changer d'un protocole exécuté à un autre) générée par un serveur de distribution de clés. Alternativement et de manière équivalente (pour se passer des actions de sortie de défi), on pourrait se demander si l'intrus peut dériver un message atomique fixé a priori, disons secret, qui est chiffré par une constante anonyme (la clé de session) : le secret chiffré peut être dérivé par l'intrus ssi l'intrus connaît la constante anonyme utilisée pour chiffrer le secret. Cependant, comme en général nous n'autorisons pas les constantes anonymes comme clés (voir les sections 4.1 et 8), nous trouvons l'utilisation d'actions de sortie de défi plus élégante que l'introduction de types spéciaux de messages avec des constantes anonymes comme clés. De plus, les actions de sortie de défi sont quelque peu liées à la façon dont la sécurité est définie dans les modèles de calcul pour les protocoles de distribution de clés où, à la fin d'une attaque, l'intrus se voit présenter une chaîne pour laquelle il doit décider s'il s'agit d'une clé de session réelle ou simplement d'une chaîne aléatoire [4].

Nous remarquons que la manière dont les attaques sont définies ici permet de se demander si l'intrus peut dériver un message appartenant à un langage d'arbre régulier prédéfini . Dans la plupart des modèles de protocoles non-bouclés, cela n'est pas possible (voir cependant [37,18,27]).

Définition 22. Soit P = ({i}i<n, S,C) un protocole avec i<n. = ((T 0, ... , T ni−1), li) et li li ×···× li pour ni−1 0

Une attaque sur P est un tuple

(O, <,)

composé de

– un ensemble non vide O {(i,j) | i<n,j<ni} d'indices d'actions de réception–envoi déclenchées lors d'une attaque, – un ordre total < sur O, l'entrelacement des actions de réception–envoi, et – une application affectant à chaque (i,j) O un tuple (i,j) = (qi mi j, j, moi j)

et satisfaisant aux conditions suivantes :

(1) Pour tout (i,j) O, si (i,j) O, alors (i,j) O et (i,j) < (i,j) pour tout j < j.

(2) Soit (i,j) le plus grand élément de O par rapport à <. Alors, pour chaque (i ,j) O, l'action T j est un défi action de sortie ssi (i,j) = (i ,j).

(3) Soit i<n et j maximal avec (i,j) O. Alors il existe qi (4) Pour j+1 , ··· , qi hi−1 tel que (qi 0, ... , qi hi−1) li, chaque (i,j) O il est vrai que mi j, moi j M et (mi j, moi j) Ti j(qi j) .

(5) Pour chaque (i,j) O, il s'avère que (occC(mi j) \ occC(mi j)) ∩ occC(Si j) = où Si = S j {mi (6) Pour j | (i , j) < (i, j)}. chaque (i,j) O, il s'avère que mi d(Si j).

Une attaque est dite réussie si la dernière action de réception–envoi, l'action de sortie de défi, disons avec l'index (i,j) O, renvoie un c tel que c d(Si j) ∩ (A C).

Notez que (5) garantit que les nouvelles constantes anonymes générées dans une action de réception–envoi sont également nouvelles par rapport à la connaissance de l'intrus avant que cette action ne soit effectuée. Notez également que dans (6) l'ensemble d(Si j) est la connaissance de l'intrus avant d'effectuer l'action de réception–envoi à l'étape (i, j).

Le problème de décision qui nous intéresse est :

Attaque. Étant donné un protocole P, décider s'il existe une attaque réussie sur P .

S'il n'y a pas d'attaque réussie sur un protocole, on dit que le protocole est sécurisé.

Machine Translated by Google

Comme mentionné ci-dessus, étendre l'intrus en lui permettant de générer de nouvelles constantes n'augmente pas sa capacité à attaquer les protocoles. La remarque suivante précise cela.

Remarque 23. Si la connaissance initiale de l'intrus contient au moins une constante anonyme, alors il existe une attaque sur un protocole P ssi il existe une attaque sur P dans laquelle l'intrus peut générer de nouvelles constantes anonymes.

Preuve. Formellement, un intrus qui peut générer des constantes anonymes est un intrus dont la connaissance initiale contient un nombre infini de constantes anonymes (en plus de sa connaissance initiale ordinaire). Nous soutenons qu'une constante anonyme, disons cl , c'est suffisant.

La raison est que les TTAC ne peuvent pas vérifier les constantes anonymes pour la disparité. Si (m, m) appartient à la transduction d'un TTAC, alors ((m), (m)) fait également partie de la transduction d'un TTAC, où mappe les constantes anonymes de m à une constante arbitraire c / occC(m) \ occC(m), c'est-à-dire une constante qui n'est pas nouvellement générée. Cela signifie, en particulier, que sj (mi, mi j) fait partie d'une attaque, alors ((mi j), (mi j)) fait partie d'une attaque où les constantes anonymes sont mappées à partir de la connaissance initiale de l'intrus dans à mi . Observez que si mi jj peut être dérivé par l'intrus, alors (mi j) aussi .

Étant donné que dans les modèles pour les protocoles non-bouclés, les tests d'inégalité entre les messages ne sont généralement pas possibles (une exception est [15]), dans ces modèles, étendre l'intrus avec la capacité de générer de nouvelles constantes n'augmenterait pas non plus son pouvoir d'attaque des protocoles.

5. Le résultat de décidabilité

Le résultat principal de cette section est le suivant :

Théorème 24. Pour les protocoles basés sur TTAC, l'attaque est décidable.

Pour démontrer ce théorème, il suffit évidemment de montrer que le problème suivant est décidable.

Atteindre entrelacement. Étant donné un ensemble fini S M (la connaissance initiale de l'intrus), une séquence T0, ... ,Tl−1 de transducteurs de messages (l'entrelacement des actions de réception-envoi) avec Ti = (Qi, Ii, Ai , i) pour i<l, décider s'il existe des messages mi, m (1) (mi, M, i<l, tel que
m (2)) Ti pour chaque i<l,
(occC(m (3)) \ occC(mi)) ∩ occC(Si) = pour tout i<l,
mi d(Si) pour tout i<l, et (4) m
d(Sl−1) ∩ (A C), où Si = S {m
0, ... , m i−1 } est la connaissance de l'intrus avant que la ième action de réception–envoi ne soit effectuée.

Nous écrivons (S,T0, ... ,Tl−1) InterleavingAttack si toutes les conditions ci-dessus sont satisfaites.

La preuve de la décidabilité d'InterleavingAttack se déroule en deux étapes. Tout d'abord, nous montrons que l'intrus peut être simulé par un TTAC (voir Section 5.1). Ensuite, nous réduisons le problème InterleavingAttack au problème IteratedPreImage (Section 5.2), dont nous savons qu'il est décidable.

5.1. Dériver est réalisable en TTAC

Nous souhaitons montrer que les messages dans d({m})pour un message m peuvent être produits par un TTAC. Plus précisément, nous allons construire un TTAC Tder tel que Tder (m) = d({m}) pour chaque message m.

Nous définissons d'abord ce que nous appelons l'automate de découverte de clés qui est utilisé comme anticipation dans Tder.

5.1.1. Découverte de clés

L'automate de découverte de clés D est un WTAAC complet et déterministe contenant toutes les informations sur les clés auxquelles on peut accéder dans un message donné. Plus précisément, un état de l'automate de découverte de clés est une fonction 2A →2A et l'automate est configuré de telle manière que l'état [m]D dans lequel se trouve l'automate après la lecture du message

Machine Translated by Google

message m — notez que puisque D est complet et déterministe, [m]D = { } pour une fonction — a la propriété suivante : [m]D(K) est l'ensemble des atomes qui peuvent être dérivés de K et m, c'est-à-dire, [m]D(K) = an({m} K) ∩ A pour tout K A et message m. Dans ce qui suit, il est soutenu que cela est effectivement possible, c'est-à-dire que nous construirons D avec la propriété souhaitée. L'ensemble de toutes les fonctions 2A →2A, c'est-à-dire l'ensemble des états de D, est noté QD. Nous notons que la cardinalité de QD est doublement exponentielle dans la cardinalité de A.

L'état par défaut de D est l'application d'identité. Les transitions de D sont définies comme suit. Pour tout a A, D contient une transition

$$a \to (K \to K \quad \{a\})$$

pour chaque K A.

Pour chaque a A, k K et d QD, le WTAAC D contient des transitions

$$\begin{aligned} &\text{encs}_{un}(d) \to d, \\ &\text{enca}_{(ré)} \to ré, k\ \text{hasha}(ré) \\ &\to ré, \end{aligned}$$

où d est déterminé par le tableau suivant :

condition du côté gauche sur K	d(K)
$\text{encs}_{un}(d)$	a K d (K)
$\text{encs}_{un}(d)$	a / KK k−1 K d
$\text{enca}_{k}(d)$	(K) k−1 / KK aucune
$\text{enca}_{(d)_k}$	condition K.
$\text{hasha}(d)$	

Enfin, pour chaque d , d QD, le WTAAC D contient une transition

$$d, d \to d \tag{19}$$

où, pour tout K A, d(K) est le plus petit ensemble tel que

– K d(K), –
si K d(K), alors d (K) d(K), – si K d(K), alors d(K) d(K).

La définition de (19) est plus complexe que celle des autres transitions puisque dans un message de la forme m, m clés dans m pourraient être utilisées pour obtenir de nouvelles clés dans m (et vice versa) et ces clés peuvent à leur tour être utilisées pour obtenir de nouvelles clés dans m, qui peuvent à leur tour être utilisées pour obtenir de nouvelles clés dans m et ainsi de suite. Par conséquent, d(K) est défini par un point fixe minimum.

La correction de la construction est affirmée dans le lemme suivant :

Lemme 25. Pour tout message m M et K A nous avons que

$$[m]D(K) = un(\{m\} \quad K) \cap A.$$

Preuve. La preuve est simple et peut être réalisée par induction sur la structure de m, en utilisant la définition de an(·).

Machine Translated by Google

5.1.2. Le transducteur Tder

Le TTAC Tder est basé sur une idée simple, motivée par le lemme 18. Pour le décrire, nous avons besoin de plus de notation. Pour un ensemble K A, soit (K) la signature définie par

$$(K) = \{\text{encs}_{un}, \text{hash}_a \mid a \in K\} \cup \{\text{enc}_{a_k} \mid k \in K \cap K\}.$$

En utilisant cette notation, le lemme 18 implique :

Lemme 26. Soient m, m M. Alors les énoncés suivants sont équivalents :

(A) m d({m}).

(B) m est de la forme $t[m_0, ..., m_{n-1}]$ où les mi sont obtenus par décryptages et découpages successifs à partir de m, et les clés en t peuvent également être obtenues par décryptages et fractionnements successifs à partir de m, c'est-à-dire qu'il existe un terme linéaire $t(x_0, ..., x_{n-1})$ T(an({m})∩A)(X) tel que $m = t[m_0, ..., m_{n-1}]$ où mi an({m}) pour tout i < n.

Le TTAC Tder a un état initial distinct qI et, pour chaque K A, il y a deux états (qS ,K) et (qA,K), les indices rappelant « syn » et « an ». Le transducteur fonctionne en trois phases sur un message m donné. La première phase n'est qu'une étape et détermine simplement l'ensemble K de clés qui peuvent être découvertes à partir de la donnée message m, c'est-à-dire qu'il détermine K = an{m} ∩ A. Dans la deuxième phase, le terme t ci-dessus est généré, c'est-à-dire de manière non déterministe, un message mS est construit qui peut être écrit comme $t[m, m, m, ..., x_{r-1}]$, m] où $t(x_0, ...$ est un terme linéaire de T(an({m})∩A)(X). Dans la troisième phase, chaque copie de m dans t[m, ... m] est (de manière non déterministe) remplacé par un message de an({m}). Le lemme ci-dessus garantit que Tder calcule exactement les messages dérivable de m.

Pour être plus précis, nous avons les transitions suivantes dans Tder. Comme forTder aucun registre n'est utilisé, dans ce qui suit on écrit s(t) au lieu de s(, t) où s est un état de Tder, c'est-à-dire s = qI , s = (qS ,K), ou s = (qA,K) pour un certain K A.

Pour la première phase, pour tout d QD, Tder contient la transition

$$q_I(x) \to d\ (q_S, d(\ ))(x)$$

Pour la deuxième phase, pour tout K A, Tder contient les transitions suivantes :

$$\begin{aligned}
&(q_S,K)(x) \to (q_S,K)(x),(q_S,K)(x) && \\
&(q_S,K)(x) \to \text{encs}_{un}((q_S,K)(x)) && \text{pour } a \in K \\
&(q_S,K)(x) \to \text{enc}_{a_k}((q_S,K)(x)) && \text{pour } k \in K \cap K \\
&(q_S,K)(x) \to \text{hash}_a((q_S,K)(x))\ (q_S,K) && \text{pour } a \in K \\
&(x) \to (q_A,K)(x) &&
\end{aligned}$$

Pour la troisième phase, pour tout K A, Tder contient les transitions suivantes :

$$\begin{aligned}
&(q_A,K)(x) \to x && \\
&(q_A,K)(x) \to a && \text{pour } a \in K \\
&(q_A,K)(x_0, x_1) \to (q_A,K)(x_0) && \\
&(q_A,K)(x_0, x_1) \to (q_A,K)(x_1) && \\
&(q_A,K)(\text{encs}_{un}(x)) \to (q_A,K)(x) && \text{pour } a \in K \\
&(q_A,K)(\text{enc}_{a_k}(x)) \to (q_A,K)(x)\ (q_A,K) && \text{pour } k-1 \in K \cap K \\
&(\text{hash}_a(x)) \to (q_A,K)(x) && \text{pour } a \in A
\end{aligned}$$

Écrire der au lieu de $\text{Tder}_{,}$ nous pouvons affirmer :

Lemme 27. der(m) = d({m}) pour tout m M.

Preuve. A partir de la construction de Tder, il est facile de voir que si m est un message avec K = an({m}) ∩ A et m est tel que qI(m) m , alors

$$q_I(m)\ (q_S,K)(m) \to t[(q_A,K) \quad t[(q_S,K)(m), ..., (q_S,K)(m)]$$
$$(m), ..., (q_A,K)(m)] \qquad t(m_0, ..., m_{n-1}) = m \tag{20}$$

Machine Translated by Google

où t T(an({m})∩A)(X) et mi an({m}) pour tout i<n. Ceci montre que der(m) d({m}).

De même, il est facile de voir que pour tout choix de t et m comme ci-dessus, on a (20) avec K = an({m}) ∩ A, qui implique d({m}) der(m).

Notez que même si nous permettions à l'intrus de générer des constantes anonymes, nous pourrions modéliser un tel intrus par un TTAC puisque les TTAC peuvent générer des constantes anonymes. Plus précisément, on pourrait simuler l'intrus par une composition de deux TTAC : le premier TTAC copie l'entrée dans la sortie et ajoute un nombre arbitraire de nouvelles constantes anonymes à la sortie. Ceci peut être réalisé en utilisant des transitions ε . Les deuxièmes transducteurs fonctionne exactement comme le transducteur décrit ci-dessus. Notez que ce transducteur obtient le message d'origine avec avec les constantes générées par le premier transducteur en entrée. Cependant, comme indiqué dans la remarque 23, étant donné que l'intrus n'est pas plus puissant s'il peut générer des constantes anonymes, il suffit de modéliser l'intrus plus simple.

5.2. Réduction au problème du mot pré-image itéré

Nous réduisons maintenant InterleavingAttack à IteratedPreImage en formulant une attaque comme une composition de transducteurs. Nous devons d'abord introduire deux variantes de Tder et une variante de Ti , principalement pour transmettre l'intrus connaissances d'un transducteur à l'autre.

La première variante de Tder, appelée Tcopy, copie son entrée dans le premier composant d'une paire et simule Tder sur le deuxième composant, c'est-à-dire,

$$\text{Copier}_{\text{der}} = \{(m, m, m') \mid m \in d(\{m\})\}.$$

La copie TTAC $_{\text{der}}$ peut être dérivé de Tder de manière simple. Nous dotons Tder d'un état supplémentaire, dire $q_{\text{je}}^{\text{copie}}$, et le déclarer comme étant l'état initial de Tcopy de . De plus, nous ajoutons la transition suivante :

$$q_{\text{je}}^{\text{copie}}(\ ,x) \to x, q_I(\ ,x).$$

Rappelons que qI est l'état initial de Tder. Pour faciliter la notation, soit $^{\text{copie}}_{\text{der}} = \text{Copier}_{\text{der}}$.

La deuxième variante, appelée $T_{\text{der}}^{\text{défi}}$, attend une entrée de la forme m, m , copie le deuxième composant dans la sortie et simule Tder sur le premier composant. Nous appelons ce transducteur le transducteur de défi car il reçoit dans le deuxième composant le défi et tente de le dériver du premier composant, l'intrus connaissance. Encore une fois, cette variante de Tder peut facilement être obtenue à partir de Tder. Nous ajoutons un état supplémentaire, disons $q_{\text{challenge}}$, lequel est l'état initial de $T_{\text{der}}^{\text{défi}}$, et nous ajoutons également la transition suivante :

$$q_{\text{challenge}}(\ ,x0,x1) \to q_I(\ ,x0),x1.$$

Laisser $^{\text{défi}}_{\text{der}} = T_{\text{der}}^{\text{défi}}$.

Enfin, nous introduisons une variante $\hat{T}_i$ de Ti pour (i) transmettre la connaissance de l'intrus et (ii) pour satisfaire la condition (2) dans la définition de InterleavingAttack, c'est-à-dire que les constantes anonymes générées dans une action de réception-envoi sont différent des constantes anonymes générées jusqu'à présent. À cette fin, Ti n'accepte que des paires en entrée, des copies le premier composant dans la sortie (ce composant représente la connaissance de l'intrus) et simule Ti sur $\hat{}$ le deuxième composant (ce composant correspond à l'entrée pour Ti). Évidemment, Ti défini de cette façon accomplit (i). Mais il accomplit également (ii) puisque selon notre définition des calculs d'un TTAC, anonyme les constantes générées par un transducteur sont différentes de celles qui se produisent dans l'entrée. C'est encore simple pour obtenir $\hat{}$ Ti de Ti : Nous ajoutons un état q à l'ensemble des états de Ti et le déclarons comme étant le (seul) état initial de $\hat{}$ Ti , et pour tout état initial qI de Ti , nous ajoutons la transition

$$q(\ ,x0,x1) \to x0, q_I(\ ,x1).$$

Soit $\hat{i} = \hat{}_{\text{Ti}}$.

Machine Translated by Google

Nous avons aussi besoin du langage des arbres

$$R = \{a, a \mid a \in A\} \cup \{c, c \mid c \in C\}$$

qui, en utilisant l'exemple 1, peut facilement être vu comme étant reconnaissable par TAAC. Pour un ensemble fini $S = \{u_0, \ldots, u_{n-1}\}$ de messages soit m_S le message défini par

$$m_S = u_0, u_1, \cdots u_{n-2}, u_{n-1} \cdots$$

(l'ordre des u_i s n'a pas d'importance) ; cela garantit que nous avons $d(S) = d(m_S)$. Enfin, soit

$$= \text{défi}_{\text{der}} \circ \hat{}l-1 \circ \text{copie}_{\text{der}} \circ \hat{}l-2 \circ \text{copie} \circ \cdots \circ \text{der} \circ \text{copie}_{\text{der}} \circ \hat{}0 \circ \text{copie}_{\text{de}}.$$

Ensuite, par construction, nous obtenons la caractérisation suivante pour le problème InterleavingAttack.

Lemme 28. Pour tout S et $T_0, \ldots, T_{l-1}$ comme dans la définition du problème InterleavingAttack, on a

$$(S, T_0, \ldots, T_{l-1}) \in \text{Attaque entrelacée iff } m_S \in \ ^{-1}(R).$$

Avec le corollaire 12, cela implique immédiatement :

Théorème 29. InterleavingAttack est décidable.

Ceci conclut la preuve du théorème 24.

Par réduction du problème d'intersection pour les automates d'arbres descendants - étant donné une séquence d'arbres descendants $A_1, \ldots, A_n$, décide si l'intersection $L(A_1) \cap \cdots \cap L(A_n)$ est vide—, ce qui est connu pour être un automate EXP TIME-complet [35], il est facile de voir que les problèmes Attack et InterleavingAttack sont EXPTIME-difficiles puisque les TTAC peuvent simuler de tels automates d'arbres et en composant des TTAC, on peut simuler l'intersection de de tels automates. Étant donné que notre procédure de décision pour le problème du mot pré-image itéré n'est pas élémentaire, cela c'est aussi le cas pour les problèmes Attack et InterleavingAttack. Il reste donc à trouver une complexité serrée lié à ces problèmes.

6. Extensions du modèle et résultats d'indécidabilité

Comme mentionné dans la section 4, les hypothèses de base de notre modèle de protocole basé sur un transducteur d'arbre et de nos modèles pour les protocoles sans boucle, coïncident (nombre fini d'actions de réception-envoi, intrus Dolev-Yao sans limite sur la taille des messages). En fait, dans le modèle de protocole basé sur TTAC tel qu'introduit dans la section 4, de nombreux protocoles non en boucle peuvent être analysés avec la même précision que dans les modèles décidables pour les protocoles non en boucle avec clés atomiques (voir, par exemple, [2]). Plus précisément, c'est le cas pour les protocoles où (a) les actions de réception-envoi peuvent être décrit par des règles de réécriture avec un côté gauche linéaire, puisque les TTAC peuvent simuler toutes ces règles de réécriture, et (b) seule une quantité finie d'informations doit être transmise d'une action de réception-envoi à la suivante. inclut par exemple de nombreux protocoles de la bibliothèque Clark-Jacobs [11]. (Pour illustrer cela, dans la section 7.2, nous fournir un modèle formel basé sur TTAC du protocole de clé publique Needham-Schroeder.)

Cependant, certaines fonctionnalités présentes dans les modèles décidables pour les protocoles non en boucle sont manquantes dans le TTAC-modèle de protocole basé sur :

(1) tests d'égalité pour les messages de taille arbitraire, qui sont possibles lorsque
   (a) les côtés gauches des règles de réécriture peuvent être non linéaires (cela correspond à autoriser les côtés gauches non linéaires dans les transitions de TTAC) ou
   (b) des messages arbitraires peuvent être transmis d'une action de réception-envoi à une autre et peuvent ensuite être comparés avec d'autres messages [2,34,26,5] ;
(2) des clés complexes, c'est-à-dire des clés qui peuvent être des messages arbitraires [34,26,5] ; et

Machine Translated by Google

(3) assouplir l'hypothèse d'algèbre à terme libre en ajoutant l'opérateur XOR [8,13] ou l'exponentielle Diffie-Hellman [9].

Le principal résultat de cette section est que ces fonctionnalités ne peuvent pas être ajoutées sans perdre la décidabilité.

Nos résultats d'indécidabilité montrent que si un test d'égalité peut être effectué, alors Attack est indécidable. dans (1) le test d'égalité est explicitement présent, dans (2) et (3) des tests d'égalité implicites sont possibles. Dans ce qui suit sous-sections, les résultats d'indécidabilité sont présentés en détail.

Nous remarquons que lorsqu'un intrus est autorisé à utiliser un nombre illimité de copies d'un principal pour exécuter une attaque, c'est-à-dire que le protocole est analysé par rapport à un nombre illimité de sessions - et donc, reçoit - envoyer des actions—, alors Attack est également indécidable. Cela n'est pas surprenant, car il en va de même pour les modèles de protocoles non en boucle (voir, par exemple, [2]).

6.1. Problème de codage de la correspondance de Post

Nous commençons par un résultat d'indécidabilité qui est purement automate-théorique. Il sera ensuite utilisé sur et pour obtenir les résultats d'indécidabilité liés au protocole. Plus précisément, nous montrons comment modéliser le protocole de Post Problème de correspondance (PCP) en composant des transducteurs.

Rappelons qu'une instance de PCP est composée de deux séquences = 1, ... , n et = 1, ... , n des mots sur un alphabet à deux lettres. Le problème est de déterminer s'il existe une suite u = i0, ... telle que appelée solution. Les , ik des indices deux mots je0 ...je = i0 ...ik . Une telle séquence d'indices est appelée une solution réalisable ; toute séquence est simplement sont notés (u) et (u), respectivement.

Nous allons encoder un mot = a0 ...al−1 avec ai par le terme a0,a1, ... ,al−1, · · ·, que nous notons []. Ici, est une nouvelle constante. Nous allons également encoder les indices i {1, ... , n}. A cet effet, soit b une nouvelle constante et soit bi le mot b ··· b de longueur i. Alors, i est codé par [bi], qui, pour des raisons pratiques, est également noté [i]. Si u = i0i1 ...il est une solution, on écrit [u] pour [i0],[i1], ... ,[il], ...

En plus de b, nous utiliserons les nouvelles constantes b1, b2 et secret. Ainsi, l'ensemble des messages atomiques est défini être A = {b, b1, b2, , secret}.

Nous décrivons ensuite les transducteurs T0, T1 et T2 de telle sorte qu'une instance du PCP comme ci-dessus ait une solution réalisable si et seulement si dans T2 (T1 (T0 (b))) il existe un terme t, t.

Le transducteur T0 génère le codage de toute solution et présente donc les transitions suivantes :

qI(b) →[i], q(b) q(b) pour i {1, ... ,n},
→[i], q(b) q(b) → . pour i {1, ... ,n},

Le transducteur T1 est constitué d'une seule transition et double chaque terme :

qI(x0, x1) →x0, x1,x0, x1.

Dans la transition ci-dessus, nous écrivons x0, x1 au lieu de x pour nous assurer que la liste des indices reçus de T0 n'est pas vide.

Le transducteur T2 prend n'importe quelle paire de solutions codées et remplace les listes par la concaténation des mots correspondants (codés comme décrit ci-dessus). Il y a une transition qui démarre la substitution/concaténation processus dans chaque composant :

qI(x, x) →q(x), q(x).

De plus, pour chaque i {1, ... , n} avec „ = a0 ··· al−1 et „ = b0 ··· bm−1, il existe des transitions pour les substi- processus de tution/concaténation :

q([i], x) →a0,a1, ... ,al−1, q(x) · · ·,
q([i], x) →b0,b1, ... ,bm−1, q(x) · · ·.

Machine Translated by Google

Enfin, il existe deux transitions pour arrêter le processus de substitution/concaténation :

$$q(\quad) \to \quad ,$$
$$q(\quad) \to \quad .$$

Maintenant, le lemme suivant est facile à voir :

Lemme 30. Pour une instance donnée du PCP, soit T0, T1, T2 les transducteurs définis ci-dessus. Alors les énoncés suivants sont équivalents : (A) L'instance a

une solution réalisable.
(B) Il existe un terme t tel que t, t T2 (T1 (T0 (b))).

Avant d'appliquer cela dans le cadre cryptographique, nous l'utilisons pour un problème théorique d'automates.

Nous définissons l'extension suivante de TTAC. Transducteur d'arbre descendant avec côté gauche non linéaire (TTNL) est un TTAC avec des transitions de la forme définie dans (3) mais où t n'est pas nécessairement linéaire.

Maintenant, nous pouvons prouver :

Théorème 31. Pour les TTNL, le problème de pré-image itéré est indécidable.

Preuve. La preuve est par réduction à partir du PCP. Étant donné une instance de PCP comme ci-dessus, nous construisons T0, T1 et T2 comme ci-dessus. De plus, nous construisons le TTNL T3 défini par la transition unique

$$q_I(x, x) \to \text{secret}.$$

De toute évidence, (t) / = seulement si t = t0, t1 où t0 = t1, et si c'est le cas, alors (t) = {secret}. De ceci et de T3 T3)−1 ({b}), lemme 30, nous obtenons que l'instance a une solution réalisable si et seulement si secret (T3 ce qui complète la description de la réduction désirée.

Nous notons que pour que ce résultat d'indécidabilité soit vérifié, il est essentiel d'autoriser les ε -transitions dans les transducteurs : si les ε -transitions ne sont pas autorisées, alors sur une entrée donnée, un transducteur ne peut produire qu'un nombre fini de sorties modulo de nouvelles constantes anonymes. De plus, il est facile de limiter le nombre de constantes anonymes à considérer.
Ainsi, nous obtenons :

Observation 32. Pour les TTNL sans ε -transitions, c'est-à-dire uniquement avec des ε -transitions de la forme (17), le problème du mot pré-image itéré est décidable.

6.2. Tests d'égalité sur les messages

Commençons par la définition suivante. Un protocole dans lequel les actions de réception et d'envoi sont définies par des TTNL est appelé un protocole basé sur TTNL.

Nous montrons ce qui suit :

Théorème 33. Pour les protocoles basés sur TTNL, Attack et InterleavingAttack sont indécidables.

Preuve. La preuve est obtenue par réduction à partir du PCP. Nous démontrons la preuve pour InterleavingAttack ; celle pour Attack est essentiellement la même.

La réduction est essentiellement la même que celle de la preuve du théorème 31, mais nous ne pouvons pas simplement prendre les quatre transducteurs comme description du protocole, car l'intrus pourrait interférer et nous devons produire une action de sortie de défi. Pour empêcher l'intrus d'interférer, nous chiffrons simplement la sortie de chaque transducteur avec une clé inconnue de l'intrus et nous nous assurons que seuls les messages chiffrés sont acceptés. De plus, nous utilisons une clé différente pour chaque transducteur. Cela garantit que l'ordre dans lequel les transducteurs sont appliqués est préservé.

Plus précisément, on modifie les transducteurs comme suit : − T0 : On

ajoute la transition $q \quad (x) \to \text{encs } b0_{je} \quad (q_I(x))$ à T0 et déclarer q_{je} être l'état initial.

Machine Translated by Google

− T1 : On ajoute la transition q je (encs b0 (x)) →encs b1 (qI(x)) et déclarez q je comme étant l'état initial.

On ajoute la transition q je (qI(x)) et on déclare q je comme étant l'état initial. b1 b2 (encs − T2 : On ajoute la transition q je (x)) → qI(x)) et on déclare

comme étant l'état initial. b2 De plus, on ajoute le transducteur T4 défini par qI(b) →secret. Maintenant, il est clair que

l'instance de InterleavingAttack déterminée par les transducteurs T0, ..., T4 et la connaissance initiale S = {b} est résoluble ssi l'instance donnée du PCP a une solution faisable.

6.3. Clés complexes et résultats de défi

Pour modéliser des clés complexes, nous remplaçons le symbole unaire encs (·) par le symbole binaire enc(·, ·). Le message enc(m, m) avec m, m M représente le message m chiffré par m. Notez que la clé m peut être un complexe message.

En conséquence, nous étendons la capacité de l'intrus à dériver des messages. Si l'intrus connaît m, m M, alors il peut générer enc(m, m). S'il connaît enc(m, m) et m, alors il connaît m aussi.

Les transducteurs utilisés pour définir les principaux ne sont pas étendus, sauf que la signature change.

Nous avons:

Théorème 34. Pour les protocoles basés sur TTAC avec des clés complexes, Attack et InterleavingAttack sont indécidables.

Preuve. Pour le voir, il suffit d'observer que dans la réduction de la preuve du théorème 33, T3 peut être remplacé par le transducteur défini par

q je (encs (x, x) →encs b2 encs b3 (x)(secret), encs b3 (x),

où b3 est une nouvelle constante inconnue de l'intrus. Cela garantit que l'intrus peut obtenir le secret ssi les messages substitués à x et x coïncident. En d'autres termes, la réduction utilise le fait que le déchiffrement pour les clés complexes nécessite des tests d'égalité pour les messages de taille arbitraire.

De même, nous obtenons un résultat d'indécidabilité si nous autorisons des actions de sortie de défi arbitraires. Plus précisément, le réglage est le suivant. L'espace de message est défini comme dans la section 4, mais dans une action de sortie de défi, le principal est autorisé à renvoyer n'importe quel message de M comme défi (plutôt qu'un élément de C A).

Théorème 35. Pour les protocoles basés sur TTAC avec des messages de sortie de défi arbitraires, Attack et InterleavingAttack sont indécidables.

Preuve. Nous modifions en conséquence la réduction issue de la preuve du théorème 33. Tout d'abord, nous définissons le transducteur T3 par la transition

qI encs b2 (x, x) →encs b3 (x), encs b4 (x)

et T4 par

qI encs b4 (x) →encs b3 (x),

où b3 et b4 sont de nouvelles constantes inconnues de l'intrus. Ainsi, le défi pour l'intrus est encs (x) b3 qu'il ne peut dériver que si les messages substitués à x et x dans la transition de T2 coïncident.

6.4. XOR et exponentiation Diffie-Hellman

Nous prouvons ensuite que l'extension du modèle de protocole par ou exclusif (XOR) ou par exponentiation Diffie-Hellman conduit à l'indécidabilité. Nous considérons d'abord XOR.

Machine Translated by Google

L'espace des messages est étendu comme suit. Nous ajoutons la constante 0 et le symbole binaire qui, entre autres, possède la propriété algébrique suivante : m m = 0 (voir, par exemple, [8] pour d'autres propriétés de XOR.) Ces propriétés induisent une relation d'équivalence ≡sur les messages. Par exemple, encs (m m) ≡encs (0). Notez que cela donne un moyen de comparer les messages pour l'égalité.

En général, on peut étendre l'intrus en lui donnant la possibilité de combiner des messages à l'aide de l'opérateur XOR. Pour le résultat d'indécidabilité, il n'y a cependant aucune différence que l'intrus soit ou non doté de cette capacité.

De plus, il faudrait que les transducteurs fonctionnent sur des classes d'équivalence de messages selon le XOR théorie. Cependant, il est facile de voir que pour la réduction, cela n'a pas non plus d'importance.

Théorème 36. Pour les protocoles TTAC avec XOR, le problème Attack et le problème InterleavingAttack sont indécidables.

Preuve. Pour montrer l'indécidabilité, nous pouvons à nouveau modifier la réduction de PCP comme décrit dans la preuve du théorème 33. Au lieu de simplement T3, nous avons maintenant besoin de deux transducteurs, T3 et T4. Le transducteur T4 dans la preuve du théorème 33 est maintenant appelé T5.

Le transducteur T3 est maintenant donné par

ql encs $_{b2}$ (x, x) →encs $_{b3}$ x x .

Ainsi, l'intrus obtient l'encs b3 (0) ssi les messages substitués à x et x coïncident.

Maintenant, T4 vérifie si l'intrus connaît l'encs $_{b3}$ (0) et si c'est le cas, renvoie secret. Autrement dit, T4 est défini par

ql encs (0) →secret. b3

Le transducteur T5 est défini comme T4 dans la preuve du théorème 33.

Une réduction similaire est possible pour l'exponentiation Diffie-Hellman [9] puisque la normalisation implique également la comparaison de messages arbitraires, par exemple, Exp(g, x · x−1) = 1 ssi x = x . On obtient :

Théorème 37. Pour les protocoles basés sur TTAC avec exponentiation Diffie-Hellman, Attack et InterleavingAttack sont indécidables.

7. Modélisation des protocoles cryptographiques

Dans cette section, nous présentons des modèles de protocoles formels basés sur TTAC pour le protocole d'authentification récursive (comme exemple de protocole récursif) et le protocole de clé publique Needham-Schroeder (comme exemple de protocole sans boucle).

7.1. Le protocole d'authentification récursive

Dans la section 7.1.1, nous donnons d'abord une description informelle du protocole d'authentification récursive (protocole RA). La section 7.1.2 fournit un modèle formel basé sur TTAC pour ce protocole. Dans ce qui suit, nous abrégeons les messages de la forme m0, ... ,mn−1, mn · · · par m0 ··· mn ou m0, ... , mn.

7.1.1. Description informelle du protocole RA

Le protocole RA a été proposé par [7] et étend le protocole d'authentification par [31] en ce sens qu'il permet d'établir des clés de session entre un nombre illimité a priori de principaux dans une exécution de protocole. Notre description du protocole RA suit [32].

Dans le protocole RA, on suppose qu'un serveur de distribution de clés S partage des clés à long terme avec les principaux. La figure 1 illustre une exécution typique du protocole. Dans cette exécution, A souhaite établir une clé de session avec B et B souhaite

Machine Translated by Google

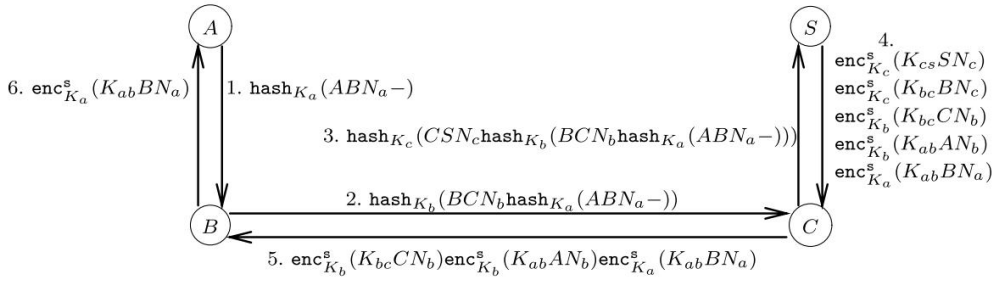

Fig. 1. Une exécution du protocole d'authentification récursive.

établir une clé de session avec C. Le nombre de principaux impliqués dans une exécution de protocole n'est pas limité. En particulier, C pourrait envoyer un message à un principal D afin d'établir une clé de session avec D et D pourrait continuer et envoyer un message à E, et ainsi de suite. Dans l'exécution de protocole illustrée dans la Figure 1, nous supposons que C ne veut pas parler à un autre principal et envoie donc un message au serveur de distribution de clés S, qui est impliqué dans chaque exécution de protocole.

Dans la Fig. 1, Ka (resp. Kb, Kc) désigne la clé à long terme partagée entre A et S (resp. B et S, C et S). Avec Na, Nb et Nc, nous désignons les nonces (c'est-à-dire les nombres aléatoires) générés par A, B et C, respectivement. Enfin, Kab, Kbc et Kcs sont les clés de session générées par le serveur et utilisées par les principaux pour la communication sécurisée entre A et B, B et C, et C et S, respectivement. Les nombres ((1)–(6)) attachés aux messages indiquent uniquement l'ordre dans lequel les messages sont envoyés et n'appartiennent pas au protocole.

Nous allons maintenant examiner de plus près les messages échangés entre les principaux dans l'ordre dans lequel ils sont envoyés : dans le premier message (1), le principal A indique qu'il demande une clé de session au serveur pour une communication sécurisée avec B. Le symbole « − » indique que ce message a lancé l'exécution du protocole. Maintenant, dans le deuxième message (2), B envoie quelque chose de similaire à C mais avec le message de A au lieu de « − », indiquant qu'il souhaite partager une clé de session avec C. Comme mentionné précédemment, cette étape peut être répétée autant de fois que souhaité, ce qui produit une pile de demandes toujours croissante. Le processus est terminé si un principal contacte S. Dans notre exemple, nous supposons que C ne demande pas une autre clé de session et envoie donc le message reçu de B à S (3). Ce message est maintenant traité par S. Cela peut être fait de différentes manières. Dans ce qui suit, nous décrivons une manière possible.

Premièrement, S vérifie si la requête externe est bien adressée à S. Si c'est le cas, S génère une nouvelle clé de session et la stocke. S traite ensuite les requêtes en commençant par la plus externe. En général, S a une « trame » contenant deux requêtes à la fois. Dans l'exemple, S commence par une trame contenant les requêtes CSNc et BCNb. Ainsi, S sait que C veut parler à S et que B veut parler à C. Par conséquent, S doit générer deux certificats pour C, l'un contenant la clé de session pour la communication avec S et l'autre pour la communication avec B. Ces certificats sont générés par S comme suit. Le premier contient la clé de session stockée, le nom S du serveur et le nonce Nc de C. Pour le deuxième certificat, S génère une nouvelle clé de session, la stocke pour une utilisation ultérieure, puis assemble le deuxième certificat pour C contenant la clé de session qui vient d'être générée, le nom de B et le nonce Nc de C. À ce stade, tous les certificats pour C ont été préparés. Par conséquent, S déplace la trame d'une requête plus loin et traite cette trame comme avant. Notez que la trame contient maintenant les requêtes BCNb et ABNa, et que pour le premier certificat envoyé à B, S utilise la clé de session stockée. Une fois les deux certificats pour B préparés, S déplace la trame d'une requête plus loin. Cette trame ne contient maintenant qu'une seule requête, à savoir ABNa−. Le marqueur « − » indique que A a démarré le protocole. Par conséquent, un seul certificat pour A est généré. Il contient la clé de session stockée, le nom de B et le nonce Na de A. Après cela, S a préparé tous les certificats et les renvoie au principal qui a appelé S. Dans l'exemple, il s'agit de C.

Le principal C accepte les deux premiers certificats, extrait les deux clés de session et transmet le reste du message à son prédécesseur dans la chaîne (5). Ensuite, B fait de même et transmet le dernier certificat à A (6). Puisque selon le modèle de l'intrus, le message envoyé par S est envoyé à l'intrus, nous pouvons supposer que chaque principal ne reçoit que ses propres certificats et n'a pas besoin de transmettre le reste du message à son prédécesseur.

Machine Translated by Google

7.1.2. Le modèle de protocole basé sur TTAC

Nous fournissons maintenant une description formelle du protocole RA dans le modèle de protocole basé sur TTAC. Dans ce soit P0, ... , qui suit, Pn sont les principaux participant au protocole RA. Nous supposons que Pn = S est le serveur. Chaque Pi, i<n, partage une clé à long terme Ki avec S. Nous modélisons la corruption statique et par conséquent partitionnons l'ensemble des principaux en principaux honnêtes H et principaux malhonnêtes D, c'est-à-dire que nous avons H D = {0, ... , n} et H ∩ D = . Nous supposons que S est honnête, et donc, S H. L'intrus jouera le rôle des principaux malhonnêtes et à cette fin, ses connaissances initiales contiendront Ki pour chaque i D.

Dans ce qui suit, nous spécifions d'abord les agents honnêtes et le serveur. Nous mettons ensuite tout ensemble pour formaliser spécifier le protocole RA comme un protocole basé sur TTAC.

Modélisation des agents honnêtes. Un agent honnête Pi, i<n et i H, effectue deux actions de réception-envoi et est donné tuple i 0 = T par le T'_0, T'_1, Je . Les différents composants sont définis ensuite. Le nonce envoyé par Pi dans la requête le message est désigné par la constante Ni.

Le transducteur de message T'_0 pour envoyer le message de demande se compose de deux transitions. La première est

(requête, , Pj)(, init) →hashKi(Pi, Pj ,Ni, −),

et le deuxième est

(requête, Pj, Pj)(, hasha(Pj, Pi, x0, x1)) →hashKi(Pi, Pj , Ni, hasha(Pj, Pi, x0, x1)),

où x0 et x1 sont des variables, jn, j<n, a A, et init A est un message atomique connu de l'intrus.
La première transition est appliquée si Pi lance une exécution de protocole et appelle Pj . La deuxième transition est appliquée si Pi est appelé par Pj et envoie un message à Pj . Les états initiaux de T sont ($r_0$equest, , Pj) et (request, Pj, Pj) pour chaque jn et j<n. est une action de sortie de

Le transducteur T'_1 défi qui reçoit une clé de session et l'envoie à l'intrus en tant que défi dans le cas où le partenaire de communication est honnête. Notez que le secret d'une clé de session ne doit être garanti qu'entre les principaux honnêtes. Si l'un des partenaires de communication est malhonnête, il est clair que l'intrus peut obtenir la clé de session en suivant simplement le protocole. Par conséquent, dans ce cas, le défi pourrait toujours être relevé par l'intrus. Pour chaque j1,j2,j3 n et j4,j5 < n avec j1,j3,j4 H, T contient les transitions suivantes : T'_1

(clé, , Pj1)(, encs $_{Ki}$(x0, Pj1,Ni)) →x0
(clé, Pj2 , Pj3)(,encs $_{Ki}$(x0, Pj3 ,Ni), encs $_{Ki}$(x1, Pj2 ,Ni)) →x0
(clé, Pj4 , Pj5)(,encs $_{Ki}$(x0, Pj5 ,Ni), encs $_{Ki}$(x1, Pj4 ,Ni)) →x1

où x0 et x1 sont des variables. La première transition est appliquée si Pi a initié l'exécution du protocole pour la communication avec Pj1. Nous exigeons que j1 H car si Pj1 était malhonnête, alors, comme déjà mentionné ci-dessus, il est clair que l'intrus pourrait obtenir x0 en suivant simplement le protocole. Les deux autres transitions sont appliquées si Pi a été appelé par Pj2 et Pj4 , respectivement, et appelé Pj3 et Pj5 , respectivement. Pour la même raison que ci-dessus, nous exigeons que j3,j4 H. Tous les états se produisant dans T 1 Il reste à définir Ii . Nous voulons garantir que Pi se souvient de qui il a appelé et qui veut communiquer avec Pi. Par conséquent, nous définissons

T'_1 sont des états initiaux.

Ii = {((requête, , Pj),(clé, , Pj)) | jn} {((requête,
Pj, Pj),(clé, Pj, Pj)) | jn,j<n}.

Ce modèle d'agents est plus précis que celui présenté dans [20] où des transducteurs de mots ont été utilisés à la place des transducteurs d'arbres. Alors que dans [20], les nonces (les messages substitués à x0 dans T) devaient être typés car un T'_0 transducteur de mots ne peut pas analyser un message arbitraire, ici n'importe quel message peut être substitué à x0.

Machine Translated by Google

Modélisation du serveur. Étant donné que le serveur S = Pn n'exécute qu'une seule action de réception-envoi, il peut être décrit par un seul transducteur de message, que nous appelons Tn. Formellement, Pn est défini par le tuple n = (Tn,{start}) où start est l'état initial de Tn, avec Tn défini ensuite.

Le transducteur Tn possède deux états et fonctionne comme décrit dans la section 7.1.1. Dans l'état start, l'état initial, Tn vérifie si la première requête est adressée à S et génère une clé de session qui est stockée dans le registre. Dans l'état read, les requêtes sont traitées. Dans cette phase, le registre est utilisé pour stocker une clé de session tout en déplaçant la trame vers la requête suivante.

Les transitions de Tn sont spécifiées comme suit :

$$\text{start}(\quad, \text{hashKi}(Pi, Pn, x0, x1)) \rightarrow \text{read}(vN, \text{hashKi}(Pi, Pn, x0, x1))$$

$$\text{read}(vR, \text{hashKi}(Pi, Pj, x0, -)) \rightarrow \text{encs}_{Ki}(vR, Pj, x0)$$

$$\text{lire}(vR, \text{hashKi}(Pi, Pj, x0, \text{hashKi}(Pi, Pi, x1, x2))) \rightarrow (vR, \text{encs}_{Ki}(Pj, x0), \text{encs}_{Ki}(vN, Pi, x0), \text{lecture}(vN, \text{hashKi}(Pi, Pi, x1, x2))$$

où i, i , jn et x0, x1, x2 sont des variables qui prennent des messages arbitraires, et vR et vN sont respectivement les variables pour le registre et la nouvelle constante anonyme.

Ce modèle de serveur est plus précis que celui présenté dans [20]. Tout d'abord, nous n'avons pas besoin de supposer que les nonces sont typés. Le serveur accepte n'importe quel message comme nonce. Dans le modèle de transducteur de mots, cela n'était pas possible car (i) les transducteurs de mots ne peuvent pas analyser des messages imbriqués arbitrairement et (ii) ils ne peuvent pas copier des messages de taille arbitraire, ce qui est cependant nécessaire dans la dernière transition du serveur. Deuxièmement, les TTAC permettent de générer des constantes anonymes et fournissent ainsi une manière très naturelle de modéliser la création de nouvelles clés de session. Les transducteurs de mots considérés dans [20] n'avaient pas cette capacité. Par conséquent, dans [20], le serveur ne pouvait choisir que parmi un ensemble fini de clés de session. Comme le nombre de clés de session que le serveur doit générer n'est pas fixé a priori, il ne s'agissait que d'une approximation du comportement réel du serveur.

Il est clair qu'avec des modèles décidables pour les protocoles non en boucle [34,26,5,2] le serveur ne peut pas être modélisé fidèlement car ces modèles ne permettent pas de décrire les processus récursifs.

La spécification du protocole RA. Étant donné la spécification des agents honnêtes et du serveur ci-dessus, le protocole RA est désormais formellement spécifié par le tuple

$$(\{i\}_{i \quad H}, \{P0, \ldots \quad, Pn\} \quad \{Ki \mid i \quad D\}),$$

c'est-à-dire que nous modélisons explicitement le comportement des agents honnêtes, y compris le serveur, par les principes basés sur TTAC i. Les agents malhonnêtes sont absorbés par l'intrus qui a dans sa connaissance initiale les noms de tous les mandants et les clés à long terme que les agents malhonnêtes partagent avec le serveur.

Nous notons que, bien que dans une attaque, une seule session d'un agent honnête soit effectuée au plus, l'intrus peut simuler un nombre illimité de sessions d'agents malhonnêtes. En particulier, un agent malhonnête peut être impliqué dans de nombreuses requêtes adressées au serveur et, par conséquent, la longueur de la liste des requêtes envoyées au serveur lors d'une attaque est illimitée.

7.2. Le protocole à clé publique Needham-Schroeder

Le protocole de clé publique Needham-Schroeder est un protocole de réponse aux défis de clé publique bien connu (voir, par exemple, [11] pour une description plus détaillée). Dans notre terminologie, il s'agit d'un protocole sans boucle puisque ses actions de réception-envoi ne nécessitent pas d'itération ou de récursivité.

Dans la notation standard d'Alice et Bob, le protocole peut être décrit comme suit, où KA et KB désignent les A et la clé publique de B, respectivement, et NA et NB désignent les nonces générés par A et B, respectivement :

$$A \rightarrow B : \text{enca}_{Ko}(NA, A)$$

$$B \rightarrow A : \text{enca}_{KA}(AN, NB)$$

Machine Translated by Google

$A \to B : enca_{Ko}(NB)$

$B \to : NB$

La dernière action de B est une action de sortie de défi. C'est-à-dire que B présente NB comme un défi pour l'intrus puisque NB peut être utilisé comme clé de session.

Nous modélisons le protocole comme suit : nous supposons que l'honnête A exécute une instance du protocole en tant qu'initiateur avec l'intrus I. Nous modélisons également une instance de l'honnête B fonctionnant dans le rôle d'un intervenant avec A.

Toutes les actions de réception-envoi peuvent être modélisées par des TTAC avec un seul état, que nous appelons début, et une seule transition.

Le principal A est formellement spécifié comme un principal basé sur TTAC par le tuple $A = ((T^{UN}_0, T^{UN}_1), \{début\} \times \{début\})$ où T^{UN}_0 et T^{UN}_1 spécifiez les deux actions de réception-envoi effectuées par A. Ces TTAC ont les caractéristiques suivantes transition:

T^{TA}_0 démarrer(, init) $\to enca_{KI}(NA, A)$

T^{TA}_1 début(, $enca_{KA}(NA, x)) \to enca_{KI}(x)$

Le principal B est formellement spécifié comme un principal basé sur TTAC par le tuple $B = ((T^{B}_0, T^{B}_1), \{début\} \times \{début\})$ où T^{B}_0 et T^{B}_1 spécifiez les deux actions de réception-envoi effectuées par B. Ces TTAC ont les caractéristiques suivantes transitions :

$T^{tuberculose}_0$ début(, $enca_{Ko}(x, A)) \to enca_{KA}(x,NB)$

$T^{tuberculose}_1$ début(, $enca_{Ko}(NB)) \to NB$

Maintenant, le protocole à clé publique Needham-Schroeder avec honnête A exécute une instance du protocole comme initiateur avec l'intrus I et honnête B exécutant une instance du protocole en tant que répondeur avec A est formellement spécifié comme un protocole basé sur TTAC par le tuple

$(\{A, B\}, \{A, B, KA, KB, KI, K^{-1}_{je}\}),$

c'est-à-dire que les instances des mandants honnêtes A et B sont explicitement spécifiées par A et B. L'intrus initial les connaissances contiennent les noms des mandants et leurs clés publiques ainsi que la clé privée de l'intrus.

La spécification du protocole ci-dessus pourrait bien sûr être étendue par des instances d'autres principaux ou d'autres des exemples de A et B, par exemple ceux dans lesquels ils parlent à d'autres principaux. Celui décrit ci-dessus est suffisant pour découvrir l'attaque trouvée en premier par [23].

Nous soulignons que les nonces ne doivent pas nécessairement être saisis. Les principaux acceptent n'importe quel message comme nonce. En fait, la formulation du protocole Needham-Schroeder telle que décrite ici est aussi précise que d'autres formulations basées sur les modèles pour les protocoles non-bouclants [34,26,5,2]. Tout comme pour le protocole RA, dans [20] il faudrait supposer que les nonces sont typés.

8. Conclusion

L'objectif principal de cet article était de mettre en lumière la faisabilité de l'analyse automatique des protocoles cryptographiques récursifs. Les résultats obtenus ici tracent une frontière assez étroite de la décidabilité de la sécurité pour de tels protocoles. Pour obtenir nos résultats, nous avons introduit des automates d'arbres (TAAC) et des transducteurs (TTAC) sur signatures avec un ensemble infini de constantes (anonymes) et prouvé que pour les TTAC, l'image pré-itérée

Machine Translated by Google

Le problème de mots est décidable. Outre l'application aux protocoles cryptographiques, nous pensons que l'étude des TAAC et des TTAC commencée ici présente un intérêt indépendant.

Un problème ouvert est d'établir des limites de complexité strictes pour nos résultats de décidabilité. Jusqu'à présent, nos transducteurs n'autorisent qu'un seul registre. Nous pensons que nos résultats sont également valables même dans le cas de registres multiples. Nous n'avons cependant pas étudié ce cas, principalement parce que cela n'était pas nécessaire pour notre application et parce que cela aurait rendu les définitions et les preuves plus lourdes. Bien qu'ici nous n'autorisions pas les constantes anonymes comme clés, ce serait une autre extension intéressante de notre modèle. Notre procédure de décision pour analyser la sécurité des protocoles récursifs a été implémentée [30]. Cependant, peu d'efforts ont encore été consacrés aux optimisations. Jusqu'à présent, même sur des versions simplifiées du protocole d'authentification récursive et du protocole de clé publique Needham-Schroeder, l'implémentation manque de mémoire. Cela est dû au fait que pour chaque automate pré-image, l'espace d'état croît de manière exponentielle. Il n'est pas clair, d'un point de vue de la théorie de la complexité, si cette explosion peut être évitée. D'un point de vue pratique, beaucoup plus d'efforts doivent être consacrés à la recherche d'heuristiques permettant de réduire l'espace d'état des automates pré-images.

Reconnaissance

Nous remercions les évaluateurs anonymes pour leurs commentaires utiles et détaillés.

Références

[1] N. Alon, T. Milo, F. Neven, D. Suciu, V. Vianu, XML avec valeurs de données : vérification des types revisitée, Journal of Computer and System Sciences 66 (4) (2003) 688–727.

[2] R. Amadio, D. Lugiez, V. Vanackere, Sur la réduction symbolique des processus avec des fonctions cryptographiques, Theoretical Computer Science 290 (1) (2002) 695–740.

[3] G. Ateniese, M. Steiner, G. Tsudik, Accord de clé de groupe authentifié et amis, dans : Actes de la 5e conférence ACM sur la sécurité informatique et des communications (CCS'98), ACM Press, San Francisco, CA, 1998, pp. 17–26.

[4] M. Bellare, P. Rogaway, Distribution de clés de session prouvée sécurisée : le cas des trois parties, dans : Actes de la vingt-septième conférence annuelle Symposium ACM sur la théorie de l'informatique (STOC'95), ACM, 1995, pp. 57–66.

[5] M. Boreale, Analyse des traces symboliques des protocoles cryptographiques, dans : F. Orejas, P. Spirakis, J. van Leeuwen (éd.), Automates, langages et programmation, 28e colloque international (ICALP 2001), Lecture Notes in Computer Science, vol. 2076, Springer-Verlag, 2001, pp. 667–681.

[6] J. Bryans, S. Schneider, CSP, PVS et un protocole d'authentification récursive, dans : Atelier DIMACS sur la vérification formelle de la sécurité Protocoles, 1997.

[7] J. Bull, D. Otway, Le protocole d'authentification, Rapport technique DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/03, Defence Research Agence, Malvern, Royaume-Uni, 1997.

[8] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, Une procédure de décision NP pour l'insécurité du protocole avec XOR, dans : Actes du dix-huitième symposium annuel de l'IEEE sur la logique en informatique (LICS 2003). IEEE, IEEE, Computer Society Press, 2003, pp. 261–270.

[9] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, Décider de la sécurité des protocoles avec l'exponentiation Diffie-Hellman et les produits en exposants, dans : P. Pandya, J. Radhakrishnan (éd.), FSTTCS 2003 : Fondements de la technologie logicielle et de l'informatique théorique, Lecture Notes in Computer Science, vol. 2914, Springer, Berlin, 2003, pp. 124–135.

[10] Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, Décider de la sécurité des protocoles avec chiffrement à clé publique commutable, Notes électroniques en informatique théorique 125 (1) (2005) 55–66.

[11] J. Clark, J. Jacob, A Survey of Authentication Protocol Literature, version préliminaire Web 1.0, disponible à l'adresse : <http://citeseer.nj.nec.com/>, 1997.

[12] H. Comon, V. Cortier, J. Mitchell, Automates arborescents à une mémoire, contraintes d'ensemble et protocoles ping-pong, dans : F. Orejas, P. Spirakis, J. van Leeuwen (éd.), Automata, Languages and Programming, 28e colloque international (ICALP 2001), vol. 2076, Lecture Notes in Computer Science, 2001, pp. 682–693.

[13] H. Comon-Lundh, V. Shmatikov, Déductions d'intrus, résolution de contraintes et décision d'insécurité en présence de ou exclusif, dans : Actes du dix-huitième symposium annuel de l'IEEE sur la logique en informatique (LICS 2003). IEEE, Computer Society Press, 2003, pp. 271–280.

[14] D. Dolev, A. Yao, Sur la sécurité des protocoles à clé publique, IEEE Transactions on Information Theory 29 (2) (1983) 198–208.

[15] N. Durgin, P. Lincoln, J. Mitchell, A. Scedrov, Réécriture multi-ensembles et complexité des protocoles de sécurité délimités, Journal of Computer Security 12 (2) (2004) 247–311.

[16] J. Engelfriet, Trois hiérarchies de transducteurs, Mathematical Systems Theory 15 (1982) 95–125.

Machine Translated by Google

[17] N. Ferguson, B. Schneier, A Cryptographic Evaluation of IPsec, rapport technique, disponible sur : <http://www.counterpane.com/ip-sec.pdf/>, 2000.

[18] T. Genet, F. Klay, Réécriture pour la vérification de protocoles cryptographiques, dans : D. McAllester (éd.), Actes de la 17e Conférence internationale sur la déduction automatisée (CADE 2000), Lecture Notes in Computer Science, vol. 1831, Springer, 2000, pp. 271–290.

[19] J. Goubault-Larrecq, Une méthode de vérification automatique de protocoles cryptographiques (résumé étendu), dans : Workshop on Formal Methods for Parallel Programming (FMPPTA 2000), vol. 1800, Lecture Notes in Computer Science, Springer, Berlin, 2000, pp. 977–984.

[20] R. Küsters, Sur la décidabilité des protocoles cryptographiques avec des structures de données ouvertes, dans : L. Brim, P. Jancar, M. Kretinsky, A. Kucera (éd.), 13e Conférence internationale sur la théorie de la concurrence (CONCUR 2002), Lecture Notes in Computer Science, vol. 2421, Springer-Verlag, 2002, pp. 515–530.

[21] R. Küsters, T. Truderung, Sur l'analyse atomatique des protocoles de sécurité récursifs avec XOR, dans : W. Thomas, P. Weil (éd.), Actes du 24e Symposium sur les aspects théoriques de l'informatique (STACS 2007), Lecture Notes in Computer Science, vol. 4393, Springer, Berlin, 2007, pp. 646–657.

[22] R. Küsters, T. Wilke, Analyse basée sur des automates de protocoles cryptographiques récursifs, dans : V. Diekert, M. Habib (éd.), 21e Symposium sur les aspects théoriques de l'informatique (STACS 2004), Lecture Notes in Computer Science, vol. 2996, Springer-Verlag, 2004, pp. 382–393.

[23] G. Lowe, Une attaque sur le protocole d'authentification à clé publique Needham-Schroeder, Information Processing Letters 56 (1995) 131–133.

[24] C. Meadows, Extension des techniques d'analyse de protocole cryptographique formelles pour les protocoles de groupe et les primitives cryptographiques de bas niveau, dans : P. Degano (éd.), Actes du premier atelier sur les questions de théorie de la sécurité (WITS'00), 2000, pp. 87–92.

[25] C. Meadows, Questions ouvertes sur les méthodes formelles d'analyse des protocoles cryptographiques, dans : Proceedings of DISCEX 2000, IEEE Computer Presses de la Société, 2000, pp. 237–250.

[26] JK Millen, V. Shmatikov, Résolution de contraintes pour l'analyse de protocole cryptographique à processus limité, dans : Actes de la 8e conférence ACM sur la sécurité informatique et des communications, ACM Press, 2001, pp. 166–175.

[27] D. Monniau, Abstraction des protocoles cryptographiques avec des automates d'arbres, dans : A. Cortesi, G. Filé (éd.), Actes du 6e Symposium international sur l'analyse statique (SAS '99), Lecture Notes in Computer Science, vol. 1694, Springer, Berlin, 1999, pp. 149–163.

[28] D. Monniaux, Procédures de décision pour l'analyse des protocoles cryptographiques par des logiques de croyance, dans : 12th Computer Security Foundation Atelier sur les technologies (CSFW 1999), IEEE Computer Society, 1999, pp. 44–54.

[29] F. Neven, T. Schwentick, V. Vianu, Vers des langages réguliers sur des alphabets infinis, dans : J. Sgall, A. Pultr, P. Kolman (éd.), Mathematical Foundations of Computer Science (MFCS 2001), Lecture Notes in Computer Science, vol. 2136, Springer, Berlin, 2001, pp. 560–572.

[30] A. Obermann, Verifikation kryptographischer Protokolle mit Baumtransduktionen, mémoire de maîtrise, Fachbereich Informatik, TU Ka-Iserslautern, 2004.

[31] D. Otway, O. Rees, Authentification mutuelle efficace et rapide, Operating Systems Review 21 (1) (1987) 8–10.

[32] L. Paulson, Preuves mécanisées pour un protocole d'authentification récursive, dans : 10e atelier IEEE Computer Security Foundations (CSFW-10), IEEE Computer Society Press, 1997, pp. 84–95.

[33] O. Pereira, J.-J. Quisquater, Une analyse de sécurité des suites de protocoles cliques, dans : Actes de la 14e conférence IEEE sur la sécurité informatique Atelier sur les fondements (CSFW-14), 2001, pp. 73–81.

[34] M. Rusinowitch, M. Turuani, L'insécurité du protocole avec un nombre fini de sessions est NP-complète, dans : 14e IEEE Computer Security Atelier sur les fondations (CSFW-14), IEEE Computer Society, 2001, pp. 174–190.

[35] H. Seidl, La surcharge Haskell est DEXPTIME-complète, Information Processing Letters 52 (2) (1994.)

[36] V. Shmatikov, Analyse décidable des protocoles cryptographiques avec produits et exponentiation modulaire, dans : D. Schmidt (éd.), 13e Symposium européen sur la programmation (ESOP 2004), Lecture Notes in Computer Science, vol. 2986, Springer, Berlin, 2004, pp. 355–369.

[37] T. Truderung, Protocoles réguliers et attaques avec connaissances régulières, dans : R. Nieuwenhuis (éd.), Actes de la 20e Conférence internationale sur la déduction automatique (CADE 2005), Lecture Notes in Computer Science, vol. 3328, Springer-Verlag, 2005, pp. 377–391.

[38] T. Truderung, Sélection de théories et de protocoles récursifs, dans : M. Abadi, L. de Alfaro (éd.), Actes de la 16e Conférence internationale sur la théorie de la concurrence (CONCUR 2005), Lecture Notes in Computer Science, vol. 3653, Springer-Verlag, 2005, pp. 217–232.

[39] J. Zhou, Correction d'une faille de sécurité dans les protocoles IKE, Electronic Letter 35 (13) (1999) 1072–1073.