Réduction de graphes pour l'analyse de protocoles de routage sécurisés

Degrieck Jan
Encadrantes : Cortier Véronique, Delaune Stéphanie
Équipe Cassis, Loria - Équipe Secsi, LSV/ENS de Cachan

14 mars 2011 - 26 août 2011

Le contexte général

Les réseaux ad-hoc sont des réseaux organisés pour des agents mobiles et connectés sans fils. La structure de communication de ce type de réseau est construite de façon distribuée : chaque agent du réseau dispose d'un rôle important de routage, de façon autonome. Une de ses application est notamment militaire, où le système de communication doit être construit sans structure hiérarchisée initiale.
On appelle protocoles de routage les protocoles qui permettent de mettre en place cette structure de communication et qui fournissent à tous les agents le pouvoir de communiquer entre eux, en découvrant les noeuds intermédiaires situés entre deux interlocuteurs éloignés. Les protocoles de routage ont initialement été conçus sans notions de sécurité. Or des travaux ont mis en avant que cette hypothèse n'était pas réaliste. Des protocoles de routage sécurisés ont alors vu le jour.
Les protocoles sécurisés sont des protocoles dans lesquels des primitives cryptographiques interviennent, comme le chiffrement symétrique, les signatures ou le chiffrement asymétrique, et qui visent à garantir certaines propriétés comme l'authentification ou le secret.
On peut par exemple citer la technique SRP [19] qui a pour but de sécuriser les protocoles de routage existants en rajoutant une couche de chiffrement entre l'expéditeur et la destination, ou des nouveaux protocoles comme Ariadne [15] conçus avec des ambitions de sécurité.

Le problème étudié

La question abordée dans ce rapport est de pouvoir vérifier de façon formelle les protocoles de routage, et dans un deuxième temps, de pouvoir les étudier de façon automatique.
L'intérêt de l'approche formelle réside dans l'apport d'une transparence des raisonnement vis-à-vis d'un modèle bien défini et donc dans l'apport d'une rigueur et d'une fiabilité qui permet d'être exhaustif quant aux attaques modélisables.
Les protocoles de routage sécurisés ont été conçus dans les dix dernières années. Le récent problème de la vérification ces protocoles de routage sécurisés a déjà été étudié, à la conception ou dans des travaux ultérieurs. Ainsi, il a été montré dans [13] une attaque sur le protocole de routage sécurisé Ariadne. Or les travaux jusqu'à présents sont à notre connaissance spécifiques à une attaque particulière ou spécifiques à un protocole particulier.
D'autre part, la vérification des protocoles classiques de sécurité (par exemple des protocoles d'établissement de clé), n'est pas nouvelle et des outils ont été développés pour tester les propriétés de sécurité de façon automatique, comme Avispa [6] qui est devenu la plate-forme Avantssar et le logiciel Proverif [11]. Cependant, à l'heure actuelle, aucun outil de vérification n'existe pour vérifier les

protocoles de routage sécurisés. On ne peut pas non plus utiliser directement les outils existants : il faut considérer la topologie du réseau, élément indispensable à la modélisation de la propriété de sécurité et des protocoles de routage eux-mêmes.

La contribution proposée

Afin de pouvoir vérifier les protocoles de routage, on a mis en place un résultat de réduction du nombre de topologies du réseau ad-hoc à considérer, suffisant pour capturer tous les comportements possibles du protocole étudié. Ainsi, on a pu utiliser l'outil de vérification ProVerif développé à l'origine pour analyser des protocoles «classiques», pour y vérifier le petit nombre de topologies à tester.

Les arguments en faveur de sa validité

On a réussi à étudier des protocoles de routage comme SRP [19] appliqué à DSR [16] et SDMSR [10] de façon automatique.
A notre connaissance, il s'agit du premier résultat permettant d'analyser automatiquement des protocoles de routage pour des topologies arbitraires.

Le bilan et les perspectives

Dans ce travail, on a réduit le nombre de configurations à tester pour vérifier les protocoles sécurisés de routage à un nombre fini de graphes, représentants la topologie du réseau. De ce fait, on a pu utiliser un outil existant de vérification automatique ProVerif pour tester les protocoles de routage dans le cadre d'un nombre non borné de session.

On n'a étudié qu'un seul type de propriété de sécurité ici, mais notre résultat semble pouvoir s'étendre à toutes propriétés de voisinages. De même, d'autres primitives cryptographiques auraient pu rentrer dans notre formalisme. Dans un modèle adéquat, notre résultat pourrait aussi s'étendre à des protocoles que l'on ne peut pas modéliser actuellement, comme des protocoles avec des tables de routage. Il en est de même pour des tests de voisinages plus complexes, par exemple faisant des tests récursifs, on aurait pu sans trop de difficultés étendre notre résultat théorique pour capturer ce genre de tests.

Malgré les améliorations possibles au niveau théorique, l'application du résultat est limité par la puissance des outils existants. En effet, l'outil Proverif ne prend pas en compte ces mécanismes.

Outre ce problème, les protocoles de routage que l'on a vérifié sont des simplifications des protocoles existants. On ne prend en effet pas en considération les tests de différence. Dans un cadre général, ceci amène de fausses attaques. Cependant, nous pensons que dans les protocoles existants, les conditions sont réunies pour que la simplification n'apporte pas de fausses attaques.

Il semble donc possible d'étendre ce résultat pour tenir compte des tests de différence.

1 Introduction

Les réseaux ad-hoc sont des réseaux organisés pour des agents mobiles et connectés sans fils. La structure de communication de ce type de réseau est construite de façon distribuée : chaque agent du réseau dispose d'un rôle important de routage, de façon autonome. On appelle protocoles de routage les protocoles qui permettent de mettre en place cette structure de communication et qui fournissent à tous les agents le pouvoir de communiquer entre eux, en découvrant les noeuds intermédiaires situés entre deux interlocuteurs éloignés.

Dans un premiers temps, ces protocoles de routage ont été construits et pensés sans la notion de sécurité, dans un contexte où chaque agent se comporte de façon honnête. On trouve parmi ceux-ci par exemple DSR [16] et AODV[20]. Plusieurs travaux récents ont montré qu'un tel contexte n'est pas réaliste et qu'il peut mener à des attaques ([21], [18]). En effet, des agents malhonnêtes peuvent aisément prendre la fonctionnalité de routage à leur avantage, par exemple en aspirant tous les messages, ou en rendant la communication impossible.

C'est pourquoi des protocoles de routage ont ensuite été conçus à l'aide de primitives cryptographiques, comme les fonctions de hachages, les signatures et les chiffrements symétriques et asymétriques. On peut par exemple citer la technique SRP [19] qui a pour but de sécuriser les protocoles existants en rajoutant une couche de chiffrement entre l'expéditeur et la destination, ou des nouveaux protocoles comme Ariadne [15] conçus avec des ambitions de sécurité.

De ce fait, des études ont été faites sur ces protocoles pour vérifier les garanties de sécurité que pouvaient apporter l'utilisation de ces techniques cryptographiques. La majorité de ces études ont été faites sur des protocoles particuliers ou sur des configurations particulières. Par exemple, il a été montré dans [13] une attaque sur le protocole de routage sécurisé Ariadne. Ces études sont naturellement sujettes à des erreurs : toutes les vulnérabilités ne sont pas forcément découvertes.

Dans le cadre de l'étude des protocoles classiques de sécurité (par exemple des protocoles d'établissement de clé), des outils ont été développés pour tester les propriétés de sécurité de façon automatique. On peut notamment citer Avispa [6] qui est devenu la plate-forme Avantssar et le logiciel Proverif [11]. Avantssar a par exemple contribué à découvrir une faille de sécurité dans un protocole utilisé par GoogleAps (Single Sign-On basé sur SAML)[7]. De plus, les outils comme ProVerif tentent avec succès de s'attaquer à la vérification de protocoles de sécurité dans un cadre d'un nombre non-borné de session avec de nombreux résultats. On peut citer les analyses réalisées sur le protocole JFK [3] ou encore sur le système de fichier Plutus [12].

Par contre, la majorité des travaux sur les protocoles de routage ne sont pas faits de façon automatique, car on ne peut pas utiliser directement les outils existants. En effet, il faut prendre en compte la topologie du réseau ad-hoc, essentielle aux protocoles de routage, et qui n'est initialement pas prise en considération dans les outils classiques. Cependant, des efforts ont été faits dans le sens de l'automatisation dans [8], avec l'étude de la décidabilité de ces protocoles dans un cadre d'un nombre borné de session, mais sans donner d'algorithme efficace. On peut malgré tout espérer qu'un outil de vérification automatique puisse être utilisé. Même dans le cas d'un nombre non borné de session, on peut espérer traiter certains protocoles de routage, malgré l'indécidabilité du problème, à l'instar de ProVerif. On peut citer l'utilisation de la plate-forme Avantssar pour vérifier les protocoles ARAN et endairA [9]. Ce travail partage l'idée d'utiliser un outil existant de vérification de protocoles de sécurité aux protocoles de routage pour tester de façon automatique les protocoles. Cependant, les configurations automatiquement vérifiées sont deux topologies arbitrairement choisies, dans lesquels il y a usuellement des attaques. Finalement, dans [5], ils étudient un protocole dans toutes les configurations possibles de graphe, représentant la topologie du réseau, avec un nombre fixé de noeuds, et simplifient par symétrie le nombre de configuration à étudier. Cependant, aucun travail n'a vraiment tenté d'étudier les configurations pour un nombre non fixé

de noeuds, et de sessions.

Nos contributions. Dans un premier temps, pour un protocole de découverte de route, on montre qu'une attaque possible pour un graphe donné est aussi possible dans un graphe dans lequel tous les agents sont connectés, sauf deux. Ces deux noeuds non reliés sont utiles pour l'attaque. Puis, on réduit le nombre d'agents dans un graphe, en projetant sur un même noeud les noeuds qui ont les même propriétés de voisinages. Ainsi, le noeud projeté exécute tous les processus des noeuds initiaux. En assemblant les premières étapes, on obtient un nombre fini de petits graphes qui capture toutes les configurations de graphes possibles. En codant en dur ces graphes dans l'outil de vérification ProVerif, on teste alors deux protocoles de découverte de route, à savoir SRP appliqué à DSR et SDMSR [10] sur le petit ensemble des petites configurations possibles et sur un nombre non borné de sessions de façon automatisée.

2 Modèle des Protocoles

2.1 Messages

Les primitives cryptographiques sont représentées par des symboles de fonctions. On considère une signature $(\mathcal{S}, \mathcal{F})$ constituée d'un ensemble de types $\mathcal{S}$ et d'un ensemble de symboles de fonctions $\mathcal{F}$, avec des arités de la forme $ar(f) = s_1 \times \ldots \times s_k \to s$.

On considère un ensemble infini de variables $\mathcal{X}$ et un ensemble infini de noms $\mathcal{N}$ qui représente typiquement les nonces ou les noms d'agents.

On considère un type spécial loc pour les noms d'agents sur le réseau (ou noeuds).

On suppose que les noms et les variables sont typés. On suppose alors un ensemble infini de noms d'agents $\mathcal{N}_{\mathsf{loc}}$ de type loc. Les termes de type s sont définis par récurrence :

t	::=		terme de type s
	|	x	variable x de type s
	|	a	nom a de type s
	|	$f(t_1, \ldots, t_k)$	application d'une fonction $f \in \mathcal{F}$

où $ar(f) = s_1 \times \ldots \times s_k \to s$ et t_i sont des termes de type s_i.

On considère un type spécial term tel que tout terme est de type term. On écrit $var(t)$ (respectivement $names(t)$) l'ensemble des variables libres (respectivement des noms) présents dans un terme t et $St(t)$ représente l'ensemble des sous-termes syntaxiques de t. Le terme t est clos s'il ne contient pas de variables libres, *i.e.* $var(t) = \emptyset$.

Dans la suite, on va considérer la signature $(\mathcal{S}_1, \mathcal{F}_1)$ définie par $\mathcal{S}_1 = \{\mathsf{loc}, \mathsf{lists}, \mathsf{term}\}$ et $\mathcal{F}_1 = \{\mathsf{hmac}, \langle\rangle, ::, [], \{\_\}\_, \mathsf{priv}, \{|\_|\}\_, [\![\_]\!]\_\}$, avec les arités suivantes :

- $\mathsf{hmac}, \langle\_,\_\rangle, \{\_\}\_, \{|\_|\}\_, [\![\_]\!]\_, \mathsf{shk} : \mathsf{term} \times \mathsf{term} \to \mathsf{term}$,
- $:: : \mathsf{loc} \times \mathsf{lists} \to \mathsf{lists}$,
- $[] :\to \mathsf{lists}$,
- $\mathsf{priv} : \mathsf{term} \to \mathsf{term}$.

Le type lists représente des listes de termes de types loc. On considère qu'il n'existe pas de noms de type lists. Le symbole :: est le constructeur de liste. [] est la constante représentant la liste vide. Le terme $\mathsf{hmac}(m, k)$ représente le hmac d'un message m avec une clé k. Le symbole $\langle\rangle$ correspond à l'opérateur de couple. Les termes $\{m\}_k$ et $\{|m|\}_k$ représentent respectivement le message m chiffré avec une clé symétrique, respectivement asymétrique k. Le terme $[\![m]\!]_k$ représente le message m signé avec une clé k. Le terme $\mathsf{shk}(a, b)$ représente une clé partagée entre a et b. On a que $\mathsf{shk}(a, b) = \mathsf{shk}(b, a)$. Le terme $\mathsf{priv}(a)$ représente la clé privée de l'agent a. Pour simplifier,

$$\frac{T \vdash a \quad T \vdash l}{T \vdash a :: l} \qquad \frac{T \vdash u \quad T \vdash v}{T \vdash \langle u, v \rangle} \qquad \frac{T \vdash u \quad T \vdash v}{T \vdash \{u\}_v} \qquad \frac{T \vdash \{u\}_v \quad T \vdash v}{T \vdash u}$$

$$\frac{T \vdash a :: l}{T \vdash a} \qquad \frac{T \vdash \langle u, v \rangle}{T \vdash u} \qquad \frac{T \vdash u \quad T \vdash v}{T \vdash \{\!|u|\!\}_v} \qquad \frac{T \vdash \{\!|u|\!\}_v \quad T \vdash \mathsf{priv}(v)}{T \vdash u}$$

$$\frac{T \vdash a :: l}{T \vdash l} \qquad \frac{T \vdash \langle u, v \rangle}{T \vdash v} \qquad \frac{T \vdash u \quad T \vdash v}{T \vdash [\![u]\!]_v} \qquad \frac{T \vdash [\![u]\!]_v}{T \vdash u}\ (optionnel)$$

$$\frac{T \vdash u \quad T \vdash v}{T \vdash \mathsf{hmac}(u, v)} \qquad \frac{u \in T \cup \{[]\}}{T \vdash u}$$

FIGURE 1 – Système de déduction associé à la signature $(\mathcal{S}_1, \mathcal{F}_1)$.

on confond les noms des agents avec leur clé publique. On note $\langle t_1, t_2, t_3 \rangle$ le terme $\langle t_1, \langle t_2, t_3 \rangle \rangle$, et $[t_1; t_2; t_3]$ pour $t_1 :: (t_2 :: (t_3 :: []))$.

Les substitutions sont notées $\sigma = \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$ où $dom(\sigma) = \{x_1, \ldots, x_n\}$. On ne considère que les substitutions bien typées, c'est à dire telles que x_i et t_i sont de même type. Une substitution σ est close si tous les t_i sont clos. L'application d'une substitution σ à un terme t est notée $\sigma(t)$ ou $t\sigma$. L'unificateur le plus général de deux termes t et u est une substitution que l'on note par $\mathsf{mgu}(t, u)$. On considère que $\mathsf{mgu}(t, u) = \bot$ si t et u ne sont pas unifiables.

La capacité des attaquants est modélisée par une relation de déduction $\vdash \subseteq 2^{\mathsf{term}} \times \mathsf{term}$. La relation $T \vdash t$ représente le fait qu'un terme t est calculable d'un ensemble de termes T. La relation de déduction peut être arbitraire dans le modèle, et n'est donc pas spécifiée. On la définit en pratique par un système de déduction comme dans la figure 1.

On considère la signature $(\mathcal{S}_1, \mathcal{F}_1)$ définie précédemment, le système de déduction présenté dans la figure 1 rend compte de la capacité de l'attaquant de composer les messages avec les constructeurs et des destructeurs classiques, et le déchiffrement s'il connaît la clé. On considère aussi une règle optionnelle

$$\frac{T \vdash [\![u]\!]_v}{T \vdash u}$$

qui exprime la capacité de récupérer le message de sa signature. Ceci est possible suivant le schéma de signature considéré.

2.2 Algèbre de Processus

Plusieurs algèbres existent pour modéliser les protocoles de sécurité, par exemple, dans [2, 1]. Cependant, modéliser les protocoles de routage nécessitent quelques caractéristiques supplémentaires, par exemple, représenter les listes de noms d'agent. On doit aussi prendre en compte la topologie du réseau sous-jacent. L'algèbre de processus est inspirée de CBS# [18] et plus directement de celle présentée dans [8], dont on reprend en grande partie le formalisme. On a fait le choix de simplifier l'algèbre de [8] pour ne présenter que le fragment sur lequel notre résultat de réduction de la section 3 s'applique. L'algèbre de processus que l'on considère permet aux noeuds d'effectuer des tests concernant la validité des routes qu'elle reçoivent, notamment les tests concernant les propriétés de voisinage, comme c'est le cas dans le contexte des protocoles de routage sécurisés.

Le comportement prétendu de chaque noeud peut être modélisé par un processus défini par la grammaire donnée dans la figure 2. Cette algèbre dépend d'un ensemble $\mathcal{L}$ de formules. Le processus

$P, Q :=$	processus
0	processus nul
$\mathsf{out}(u).P$	émission
$\mathsf{in}\ u[\Phi].P$	réception, $\Phi \in \mathcal{L}$
$\mathsf{if}\ \Phi\ \mathsf{then}\ P$	conditionnelle, $\Phi \in \mathcal{L}$
$P \mid Q$	composition parallèle
$!P$	réplication
$\mathsf{new}\ \ m.P$	génération d'un nom frais

FIGURE 2 – Processus

$\mathsf{out}(u).P$ émet u puis se comporte comme P. Le processus $\mathsf{in}\ u[\Phi].P$ attend un message m de la forme u tel que la formule Φ est vrai puis se comporte comme $P\sigma$ où σ est tel que $m = u\sigma$. Si Φ est la formule «vrai», alors on peut simplement écrire $\mathsf{in}\ u.P$. On note $fv(P)$ l'ensemble des variables libres de P. Un processus P est clos quand $fv(P) = \emptyset$.

Les protocoles de routage sécurisés nécessitent certains tests sur les messages qu'ils reçoivent. On va usuellement considérer une logique qui exprime les tests que les noeuds sont capables d'effectuer : $\mathcal{L}_{noeud}$ définie par la grammaire dans la figure 3. On considérera que les agents qui exécutent le protocole de découverte de route que l'on étudie, utilisent un protocole de découverte des voisins [22]. Les agents peuvent ainsi connaître leurs voisins. On abstrait cette connaissance, et on définit des tests que peuvent effectuer ces agents à partir de ce protocole de découverte de route. Ainsi, $\mathsf{check}(a, b)$ représente le fait que deux noeuds a et b sont voisins, ou que a et b sont égaux. $\mathsf{checkl}(c, l)$ est vrai si l est une route possible du point de vue de c, i.e. le prédécesseur et le successeur de c dans la liste sont des voisins de c. Le prédicat $\mathsf{alone}(c, l)$ rend compte du fait que c apparaisse exactement une fois dans la liste l. Enfin, le prédicat $\mathsf{loop}(l)$ rend compte de l'existence d'une boucle dans la liste l. On peut noter que la logique permet de simuler des tests «if Φ then P else Q» quand $\Phi \in \mathcal{L}_{noeud}$ et $\neg\Phi \in \mathcal{L}_{noeud}$, avec if Φ then $P \mid$ if $\neg\Phi$ then Q. Notons tout de même que la logique n'est pas close pas négation. On ne considère pas ici des protocoles qui puissent faire des tests de non-voisinages. Ceci n'est pas une restriction. Dans les protocoles de découverte de route existants, ce genre de tests ne sont jamais effectués. Cette hypothèse est importante dans la démonstration du théorème de complétion 1.

$\Phi :=$	formula
$\mathsf{check}(a, b)$	test de voisinage de deux noeuds
$\mathsf{checkl}(c, l)$	test local de voisinage dans une liste
$\mathsf{alone}(c, l)$	c apparaît dans l exactement une fois
$\neg\mathsf{alone}(c, l)$	c n'est seul dans la liste, ou n'existe pas
$\mathsf{loop}(l)$	existence d'une boucle dans la liste
$\neg\mathsf{loop}(l)$	tous les éléments de la liste sont différents
$\Phi_1 \wedge \Phi_2$	conjonction
$\Phi_1 \vee \Phi_2$	disjonction

FIGURE 3 – Logique $\mathcal{L}_{noeud}$

Étant donné un graphe non orienté $G = (N, E)$, avec $N \subset \mathcal{N}_{\mathsf{loc}}$ un ensemble fini de noms

d'agents, $E \subseteq N \times N$ un ensemble symétrique et réflexif de couples, appelés arêtes, la sémantique de $[\![\Phi]\!]_G$ d'une formule de $\Phi \in \mathcal{L}_{noeud}$ est définie comme suit :

- $[\![\mathsf{check}(a,b)]\!]_G = 1$ si et seulement si $(a,b) \in E$.
- $[\![\mathsf{checkl}(c,l)]\!]_G = 1$ si et seulement si l est de type lists, pour toute sous liste l' de l, on a
  - si $l' = a :: c :: l_1$, alors $(a,c) \in E$.
  - si $l' = c :: b :: l_1$, alors $(c,b) \in E$.
- $[\![\mathsf{loop}(l)]\!]_G = 1$ si et seulement si l est de type lists et il existe un élément apparaissant au moins deux fois dans la liste l.
- $[\![\mathsf{alone}(c,l)]\!]_G = 1$ si et seulement si l est de type lists et c apparaît exactement une fois dans la liste l.
- $[\![\Phi_1 \wedge \Phi_2]\!]_G = [\![\Phi_1]\!]_G \wedge [\![\Phi_2]\!]_G$.
- $[\![\Phi_1 \vee \Phi_2]\!]_G = [\![\Phi_1]\!]_G \vee [\![\Phi_2]\!]_G$.
- $[\![\neg\Phi]\!]_G = \neg[\![\Phi]\!]_G$.

2.3 Exemple : modélisation du protocole SRP appliqué à DSR

On considère le protocole de routage sécurisé SRP introduit dans [19]. On considère que les noeuds effectuent en parallèle un protocole de découverte de voisins. SRP n'est pas un protocole de routage en lui même, mais il décrit une manière générale pour sécuriser certains types de protocoles de routage. On modélise ici une application au protocole DSR [16]. DSR est un protocole utilisé quand un agent S (la source) veut communiquer avec un autre agent D (le destinataire), qui n'est pas son voisin immédiat. Dans un réseau ad-hoc, les messages ne peuvent pas toujours être envoyés au destinataire directement, il est parfois nécessaire de passer par un assez grand nombre de noeuds.

Pour découvrir une route vers la destination, la source construit un message de requête et le diffuse à tous ses voisins. La requête contient son propre nom S, le nom du destinataire D, un identifiant de la requête id, une liste contenant le début de la route vers D, et un hmac calculé sur le contenu de la requête avec une clé partagée $\mathsf{shk}(S,D)$ entre S et D. La source attend alors une réponse contenant la route pour aller jusqu'à D avec un hmac confirmant cette route, et vérifie ensuite s'il s'agit d'une route plausible en vérifiant que la route ne contient pas de boucle et que son voisin dans la liste est bel et bien un réel voisin dans le réseau.

On considère toujours la signature $(\mathcal{S}_1, \mathcal{F}_1)$. De plus, soient $S, D \in \mathcal{N}_{\mathsf{loc}}$ des noms d'agents, $\mathsf{req}, \mathsf{rep}$ des termes constants, id un nonce, $\mathsf{shk}(S,D)$ la clé partagée entre S et D et x_L une variable de type lists. Le processus exécuté par le noeud source S initiant la découverte de route vers un noeud destinataire D est $P_i(S,D) = \mathsf{new}\ id.\mathsf{out}(u_1).\mathsf{in}\ u_2[\Phi_S].0$ où :

$$u_1 = \langle \mathsf{req}, S, D, id, S :: [], \mathsf{hmac}(\langle \mathsf{req}, S, D, id\rangle, \mathsf{shk}(S,D))\rangle$$
$$u_2 = \langle \mathsf{rep}, D, S, id, x_L, \mathsf{hmac}(\langle \mathsf{rep}, D, S, id, x_L\rangle, \mathsf{shk}(S,D))\rangle$$
$$\Phi_S = \mathsf{checkl}(S, x_L) \wedge \neg\mathsf{loop}(x_L) \wedge \mathsf{alone}(S, x_L).$$

Les noms des noeuds intermédiaires sont alors accumulés dans la route du message de requête. Dans la modélisation que l'on fait de ce protocole SRP appliqué à DSR, du fait que l'on ne considère pas de mémoire, les noeuds intermédiaires n'ignorent pas les doublons et les propagent sur le réseau. En terme de vérification, on n'affaiblit pas la propriété demandée, car les doublons ne permettent pas d'avoir de nouveaux comportements. Un noeud intermédiaire vérifie aussi que la requête reçue est localement correcte, en vérifiant que le nom du noeud en tête de la liste représentant la route est bien un de ses voisins direct dans le réseau. Dans la suite, $V \in \mathcal{N}_{\mathsf{loc}}$, x_S, x_D et x_a sont des variables de type loc alors que x_r est une variable de type lists et x_{id}, x_m sont des variables de type term. Le

processus exécuté par un noeud intermédiaire V en transférant une requête est alors :

$$P_{\mathsf{req}}(V) = \mathsf{in}\ w_1[\Phi_V].\mathsf{out}(w_2)$$

$$\text{où} \quad \begin{cases} w_1 = \langle \mathsf{req}, x_S, x_D, x_{id}, x_a :: x_r, x_m \rangle \\ \Phi_V = \mathsf{check}(V, x_a) \\ w_2 = \langle \mathsf{req}, x_S, x_D, x_{id}, V :: (x_a :: x_r), x_m \rangle \end{cases}$$

Puis la requête atteint le destinataire D, il vérifie que la requête contient un hmac valide et que le nom du noeud en tête de la liste représentant la route est bien un de ses voisins dans le réseau. Ensuite, le destinataire D construit un message de réponse. Pour ce faire, il calcule un nouveau hmac avec la même clé partagée $\mathsf{shk}(S, D)$ avec l'expéditeur sur la liste qui représente la route, et la renvoie sur le réseau. Le processus exécuté par le destinataire D est $P_{\mathsf{dest}}(D, S) = \mathsf{in}\ v_1[\Phi_D].\mathsf{out}(v_2).0$ où :

$$\begin{aligned} v_1 &= \langle \mathsf{req}, S, D, x_{id}, x_a :: x_L, \mathsf{hmac}(\langle \mathsf{req}, S, D, x_{id} \rangle, \mathsf{shk}(S, D)) \rangle \\ \Phi_D &= \mathsf{check}(D, x_a) \\ v_2 &= \langle \mathsf{rep}, D, S, x_{id}, x_a :: x_L, \mathsf{hmac}(\langle \mathsf{rep}, D, S, x_{id}, x_a :: x_L \rangle, \mathsf{shk}(S, D)) \rangle \end{aligned}$$

Puis, la réponse retraverse le réseau vers l'initiateur du protocole S. Les noeuds intermédiaires vérifient que la route dans la liste est localement correcte (i.e. ils vérifient que leur nom n'apparaît qu'une fois dans la liste et que les voisins dans la liste correspondent à des voisins directs dans le réseau) avant de le transférer. Le processus d'un noeud intermédiaire V en transférant une réponse est alors :

$$P_{\mathsf{rep}}(V) = \mathsf{in}\ w'[\Phi'_V].\mathsf{out}(w').0$$

$$\text{où} \quad \begin{cases} w' = \langle \mathsf{rep}, x_D, x_S, x_{id}, x_r, x_m \rangle \\ \Phi'_V = \mathsf{checkl}(V, x_r) \wedge \mathsf{alone}(V, x_r) \end{cases}$$

2.4 Modèle d'exécution

Chaque processus est localisé dans un noeud spécifique sur le réseau. Contrairement au modèle classique de Dolev-Yao, les attaquants n'ont pas un contrôle sur l'ensemble du réseau, mais ne peuvent interagir qu'avec leurs voisins directs. Plus précisément, on suppose que la topologie du réseau est représentée par un graphe non dirigé $G = (N, E)$ avec $N \subset \mathcal{N}_{\mathsf{loc}}$ un ensemble fini, et $E \subseteq N \times N$ un ensemble réflexif et symétrique d'arêtes. On suppose aussi que l'on a un ensemble de noeuds dit malicieux $\mathcal{M} \subseteq N$ qui sont contrôlés par l'attaquant. On suppose que ces noeuds malicieux partagent leurs connaissances. Ce modèle n'est pas restreint à un seul noeud malicieux. Cependant, il est connu que des configurations avec plus de deux noeuds malicieux avec une mémoire partagée entraîne certaines fois des attaques triviales appelées «wormhole» [14, 17].

Une configuration concrète (close) d'un réseau est un couple $(\mathcal{P}; \mathcal{I})$ où :

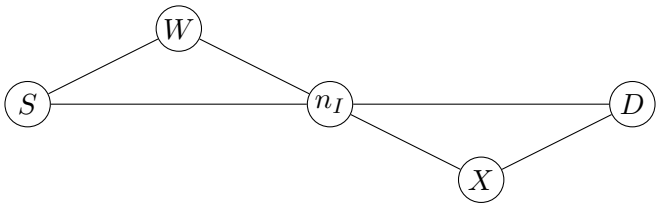
- $\mathcal{P}$ est un ensemble d'expressions de la forme $\lfloor P \rfloor_n$ où les processus nuls, i.e. les expressions de la forme $\lfloor 0 \rfloor_n$ sont supprimés. $\lfloor P \rfloor_n$ représente le processus (clos) P situé au noeud $n \in N$. On pourra noter $\lfloor P \rfloor_n \cup \mathcal{P}$ à la place de $\{\lfloor P \rfloor_n\} \cup \mathcal{P}$.
- $\mathcal{I}$ est l'ensemble de termes (clos) représentant les messages que l'attaquant a pu intercepter.

Exemple 1. *En continuant la modélisation de SRP, une configuration initiale possible pour le protocole est*

$$K_0 = (\lfloor P_i(S, D) \rfloor_S \mid \lfloor P_{\mathsf{dest}}(D, S) \rfloor_D; \mathcal{I}_0)$$

où à la fois le noeud source S et le noeud destination D veulent communiquer. Une topologie plus réaliste inclurait encore plus des noeuds intermédiaires, mais celle-ci suffit déjà pour monter une

attaque, découverte dans [13]. On suppose que la connaissance de l'attaquant est donnée par $\mathcal{I}_0$. Une topologie possible est modélisée par le graphe $G_0 = (N, E_0)$ ci dessous. On suppose qu'il n'y a qu'un seul noeud malicieux : $\mathcal{M}_0 = \{n_I\}$. Le noeuds W et X sont deux autres noeuds honnêtes. On a donc $N = \{n_I, X, W, D, S\}$. Il n'est pas nécessaire que ces noeuds exécutent le protocole.



Le système de communication est basé sur des canaux de type «lossy». On pourra en effet considérer que les messages soient perdus sur le réseau. Seulement les voisins peuvent recevoir un message diffusé par un agent, mais il est possible qu'ils ne le reçoivent pas. Le système de communication est formellement défini par les règles présentées dans la figure 4. Elles sont paramétrées par le graphe sous-jacent G et l'ensemble des noeuds malicieux $\mathcal{M}$. La règle Comm est la règle qui symbolise les canaux «lossy». La règle In rend compte du pouvoir de déduction de l'attaquant. L'attaquant peut envoyer à des voisins choisis des données qu'il a pu déduire de ses connaissances, avec le système de déduction présenté en figure 1. Les autres règles représentent l'exécution d'un processus par un noeud en particulier.

$$\textsc{Comm} \quad (\{\lfloor \mathsf{in}\ u_j[\Phi_j].P_j \rfloor_{n_j} \mid \mathsf{mgu}(t,u_j) \neq \bot, [\![\Phi_j\sigma_j]\!]_G = 1, (n,n_j) \in E\} \cup \lfloor \mathsf{out}(t).P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \to_{G,\mathcal{M}} (\{\lfloor P_j\sigma_j \rfloor_{n_j}\} \cup \lfloor P \rfloor_n \cup \mathcal{P}; \mathcal{I}')$$

où $\sigma_j = \mathsf{mgu}(t,u_j)$, $\mathcal{I}' = \mathcal{I} \cup \{t\}$ si $(n,n_I) \in E$ pour un $n_I \in \mathcal{M}$ et $\mathcal{I}' = \mathcal{I}$ sinon.

$$\textsc{In} \quad (\lfloor \mathsf{in}\ u[\Phi].P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \to_{G,\mathcal{M}} (\lfloor P\sigma \rfloor_n \cup \mathcal{P}; \mathcal{I})$$
si $(n_I,n) \in E$ pour un $n_I \in \mathcal{M}$, $\mathcal{I} \vdash t$, $\sigma = \mathsf{mgu}(t,u)$ et $[\![\Phi\sigma]\!]_G = 1$

$$\textsc{If-Then} \quad (\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \to_{G,\mathcal{M}} (\lfloor P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \quad \text{si } [\![\Phi]\!]_G = 1$$

$$\textsc{Par} \quad (\lfloor P_1 \mid P_2 \rfloor_n \cup \mathcal{P}; \mathcal{I}) \to_{G,\mathcal{M}} (\lfloor P_1 \rfloor_n \cup \lfloor P_2 \rfloor_n \cup \mathcal{P}; \mathcal{I})$$

$$\textsc{Repl} \quad (\lfloor !P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \to_{G,\mathcal{M}} (\lfloor P\alpha \rfloor_n \cup \lfloor !P \rfloor_n \cup \mathcal{P}; \mathcal{I})$$
où α est le renommage des variables liées de P

$$\textsc{New} \quad (\lfloor \mathsf{new}\ m.P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \to_{G,\mathcal{M}} (\lfloor P\{^{m'}/_m\} \rfloor_n \cup \mathcal{P}; \mathcal{I})$$
où m' est un nom frais

Figure 4 – Système de transition concret

Dans la règle Comm, un noeud n veut envoyer un message sur le réseau. Les noeuds qui vont recevoir ce message sont nécessairement voisins de n et sont en attente d'un message de la bonne

forme. De plus, si n est un voisin d'un noeud malicieux, la connaissance de l'attaquant augmente du message envoyé.

La relation $\rightarrow^*_{G,\mathcal{M}}$ est la clôture réflexive et transitive de la relation $\rightarrow_{G,\mathcal{M}}$. On pourra noter $\rightarrow_{\mathcal{M}}$, $\rightarrow_G$, $\rightarrow$ à la place de $\rightarrow_{G,\mathcal{M}}$ quand le graphe sous-jacent G ou l'ensemble des noeuds malicieux $\mathcal{M}$ est clair dans le contexte.

On peut remarquer que dans le cas où chaque noeud honnête est relié à au moins un attaquant, on récupère le modèle dans lequel l'attaquant est censé contrôler toutes les communications.

Exemple 2. *Continuant l'exemple développé dans la section 2.3, la séquence de transitions suivante est possible depuis la configuration initiale K_0 :*

$$K_0 \rightarrow^*_{G_0,\mathcal{M}_0} (\lfloor \mathsf{in}\ u_2[\Phi_S].0 \rfloor_S \cup \lfloor P_{\mathsf{dest}}(D,S) \rfloor_D ; \mathcal{I}_0 \cup \{u_1\})$$

où u_1, u_2, Φ_S ont déjà été définies dans la section précédente :

$$\begin{array}{l} u_1 = \langle \mathsf{req}, S, D, id, S :: [], \mathsf{hmac}(\langle \mathsf{req}, S, D, id \rangle, \mathsf{shk}(S,D)) \rangle \\ u_2 = \langle \mathsf{rep}, D, S, id, x_L, \mathsf{hmac}(\langle \mathsf{rep}, D, S, id, x_L \rangle, \mathsf{shk}(S,D)) \rangle \\ \Phi_S = \mathsf{checkl}(S, x_L) \wedge \neg \mathsf{loop}(x_L) \end{array}$$

Durant cette transition, S diffuse à ses voisins une requête afin de trouver une route pour D. L'attaquant n_I est un voisin de S dans G_0 (cf. l'exemple 1), donc il apprend le message de requête. En supposant que l'attaquant connaisse les noms de ses voisins, i.e. $W, X \in \mathcal{I}_0$, il peut alors construire le message suivant :

$$m = \langle \mathsf{req}, S, D, id, [X; W; S], \mathsf{hmac}(\langle \mathsf{req}, S, D, id \rangle, \mathsf{shk}(S,D)) \rangle$$

et l'envoyer à D. Étant donné que $(X, D) \in E_0$, le noeud D accepte le message et la configuration qui en résulte est : $(\lfloor \mathsf{in}\ u_2[\Phi_S].0 \rfloor_S \cup \lfloor \mathsf{out}(v_2\sigma).0 \rfloor_D ; \mathcal{I}_0 \cup \{u_1\})$ *où*

$$\begin{cases} v_2 = \langle \mathsf{rep}, D, S, x_{id}, x_a :: x_L, \mathsf{hmac}(\langle D, S, x_{id}, x_a :: x_L \rangle, \mathsf{shk}(S,D)) \rangle \\ \sigma = \{ {}^{id}/_{x_{id}}, {}^{X}/_{x_a}, {}^{[W;S]}/_{x_L} \} \end{cases}$$

Définition 1 (Protocole de découverte de route)**.** *On définit un protocole de découverte de route par un ensemble $\mathcal{Q}_{route}$ de processus paramétrés sans réplication ni composition parallèle. Chaque processus paramétré a au moins un paramètre et les paramètres sont nécessairement de type* loc *et on peut considérer sans perte de généralité que le premier paramètre n_1 est le nom de l'agent qui va exécuter le processus. De plus, pour tout processus paramétré $P \in \mathcal{Q}_{route}$ de paramètres $n_1, ..., n_k$, on a :*

- *$P(n_1, ..., n_k)$ comprend le terme $\mathsf{priv}(m) \Rightarrow m = n_1$*
- *$P(n_1, ..., n_k)$ comprend le terme $\mathsf{shk}(m,p) \Rightarrow m = n_1$ ou $p = n_1$*
- *$P(n_1, ..., n_k)$ comprend le terme $\mathsf{check}(m,p)$ ou $\mathsf{checkl}(m,l)$ dans un test $\Rightarrow m = n_1$ ou $p = n_1$*
- *P ne contient pas de terme clos de type $\mathcal{N}_{\mathsf{loc}}$.*

De plus, il existe un processus de deux paramètres $P_i(n_1, n_2)$ appelé initiateur comprenant une variable x_L de type lists*.*

Les conditions listées dans la définition ci-dessus ne sont pas des restrictions. La quatrième condition est classique dans un protocole de découverte de route réel, à savoir qu'il ne dépend pas d'un nom d'agent codé en dur. Les autres conditions représentent le fait qu'un agent ne peut pas disposer des secrets des autres agents.

On note que les processus que l'on considère ici sont des processus simples dans le sens qu'il n'y a pas de branches else, ni de composition parallèle et de réplication. On ne considère pas l'opération de réplication ni de branches else, et ainsi, on peut parler de dernière instruction dans un processus. Dans ce cas, on notera un processus $P = Q.R$ le processus Q auquel on a substitué la dernière instruction 0. par le processus R.

Ne pas considérer de réplication n'est pas restrictif car l'étude de ces protocoles se fera dans un nombre non borné de sessions et tous les processus paramétrés seront répliqué dans les configurations étudiées.

Dans le cadre d'une session d'un protocole de découverte de route, le protocole initiateur $P_i(n_1, n_2)$ est en pratique exécuté par l'agent n_1 qui souhaite découvrir une route vers un destinataire n_2. La variable x_L contient alors en fin d'exécution du processus $P_i(n_1, n_2)$ la liste des noms des noeuds permettant d'aller de n_1 à n_2.

Dans un objectif de vérification des protocoles de découverte de route, on peut définir une attaque en exigeant une propriété sur la liste x_L d'un des processus initiateurs. Ces propriétés peuvent être plus ou moins fortes. On peut demander que chaque paire de voisins dans la liste soit effectivement une paire de voisins dans la topologie du réseau. Cette condition est trop forte si on considère plusieurs agents malhonnêtes non voisins. En effet, ces deux noeuds peuvent annoncer au reste du réseau qu'ils sont voisins, car ils partagent leur mémoire dans notre modèle.

On va donc se concentrer sur une propriété plus faible que appelons way. La sémantique de way est :
$[\![\mathsf{way}(l)]\!]_G = 1$ si et seulement si l est de type lists et $l = [a_1; \ldots; a_n]$, pour chaque $1 \leq i < n$, $a_i \notin \mathcal{M} \vee a_{i+1} \notin \mathcal{M} \Rightarrow (a_i, a_{i+1}) \in E$

L'idée derrière la propriété way est que si un noeud dans la liste correspondant à une route potentielle est honnête, alors ses voisins dans la liste sont des voisins réels dans le graphe sous-jacent.

Il semble qu'il ne soit pas possible de demander une propriété plus forte que way car deux noeuds malhonnêtes peuvent toujours se faire passer des messages par des canaux auxiliaires (attaque de type «wormhole»), et aucun des autres noeuds honnêtes n'est capable de déduire que deux noeuds ne sont pas voisins. Par contre, si un des deux noeuds voisins dans la liste est honnête, la justification qu'ils soient effectivement voisins peut manquer, et les autres noeuds du réseau peuvent potentiellement déduire qu'il n'y a pas de lien entre ces noeuds.

Définition 2 ($\mathcal{Q}_{route}$-configuration)**.** *Soit un graphe $G = (N, E)$ et un ensemble de noeuds malicieux $\mathcal{M} \in N$. On considère un protocole de découverte de route $\mathcal{Q}_{route}$, de processus paramétré initiateur P_i. Soit deux noeuds honnêtes dans le graphe G : $a \in N$ et $b \in N$ tels que $a \notin \mathcal{M}$ et $b \notin \mathcal{M}$. On considère la propriété* $\mathsf{way}(x_L)$.

Une $\mathcal{Q}_{route}$-configuration pour G, $\mathcal{M}$, a, b et $\mathsf{way}(x_L)$ *est une configuration concrète $(\mathcal{P}; \mathcal{I})$ telle que :*

- $\mathcal{P} = \lfloor P_i(a, b).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error}) \rfloor_a \cup \mathcal{P}'$ *où* error *n'apparaît nul par ailleurs dans la configuration*
- $\forall \lfloor P \rfloor_n \in \mathcal{P}' \Rightarrow (\exists a_2, ..., a_k \in N, P' \in \mathcal{Q}_{route}$ *tel que* $P = !P'(n, a_2, ..., a_k))$
- $\forall a_1, ..., a_k \in N, P' \in \mathcal{Q}_{route} \Rightarrow \lfloor !P'(a_1, ..., a_k) \rfloor_{a_1} \in \mathcal{P}'$

De plus, on a aussi $\mathcal{I} = N \cup \{\mathsf{shk}(n, p) \mid n \in \mathcal{M} \vee p \in \mathcal{M}\} \cup \{\mathsf{priv}(n) \mid n \in \mathcal{M}\}$

On remarque que la notation $P_i(a, b).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})$ est légitime car les processus $P_i(a, b)$ et $\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})$ sont des processus sans réplication et sans branche else.

On remarque aussi le symbole de réplication dans la configuration. L'idée derrière cette définition est que l'on considère l'ensemble des configurations dans laquelle pourrait être un réseau. On prend

alors en considération un nombre non borné de sessions du protocole. Le cas non borné de session capture alors tous les comportements que pourrait avoir un protocole. Les seuls paramètres ici sont les noeuds initiateur et destinataire du protocole de découverte de route.

Comme souvent, une attaque peut être vue comme une propriété d'atteignabilité, à partir d'une configuration initiale.

Définition 3 ($\mathcal{M}$-attaque)**.** *Soit $G = (N, E)$ un graphe et $\mathcal{M}$ un ensemble de noeuds malicieux. On considère une $\mathcal{Q}_{route}$-configuration pour G, $\mathcal{M}$, a et b honnêtes et la propriété* $\mathsf{way}(x_L)$*. On dit qu'il y a une $\mathcal{M}$-attaque sur cette $\mathcal{Q}_{route}$-configuration s'il existe $\mathcal{P}', \mathcal{I}'$ tels que :*

$$(\lfloor P_i(a,b).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\rfloor_a \cup \mathcal{P}; \mathcal{I}) \ \rightarrow^*_{G,\mathcal{M}}\ (\lfloor \mathsf{out}(\mathsf{error})\rfloor_a \cup \mathcal{P}', \mathcal{I}')$$

autrement dit, que le processus initiateur accepte comme route valide une liste représentée par la variable x_L qui ne satisfait pas la propriété way*.*

De plus, on dit qu'il existe une attaque sur un protocole de découverte de route $\mathcal{Q}_{route}$ pour une propriété $\mathsf{way}(x_L)$ *s'il existe un graphe G, un ensemble de noeuds malicieux $\mathcal{M}$, a et b honnêtes tels qu'il y ait une $\mathcal{M}$-attaque la $\mathcal{Q}_{route}$-configuration correspondante.*

Exemple 3. *On continue l'exemple SRP. On considère que S veut initier le protocole SRP avec D, dans le graphe décrit précédemment. La propriété que l'on veut garantir à la liste x_L instanciée pendant l'exécution du protocole est* $\neg\mathsf{way}(x_L)$*. De ce fait, soit $P'_i(S, D)$ le processus en résultant.*

$$P'_i(S,D) = \mathsf{new}\ id.\mathsf{out}(u_1).\mathsf{in}\ u_2[\Phi_S].P$$

où $P = \mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})$*.*

On doit donc considérer la configuration initiale $K_1 = (\mathcal{P}; \mathcal{I}_1)$. avec :

$\mathcal{P}$ =

$$\begin{array}{l}
\lfloor P_i(S,D).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\rfloor_S \cup \\
(\lfloor !P_{\mathsf{req}}(W)\rfloor_W \cup \lfloor !P_{\mathsf{rep}}(W)\rfloor_W \cup \bigcup_{n\in N}\lfloor !P_i(W,n)\rfloor_W \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(W,n)\rfloor_W \cup \\
(\lfloor !P_{\mathsf{req}}(X)\rfloor_X \cup \lfloor !P_{\mathsf{rep}}(X)\rfloor_X \cup \bigcup_{n\in N}\lfloor !P_i(X,n)\rfloor_X \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(X,n)\rfloor_X \cup \\
(\lfloor !P_{\mathsf{req}}(S)\rfloor_S \cup \lfloor !P_{\mathsf{rep}}(S)\rfloor_S \cup \bigcup_{n\in N}\lfloor !P_i(S,n)\rfloor_S \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(S,n)\rfloor_S \cup \\
(\lfloor !P_{\mathsf{req}}(n_I)\rfloor_{n_I} \cup \lfloor !P_{\mathsf{rep}}(n_I)\rfloor_{n_I} \cup \bigcup_{n\in N}\lfloor !P_i(n_I,n)\rfloor_{n_I} \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(n_I,n)\rfloor_{n_I} \\
(\lfloor !P_{\mathsf{req}}(D)\rfloor_D \cup \lfloor !P_{\mathsf{rep}}(D)\rfloor_D \cup \bigcup_{n\in N}\lfloor !P_i(D,n)\rfloor_D \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(D,n)\rfloor_D \cup
\end{array}$$

où les processus paramétrés du protocole de découverte de route sont les même que dans la section 2.3.

De plus, il faut considérer dans la connaissance de l'attaquant les clés partagées avec n_I : $\mathcal{I}_1 = \{\bigcup_{n\in N} \mathsf{shk}(n_I, n)\}$*.*

Le scénario de l'attaque est le suivant. La source S envoie une requête vers D. La requête atteint le noeud n_I. L'attaquant reçoit le message suivant $\langle \mathsf{req}, S, D, id, S :: [], \mathsf{hmac}(\langle \mathsf{req}, S, D, id\rangle, \mathsf{shk}(S,D))\rangle$*. L'attaquant peut alors envoyer le message à D au nom de X :*

$$\langle \mathsf{req}, S, D, id, [X; W; S], \mathsf{hmac}(\langle \mathsf{req}, S, D, id\rangle, \mathsf{shk}(S,D))\rangle.$$

Étant donné le fait que D est un voisin de n_I, il va pouvoir capter la transmission. De plus, vu que la liste de noeuds $[X; W; S]$ fini avec X, qui est aussi un voisin de D, la destination D va alors traiter la demande et va envoyer la réponse vers S :
$\langle \mathsf{rep}, D, S, id, [X; W; S], \mathsf{hmac}(\langle \mathsf{rep}, D, S, id, [X; W; S]\rangle, \mathsf{shk}(S,D))\rangle$*.*

Cette réponse va atteindre S en repassant par le noeud malicieux n_I. Plus précisément, l'attaquant va envoyer la réponse vers S au nom de W. Vu que W est un voisin de S, la source va accepter la réponse, qui contient une fausse route.

Dans notre formalisme, on a que :

$$\begin{aligned} K_1 \rightarrow^* & (\lfloor \mathsf{in}\ u_2[\Phi_S].P \rfloor_S \cup \lfloor \mathsf{out}(m').0 \rfloor_D \cup \mathcal{P}'; \mathcal{I}) \\ \rightarrow & (\lfloor \mathsf{in}\ u_2[\Phi_S].P \rfloor_S \cup \lfloor 0 \rfloor_D \cup \mathcal{P}'; \mathcal{I}') \\ \rightarrow & (\lfloor \mathsf{if} \neg \mathsf{way}([X;W;S])\ \mathsf{then}\ \mathsf{out}(\mathsf{error}) \rfloor_S \cup \mathcal{P}'; \mathcal{I}') \\ \rightarrow & (\lfloor \mathsf{out}(\mathsf{error}).0 \rfloor_S \cup \mathcal{P}'; \mathcal{I}') \end{aligned}$$

où $\mathcal{P}'$ *est le reste des processus dans les autres noeuds*

et $\begin{cases} m' = \langle \mathsf{rep}, D, S, id, [X;W;S], \mathsf{hmac}(\langle D, S, id, [X;W;S] \rangle, \mathsf{shk}(S,D)) \rangle \\ \mathcal{I} = \mathcal{I}_1 \cup \{u_1\}, \textit{and} \\ \mathcal{I}' = \mathcal{I}_1 \cup \{u_1\} \cup \{m'\}. \end{cases}$

Cette exécution correspond à une attaque donnée initialement dans [13].

3 Réduction

Dans cette section, on va réduire la taille des graphes à considérer pour tester les propriétés que peuvent garantir un protocole de découverte de route. Elle va se faire en deux étapes. La première consistera à compléter le graphe et la seconde à projeter le graphe. La complétion permet de donner aux noeuds sur le graphe des propriétés communes de voisinage. La deuxième étape de projection regroupe ensuite les noeuds qui ont les même propriétés de voisinage et d'honnêteté. Ainsi, rendre presque tout les agents voisins entre eux dans la première étape réduit le nombre d'agents à l'issue de la projection. L'intérêt de ces étapes de complétion et de projection est qu'elles se font sans perte d'attaque. A l'issue de ces deux étapes, il ne restera plus qu'à tester un petit nombre de petits graphes.

3.1 Complétion

Définition 4 (quasi-complétion)**.** *Soit un graphe $G = (N, E)$ et $(a, b) \in N^2$ tels que $(a, b) \notin E$ (ce qui implique que $a \neq b$). Alors la quasi-complétion de G par rapport à (a, b) est un graphe G^+ tel que $G^+ = (N, N \times N \setminus \{(a, b); (b, a)\})$.*

L'idée derrière la notion de quasi-complétion est le fait de relier tous les noeuds d'un graphe, sauf deux noeuds non voisins présents dans une liste représentant une route non valide mais malgré tout acceptée par le protocole. On montre ensuite que l'attaque dans le graphe original est aussi possible dans le graphe quasi-complété.

Théorème 1. *Soit $G = (N, E)$ un graphe et $\mathcal{M}$ un ensemble de noeuds malicieux. On considère une $\mathcal{Q}_{route}$-configuration K pour G, $\mathcal{M}$, a et b honnêtes et la propriété $\mathsf{way}(x_L)$. S'il y a une $\mathcal{M}$-attaque sur K, alors il existe un graphe quasi-complété G^+ où il y a une $\mathcal{M}$-attaque sur K pour $\mathcal{M}$, a et b.*

La preuve complète se trouve en annexe.

Idée de la démonstration. L'idée de la preuve est de fixer deux noeuds non-voisins dans la topologie mais néanmoins voisins dans la liste acceptée par le processus initiateur (d'où l'attaque). On prend ensuite le quasi complété de ce graphe, où les deux noeuds non voisins ne sont pas reliés. On vérifie que chaque transition effectuée dans le graphe initial est toujours possible dans le graphe quasi-complété. Ainsi, par induction, il existera une attaque dans le graphe quasi-complété.

Il faut prêter une attention toute particulières aux règles liées à la communication et liées aux tests exprimés par rapport à la topologie.

Pour ce qui est des tests, on vérifie que les littéraux vrais dans le graphe initial le sont dans le quasi-complété. Ceci est principalement vrai parce que la logique ne permet pas de tests de non-voisinage. On conclue l'induction avec l'étude de la disjonction et de la conjonction de formules.

Pour la communication, s'il existe une transition de communication dans le graphe initial, il en existe une dans le quasi-complété parce que les canaux sont de type «lossy».

Finalement, la propriété way n'est toujours pas vérifiée par construction du quasi-complété. □

3.2 Projection

On va dans la suite essayer de regrouper les noeuds qui ont les même propriétés de voisinage et d'honnêteté afin de pouvoir réduire le taille des graphes à considérer. C'est pourquoi on va définir la notion de voisinage dans un graphe.

Définition 5 (voisins)**.** *Soit un graphe $G = (N, E)$. Soit $a \in N$, on appelle l'ensemble de ses voisins noté $\mathcal{V}_G(a)$ l'ensemble $\{b \mid (a, b) \in E\}$.*

Ensuite, on construit à partir d'un graphe dans lequel on a une attaque un graphe plus petit dans lequel il y a toujours une attaque. On va projeter les noeuds du graphe en fonction de leur voisinage et de leur honnêteté.

Définition 6 (fonction de projection)**.** *On définit une fonction de projection par rapport à un graphe $G = (N, E)$ et un ensemble de noeuds malicieux $\mathcal{M}$, $\rho^G_{\mathcal{M}} : \mathcal{N}_{\mathsf{loc}} \to \mathcal{N}_{\mathsf{loc}}$ par :*
- *$\rho^G_{\mathcal{M}}(n) \neq \rho^G_{\mathcal{M}}(m)$ si $(\mathcal{V}_G(n) \neq \mathcal{V}_G(m))$ ou $(n \in \mathcal{M}$ et $m \notin \mathcal{M})$ ou $(n \notin \mathcal{M}$ et $m \in \mathcal{M})$.*
- *$\rho^G_{\mathcal{M}}(n) = \rho^G_{\mathcal{M}}(m)$ sinon.*

On note qu'il existe au moins une fonction de projection. Par exemple :

$$\rho^G_{\mathcal{M}} : \begin{cases} n \in \mathcal{M} & \mapsto a_{(\mathcal{V}_G(n), \mathcal{M})} \\ n \notin \mathcal{M} & \mapsto a_{(\mathcal{V}_G(n), \mathcal{H})} \end{cases}$$

Où les $a_{(i,j)}$ sont des noms d'agents différents si les indices sont différents.

Définition 7. *Un projeté d'un graphe $G = (N, E)$ suivant l'ensemble des noeuds malicieux $\mathcal{M}$ et une fonction de projection $\rho^G_{\mathcal{M}}$ est un graphe noté $\rho^G_{\mathcal{M}}(G) = (N', E')$ défini par :*
- *$N' = \rho^G_{\mathcal{M}}(N)$ i.e. $N' = \{\rho^G_{\mathcal{M}}(n) \mid n \in N\}$*
- *$E' = \rho^G_{\mathcal{M}}(E)$ i.e. $E' = \{(\rho^G_{\mathcal{M}}(n), \rho^G_{\mathcal{M}}(m)) \mid (n, m) \in E\}$*

Le projeté d'une configuration concrète $K = (\mathcal{P}; \mathcal{I})$ est une configuration concrète noté $\rho^G_{\mathcal{M}}(K) = (\mathcal{P}'; \mathcal{I}')$ telle que :
- *$\mathcal{P}' = \mathcal{P}\{{}^{\rho^G_{\mathcal{M}}(n)}/_n\}$*
- *$\mathcal{I}' = \mathcal{I}\{{}^{\rho^G_{\mathcal{M}}(n)}/_n\}$*

De plus, le projeté d'un ensemble $\mathcal{E}$ quelconque est noté $\rho^G_{\mathcal{M}}(\mathcal{E}) = \mathcal{E}\{{}^{\rho^G_{\mathcal{M}}(n)}/_n\}$.

On va alors être amené à considérer les configurations projetées des protocoles de découverte de route, pour un graphe projeté.

Lemme 1. *Le projeté d'une $\mathcal{Q}_{route}$-configuration est une $\mathcal{Q}_{route}$-configuration.*

La preuve complète se trouve en annexe.

Idée de la démonstration. La démonstration se justifie principalement par le fait que les noms d'agents présent dans une $\mathcal{Q}_{route}$-configuration sont les arguments des processus paramétrés. Les conditions d'égalité entre les paramètres du processus et le nom du noeud qui l'exécute sont conservés par projection. □

Pour qu'une exécution dans un graphe initial soit aussi possible dans un graphe projeté, le protocole ne doit alors pas faire de tests de différence. Un exécution pourrait en effet bloquer dans le graphe projeté, car la projection crée des redondances.

Définition 8 (protocole souple)**.** *Un protocole de découverte de route $\mathcal{Q}_{route}$ est dit souple s'il ne comporte que des processus paramétrés dont la logique des tests ne comporte pas de littéraux* alone*,* ¬alone*,* ¬loop*.*

Lemme 2. *Soit G un graphe et $\mathcal{M}$ un ensemble de noeuds malhonnêtes, $\rho^G_{\mathcal{M}}$ une fonction de projection, Φ une formule d'un test de protocole souple. Si $[\![\Phi]\!]_G = 1$, alors $[\![\rho^G_{\mathcal{M}}(\Phi)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1$.*

La preuve complète se trouve en annexe.

Idée de la démonstration. On fait une démonstration par induction. On vérifie que le lemme est vrai pour des littéraux. Les voisinages sont conservés par la projection : deux noeuds voisins dans le graphe initial sont toujours voisins dans le projeté. Ainsi, les littéraux concernant les propriétés de voisinages sont stables par projection. En ce qui concerne la proposition loop, une liste contenant des redondances sera projetée sur une liste avec des redondances, parce que la projection est une fonction. On conclue avec l'étude de la disjonction et de la conjonction de formules. □

On peut maintenant établir le théorème suivant : s'il y a une attaque dans un graphe, il en existe une dans un graphe projeté.

Théorème 2. *Soit $G = (N, E)$ un graphe et $\mathcal{M}$ un ensemble de noeuds malicieux, $\mathcal{Q}_{route}$ un protocole de découverte de route souple. On considère une $\mathcal{Q}_{route}$-configuration $K = (\mathcal{P}; \mathcal{I})$ pour G, $\mathcal{M}$, a et b honnêtes et la propriété* way(x_L)*. S'il y a une $\mathcal{M}$-attaque sur K et ¬way(x_L) pour la topologie G, alors quelque soit la fonction de projection $\rho^G_{\mathcal{M}}$, il existe une $\rho^G_{\mathcal{M}}(\mathcal{M})$-attaque sur la $\mathcal{Q}_{route}$-configuration $\rho^G_{\mathcal{M}}(K)$ et $\rho^G_{\mathcal{M}}(G)$.*

La preuve complète se trouve en annexe.

Idée de la démonstration. On note qu'avec le lemme 1, on a déjà que le projeté d'une $\mathcal{Q}_{route}$-configuration est une $\mathcal{Q}_{route}$-configuration.

Puis, on vérifie qu'à chaque transition dans le graphe original correspond une transition dans le graphe projeté. Plus formellement, on va montrer que si :

$$K = (\mathcal{P}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\mathcal{P}'; \mathcal{I}') = K'$$

alors on a aussi que :

$$\rho^G_{\mathcal{M}}(K) \rightarrow_{\rho^G_{\mathcal{M}}(G), \rho^G_{\mathcal{M}}(\mathcal{M})} \rho^G_{\mathcal{M}}(K')$$

Par induction, on aura le résultat escompté.

Il faut prêter attention aux règles liées à la communication et liées tests. Pour ce qui est des règles liées aux tests, on conclue avec le lemme 2.

Pour les règles liées à la communication, une transition possible dans le graphe initial est possible dans le projeté parce les voisinages sont conservés. De plus, la projection et l'unification sont

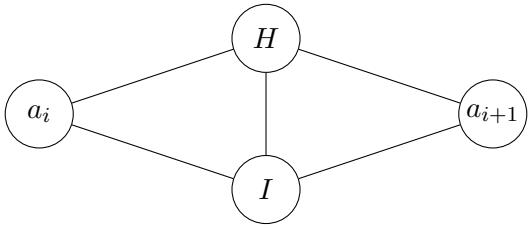
commutatives. En effet, la projection porte sur les noms d'agents, alors que l'unification porte sur variables.

□

3.3 Application

En cumulant les deux résultats principaux – théorème de complétion et théorème de projection – on peut déduire que le graphe à considérer est alors de quatre noeuds. Le nombre fini de configurations à vérifier permet d'entrevoir la possibilité d'une vérification automatique des protocoles de découverte de route.

Corollaire 1. *S'il existe une attaque pour un protocole de découverte de route souple et la propriété* $\neg\mathsf{way}(x_L)$*, alors il existe une attaque sur ce protocole sur un graphe de quatre noeuds pour la même propriété. Il peut être représenté comme suit :*



On a ici $\mathcal{M} = \{I\}$ *ou* $\mathcal{M} = \{I; a_{i+1}\}$

Démonstration. S'il existe une attaque pour un protocole de découverte de route souple $\mathcal{Q}_{route}$, alors il existe une $\mathcal{Q}_{route}$-configuration K pour un graphe G et un nombre d'agent malicieux $\mathcal{M}$, a et b honnêtes et $\neg\mathsf{way}(x_L)$ tel qu'il y ait une $\mathcal{M}$-attaque.

Avec le théorème 1, on sait qu'il existe une $\mathcal{M}$-attaque sur un graphe quasi-complété G^+. Puis avec le théorème 2, on sait que pour une fonction de projection $\rho^{G^+}_{\mathcal{M}}$ vis-à-vis de $\mathcal{M}$ et G^+, il existe une $\rho^{G^+}_{\mathcal{M}}(\mathcal{M})$-attaque sur le graphe $\rho^{G^+}_{\mathcal{M}}(G)$ pour la configuration $\rho^{G^+}_{\mathcal{M}}(K)$.

Après l'étape de complétion, on se retrouve avec deux noeuds a_i et a_{i+1} qui ne sont pas voisins. Tous les autres noeuds sont tous voisins entre eux et ont les même voisins. La projection se faisant suivant l'état du voisinage et l'honnêteté des noeuds, on se retrouve avec un graphe projeté d'au plus quatre noeuds. Les deux noeuds a_i et a_{i+1} sont les seuls noeuds non-voisins entre-eux. La projection unifiant les noeuds avec même propriétés de voisinages, ces deux noeuds restent distincts dans le graphe projeté. En ce qui concerne les autres noeuds, trois cas peuvent se présenter. Soit ils sont tous honnêtes, soit tous malhonnêtes, soit il existe des honnêtes et des malhonnêtes.

Dans le dernier cas, on projette ces noeuds sur deux noeuds distincts, H (pour les honnêtes) et I (pour les compromis). Les cas où tous les noeuds ont même honnêteté peuvent alors aussi être projeté dans le graphe de quatre noeuds, car les petits graphes sont inclus dans celui là.

Si maintenant, on veut lister toutes les configurations que l'on peut obtenir à la suite de la complétion et la projection, il faut prendre en compte l'honnêteté de a_i et a_{i+1} et la place de la source et de la destination.

Étant donné la propriété way, les agents a_i et a_{i+1} ne peuvent pas être tous deux malhonnêtes.

Ainsi les configurations des noeuds malicieux du graphe projeté à observer sont $\mathcal{M} = \{I\}$ ou $\mathcal{M} = \{I; a_i\}$ ou $\mathcal{M} = \{I; a_{i+1}\}$.

On peut noter aussi qu'il y a une symétrie de certaines configurations : a_i et a_{i+1} ont des rôles symétriques. De ce fait, les configurations à observer sont : $\mathcal{I} = \{I\}$ et (Source, Destination) $\in \{(H,H),(H,a_i),(a_i,H),(a_i,a_i),(a_i,a_{i+1})\}$ ou
$\mathcal{I} = \{I, a_{i+1}\}$ et (Source, Destination) $\in \{(H,H),(H,a_i),(a_i,H),(a_i,a_i)\}$

De plus, la configuration $\rho^{G^+}_{\mathcal{M}}(K)$ est une configuration respective au protocole de découverte de route.

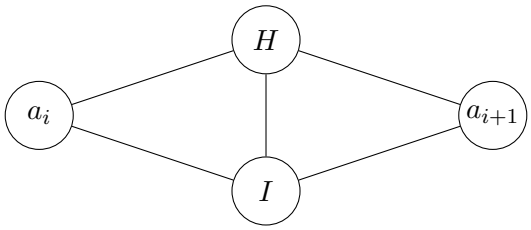
De ce fait, s'il y a une attaque sur un protocole de découverte de route souple, alors il en existe une dans une de ces configurations dans le petit graphe décrit précédemment. □

On peut noter que désormais, il suffit de tester les configurations comme décrite précédemment pour tester le protocole de découverte de route dans son ensemble. On va utiliser ce résultat dans l'exemple suivant.

Exemple 4. *On va continuer l'exemple du protocole* SRP. *On va considérer ici la version souple de ce protocole : on va ôter tous les tests qui concernent l'occurrence des noms d'agents. On peut noter que la propriété que l'on demande à la liste considérée comme une route ne repose pas sur ces tests.* ¬way *est indépendante de l'occurrence des noms d'agents dans la liste.*

Avec le résultat mis en avant par le corollaire 1, on peut alors étudier un petit nombre de graphes et configurations.

On peut considérer comme graphe $G = (N, E)$ *avec*
- $N = \{a_i; a_{i+1}; H; I\}$
- $E = \{(a_i, H); (a_i, I); (a_{i+1}, H); (a_{i+1}, H); (I, H)\}$
- $\mathcal{M} = \{a_{i+1}, I\}$ *ou* $\mathcal{M} = \{I\}$



Les configurations à observer dépendent des arguments du processus initiateur spécial (source et destination) et de l'honnêteté des agents a_i *et* a_{i+1}.

Dans la suite de cet exemple, on va considérer que H *veut lancer un protocole de découverte de route avec lui même,* a_i *et* a_{i+1} *sont honnêtes, alors on retombe sur cette configuration initiale :*

$$
\begin{array}{l}
\lfloor P_i(H, a_i).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\rfloor_H \cup \\
(\lfloor !P_{\mathsf{req}}(a_i)\rfloor_{a_i} \cup \lfloor !P_{\mathsf{rep}}(a_i)\rfloor_{a_i} \cup \bigcup_{n\in N}\lfloor !P_i(a_i, n)\rfloor_{a_i} \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(a_i, n)\rfloor_{a_i} \cup \\
(\lfloor !P_{\mathsf{req}}(a_{i+1})\rfloor_{a_{i+1}} \cup \lfloor !P_{\mathsf{rep}}(a_{i+1})\rfloor_{a_{i+1}} \cup \bigcup_{n\in N}\lfloor !P_i(a_{i+1}, n)\rfloor_{a_{i+1}} \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(a_{i+1}, n)\rfloor_{a_{i+1}} \cup \\
(\lfloor !P_{\mathsf{req}}(H)\rfloor_H \cup \lfloor !P_{\mathsf{rep}}(H)\rfloor_H \cup \bigcup_{n\in N}\lfloor !P_i(H, n)\rfloor_H \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(H, n)\rfloor_H \cup \\
(\lfloor !P_{\mathsf{req}}(I)\rfloor_I \cup \lfloor !P_{\mathsf{rep}}(I)\rfloor_I \cup \bigcup_{n\in N}\lfloor !P_i(I, n)\rfloor_I \cup \bigcup_{n\in N}\lfloor !P_{\mathsf{dest}}(I, n)\rfloor_I
\end{array}
$$

où les processus paramétrés du protocole de découverte de route souple sont ceux de la section 2.3, mais sans les tests de différence.

De plus, il faut considérer dans la connaissance de l'attaquant les clés partagées avec I *:* $\mathcal{I}_0 = \{\bigcup_{n\in N} \mathsf{shk}(I, n)\}$.

On peut noter que par rapport à l'exemple 3, on étudie le projeté du quasi-complété du graphe étudié jusqu'à présent, dont la fonction de projection est :

$$
\rho^G_{\mathcal{M}} : \begin{cases} S & \mapsto H \\ D & \mapsto H \\ W & \mapsto a_i \\ X & \mapsto a_{i+1} \\ n_I & \mapsto I \end{cases}
$$

4 Applications dans un outil de vérification automatique : ProVerif

Il existe plusieurs outils de vérification automatique des protocoles de sécurité. On peut notamment citer Avispa [6] qui est devenu la plate-forme Avantssar et le logiciel Proverif [11].

4.1 ProVerif

ProVerif est un outil de vérification automatique des propriétés de sécurité sur des protocoles cryptographiques. Il permet de vérifier des protocoles dans le modèle symbolique, c'est-à-dire considère que les primitives cryptographiques sont parfaites, incassables. Il représente les protocoles et raisonne avec des clauses de Horn. Il permet en particulier de modéliser le chiffrement symétrique comme asymétrique, les fonctions de hachages et les signatures.

Il permet de définir des prédicats. Les protocles peuvent effectuer des tests sur ces prédicats. C'est ainsi qu'on va définir les différents tests de voisinages. ProVerif utilise notamment des prédicats pour définir des ensembles comme des listes. Cette fonctionnalité est indispensable à la modélisation des protocoles de routage, qui manipulent des listes.

Il permet d'étudier la sécurité des protocoles dans un nombre non-borné de sessions. De plus, si une propriété de sécurité n'est pas prouvée, alors ProVerif essaie de construire la trace d'attaque.

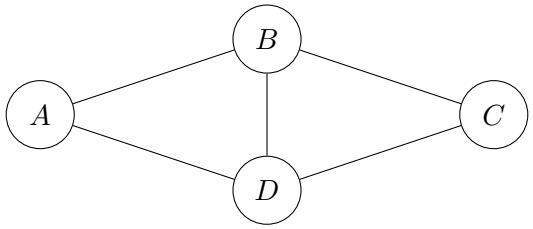
J'ai choisi cet outil car il est particulièrement adapté pour l'étude des protocoles dans un cadre d'un nombre non borné de sessions. Le résultat théorique que l'on a apporté se place de ce cadre, afin de capturer toutes les configurations possibles. De plus, j'avais déjà étudié cet outil dans le cadre de mes cours.

4.2 Les codages de la topologie des réseaux ad-hoc

4.2.1 Communication

On représente la topologie des réseaux ad-hoc par des graphes. Avec notre résultat théorique, on sait que l'on a neuf différentes configurations à étudier. On va donc coder en dur les graphes sous-jacents.

On rappelle que le graphe que l'on doit considérer est composé de quatre noeuds A, B, C, D tous voisins sauf A et C. Les différentes topologies à considérer varient selon l'honnêteté des agents et selon les sources et destinations des protocoles de routage.



On fait le même raisonnement que dans l'exemple 4. Les configurations à tester sont : $\mathcal{I} = \{D\}$ et (Source, Destination) $\in \{(B,B),(B,A),(A,B),(A,A),(A,C)\}$ ou
$\mathcal{I} = \{D,C\}$ et (Source, Destination) $\in \{(B,B),(B,A),(A,B),(A,A)\}$

Quel que soit la configuration à considérer, l'attaquant contrôle toujours D (connaît tous les secrets de D) qui est connecté à tous les autres noeuds. De ce fait, la communication entre A et C est laissée au bon vouloir de l'attaquant. De plus, le formalisme que l'on a pris considère des canaux de type «lossy», qui peuvent perdre des messages. Ceci peut être vu aussi comme contrôlé par l'attaquant. De ce fait, on peut considérer comme configuration quatre agents A, B, C, D

qui interagissent entre eux par l'intermédiaire d'un attaquant, comme modèle de communication équivalent.

4.2.2 Honnêteté des agents

Pour prendre en compte la malhonnêteté de l'agent D, on compromet tous ses secrets en les divulguant sur le canal public. Ainsi, l'attaquant est en mesure d'effectuer n'importe quelle opération que l'agent malhonnête D pourrait faire. On fait de même en fonction de l'honnêteté de C.

4.2.3 Tests de voisinages et propriété de sécurité

Pour les tests de voisinages, on liste à ProVerif les couples de noeuds qui sont voisins (tous sauf A et C), et on liste aussi les listes de nom d'agents qui ne sont pas des listes valables de route (toutes les listes ne comprenant jamais A et C ou C et A à la suite).

ProVerif permet de définir des prédicats, générés par des clauses de Horn. On définit alors deux prédicats, l'un pour le voisinage (prédicat check), l'autre pour les listes non-valables (prédicat negcheckl), qui vont pouvoir être testées par les processus.

```
pred check(node, node).

check(A, A);check(A, B);check(A, D);
check(B, A);check(B, B);check(B, C);check(B, D);
check(C, B);check(C, C);check(C, D);
check(D, A);check(D, B);check(D, C);check(D, D).
```

On choisit de coder les listes non-valables car elles sont plus simplement exprimable en clauses que les listes valables. Ainsi, on testera dans les protocoles la négation du prédicat de non-liste valable.

```
pred negcheckl(node, list).

clauses
forall l:list; negcheckl(C,consset(A, consset(C,l)));
forall l:list; negcheckl(C,consset(C, consset(A,l)));
forall l:list; negcheckl(A,consset(A, consset(C,l)));
forall l:list; negcheckl(A,consset(C, consset(A,l)));
forall a:node, l:list, b:node; negcheckl(a,l) -> negcheckl(a,consset(b,l));
```

Ainsi, ces tests permettent de spécifier la topologie du graphe sous-jacent.

On peut noter que pour les noeuds B et D, toutes les listes peuvent être acceptées. Contrairement à A ou C, ils ne savent pas directement que A et C ne sont pas voisins. Par contre, A ou C peuvent distinguer qu'ils ne sont pas voisins, et ainsi, la propriété way est se confond avec la propriété checkl pour A et C.

```
pred negway(list).

forall l:list ; negcheckl(A,l) -> negway(l);
```

4.3 Applications

4.3.1 SRP appliqué à DSR

On a déjà décrit le protocole SRP appliqué à DSR dans les sections précédentes. On obtient une attaque.

On présente la sortie de ProVerif de façon synthétique en annexe aussi. On note que l'on retrouve la projection d'une attaque déjà décrite précédemment.

4.3.2 SDMSR

Le protocole de routage sécurisé SDMSR a été introduit dans [10]. L'objectif de ce protocole est de découvrir plusieurs routes d'une source vers une destination. Il peut être basé sur deux mécanismes de chiffrements : la signature à base de RSA ou de chaînes de hachés. On considère la signature à base de RSA et ainsi il peut être modélisé dans notre formalisme. Bien que la description du protocole ne spécifie pas explicitement l'utilisation d'un protocole de découverte des voisins, on considère malgré tout son utilisation, pour éviter des attaques triviales. On peut de ce fait effectuer les tests de voisinages.

Les graphes à considérer pour la vérification sont les mêmes que pour SRP appliqué à DSR. On présente la sortie de ProVerif de façon synthétique en annexe aussi.

On note que l'on retrouve la projection d'une attaque déjà découverte dans [8].

5 Conclusion

Dans ce travail, on a réduit le nombre de configurations à tester pour vérifier les protocoles de routage sécurisés à un petit nombre de petits graphes. De ce fait, on a pu utiliser l'outil ProVerif pour analyser les protocoles de routage dans le cadre d'un nombre non borné de session.

On n'a étudié qu'une seule propriété de sécurité ici (way), mais notre résultat semble pouvoir s'étendre à d'autres propriétés de voisinages. De même, d'autres primitives cryptographiques auraient pu rentrer dans notre formalisme.

Notre formalisme ne prend pas en compte les protocoles de routage basé sur des tables de routage. Cependant, il semble que les résultats s'appliquent aussi à ce genre de protocole.

Il en est de même pour les protocoles de routage effectuant des tests récursifs. En effet, endairA[4] et Ariadne[15] ne rentrent pas dans notre formalisme car ils comportent des tests récursifs qui ne sont pas modélisables. Cependant, on aurait pu sans trop de difficultés étendre notre résultat théorique pour capturer ce genre de tests.

Malgré les améliorations possibles au niveau théorique, l'application du résultat est limité par la puissance des outils existants. En effet, l'outil Proverif ne permet pas de prendre en compte tous ces mécanismes de tests. Il n'est pas possible d'inverser des listes avec Proverif, chose utile dans les protocoles comme Ariadne. De plus, les prédicats dans ProVerif doivent être définis avec un type de clauses qui ne permet pas de construire les chaînes de hachés.

Outre ce problème, les protocoles de routage que l'on a vérifiés sont des simplifications des protocoles existants. On ne prend en effet pas en considération les tests de différence. Dans un cadre général, ceci amène de fausses attaques. Cependant, on pense que dans les protocoles existants, les conditions sont réunies pour que la simplification n'apporte pas de fausses attaques, étant donné qu'on n'étudie qu'une propriété de voisinage.

Il semble donc possible d'étendre ce résultat pour tenir compte des tests de différence.

Références

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In Proc. 28th Symposium on Principles of Programming Languages (POPL'01), pages 104–115. ACM Press, 2001.

[2] M. Abadi and A. Gordon. A calculus for cryptographic protocols : The spi calculus. In Proc. 4th Conference on Computer and Communications Security (CCS'97), pages 36–47. ACM Press, 1997.

[3] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just fast keying in the pi calculus. ACM Transactions on Information and System Security (TISSEC), 10(3) :1–59, July 2007.

[4] Gergely Ács, Levente Buttyán, and István Vajda. Provably secure on-demand source routing in mobile ad hoc networks. IEEE Transactions on Mobile Computing, 5(11) :1533–1546, 2006.

[5] Todd R. Andel, G. Back, and Alec Yasinsac. Automating the security analysis process of secure ad hoc routing protocols. Simulation Modelling Practice and Theory, 19(9) :2032–2049, 2011.

[6] A. Armando et al. The AVISPA Tool for the automated validation of internet security protocols and applications. In Proc. 17th International Conference on Computer Aided Verification, CAV'2005, volume 3576 of LNCS, pages 281–285. Springer, 2005.

[7] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra Abad. Formal Analysis of SAML 2.0 Web Browser Single Sign-On : Breaking the SAML-based Single Sign-On for Google Apps. In the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008), Hilton Alexandria Mark Center, Virginia, USA, 2008. ACM Press.

[8] Mathilde Arnaud, Véronique Cortier, and Stéphanie Delaune. Modeling and verifying ad hoc routing protocols. In Proceedings of the 23rd IEEE Computer Security Foundations Symposium (CSF'10), pages 59–74, Edinburgh, Scotland, UK, July 2010. IEEE Computer Society Press.

[9] Davide Benetti, Massimo Merro, and Luca Viganò. Model checking ad hoc network routing protocols : ARAN vs. endairA. In José Luiz Fiadeiro, Stefania Gnesi, and Andrea Maggiolo-Schettini, editors, SEFM, pages 191–202. IEEE Computer Society, 2010.

[10] Sebastien Berton, Hao Yin, Chuang Lin, and Geyong Min. Secure, disjoint, multipath source routing protocol(sdmsr) for mobile ad-hoc networks. In Proceedings of the Fifth International Conference on Grid and Cooperative Computing, GCC '06, pages 387–394, Washington, DC, USA, 2006. IEEE Computer Society.

[11] Bruno Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In Proc. 20th International Conference on Automated Deduction (CADE'05), 2005.

[12] Bruno Blanchet and Avik Chaudhuri. Automated formal analysis of a protocol for secure file sharing on untrusted storage. In Proceedings of the 29th IEEE Symposium on Security and Privacy (S&P'08), pages 417–431. IEEE, 2008.

[13] Levente Buttyán and István Vajda. Towards Provable Security for Ad Hoc Routing Protocols. In Proc. 2nd ACM workshop on Security of ad hoc and sensor networks (SASN'04), pages 94–105, New York, NY, USA, 2004. ACM.

[14] Yih-Chun Hu, A. Perrig, and D. B. Johnson. Wormhole attacks in wireless networks. Selected Areas in Communications, IEEE Journal on, 24(2) :370–380, 2006.

[15] Yih-Chun Hu, Adrian Perrig, and David Johnson. Ariadne : A Secure On-Demand Routing Protocol for Ad Hoc Networks. Wireless Networks, 11, January 2005.

[16] D. Johnson, D. Maltz, and J. Broch. DSR : The Dynamic Source Routing Protocol for multi-hop wireless ad hoc networks. In Ad Hoc Networking, pages 139–172, 2001.

[17] L. Lazos, R. Poovendran, C. Meadows, P. Syverson, and L. W. Chang. Preventing wormhole attacks on wireless ad hoc networks : a graph theoretic approach. In Wireless Communications and Networking Conference, volume 2, pages 1193–1199 Vol. 2, 2005.

[18] Sebastian Nanz and Chris Hankin. A Framework for Security Analysis of Mobile Wireless Networks. Theoretical Computer Science, 367(1) :203–227, 2006.

[19] P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. In Proc. SCS Communication Networks and Distributed Systems Modelling Simulation Conference (CNDS), 2002.

[20] Charles E. Perkins and Elizabeth M. Belding-Royer. Ad-hoc on-demand distance vector routing. In Proc. 2nd Workshop on Mobile Computing Systems and Applications (WMCSA '99), pages 90–100, 1999.

[21] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS : Security protocols for sensor networks. Wireless Networks, 8(5) :521–534, September 2002.

[22] Marcin Poturalski, Panos Papadimitratos, and Jean-Pierre Hubaux. Towards Provable Secure Neighbor Discovery in Wireless Networks. In Proceedings of the 6th ACM workshop on Formal methods in security engineering, pages 31–42. ACM, 2008.

A Annexes

Théorème 1. *Soit $G = (N, E)$ un graphe et $\mathcal{M}$ un ensemble de noeuds malicieux. On considère une $\mathcal{Q}_{route}$-configuration K pour G, $\mathcal{M}$, a et b honnêtes et la propriété $\mathsf{way}(x_L)$. S'il y a une $\mathcal{M}$-attaque sur K, alors il existe un graphe quasi-complété G^+ où il y a une $\mathcal{M}$-attaque sur K pour $\mathcal{M}$, a et b.*

Démonstration. On remarque déjà qu'une $\mathcal{Q}_{route}$-configuration pour un graphe $G = (N, E)$, $\mathcal{M}$, a et b honnêtes et la propriété $\mathsf{way}(x_L)$, est aussi une $\mathcal{Q}_{route}$-configuration pour n'importe quel graphe (N, E'), pour $\mathcal{M}$, a et b honnêtes et la propriété $\mathsf{way}(x_L)$, où E' est quelconque. De ce fait, une $\mathcal{Q}_{route}$-configuration pour un graphe G est aussi une $\mathcal{Q}_{route}$-configuration pour un quasi-complété de G.

S'il y a une $\mathcal{M}$-attaque sur la $\mathcal{Q}_{route}$-configuration sur le graphe G, alors la liste instanciée x_L du processus initiateur contient deux éléments consécutifs qui ne sont pas voisins alors qu'au moins un des deux est honnête. On note ces noeuds c et d. On prend la quasi-complétion par rapport à c et d, que l'on note $G^+ = (N, E^+)$. On vérifie qu'il y a aussi une $\mathcal{M}$-attaque sur la $\mathcal{Q}_{route}$ configuration pour le graphe quasi-complété.

On part de la même configuration concrète K. On sait qu'il y a une $\mathcal{M}$-attaque sur le graphe G donc d'après la définition d'attaque, il existe une séquence de transitions :

$$K = (\lfloor P_i(a,b).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\rfloor_a \cup \mathcal{P}; \mathcal{I}) \ \rightarrow^*_{G,\mathcal{M}}\ (\lfloor \mathsf{out}(\mathsf{error})\rfloor_a \cup \mathcal{P}', \mathcal{I}')$$

où error est un symbole spécial n'apparaissant pas ailleurs dans la configuration K.

On démontre alors que chacune des transitions concrètes dans le graphe original est possible dans le quasi-complété G^+. Plus précisément, si $K \rightarrow_{G,\mathcal{M}} K'$ alors, $K \rightarrow_{G^+,\mathcal{M}} K'$, et on pourra conclure par induction qu'il y a une $\mathcal{M}$-attaque sur le graphe quasi-complété.

Pour les règles Par, Repl, New, les conditions pour effectuer ces transitions sont indépendantes de la topologie du graphe sous-jacent. Si la transition a été effectuée dans le graphe G, elle est possible dans le quasi-complété.

S'il s'agit de la règle If-Then,

$$\begin{aligned} \textsc{If-Then}\quad & K = (\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P\rfloor_n \cup \mathcal{P}; \mathcal{I}) \\ & \rightarrow_{G,\mathcal{M}}\quad (\lfloor P\rfloor_n \cup \mathcal{P}; \mathcal{I}) = K' \text{ si } [\![\Phi]\!]_G = 1 \end{aligned}$$

On doit prendre en compte la formule Φ, car son interprétation dépend de G, le graphe sous-jacent. On doit donc considérer la logique du noeud $\mathcal{L}_{noeud}$ et le test $\neg\mathsf{way}(x_L)$.

Soit $\Phi \in \mathcal{L}_{noeud}$. On démontre par induction sur la taille de la formule $\Phi \in \mathcal{L}_{noeud}$ que $[\![\Phi]\!]_G = 1 \Rightarrow\ [\![\Phi]\!]_{G^+} = 1$.

Cas de base : Φ est un littéral. Si Φ est un test de voisinage : check ou checkl, alors étant donné que $E \subseteq E^+$, on a clairement $[\![\Phi]\!]_G = 1 \Rightarrow\ [\![\Phi]\!]_{G^+} = 1$. Sinon, Φ est un test loop, $\neg\mathsf{loop}$, alone ou $\neg\mathsf{alone}$ et la valeur du test est indépendante du graphe : on a aussi $[\![\Phi]\!]_G = 1 \Leftrightarrow\ [\![\Phi]\!]_{G^+} = 1$.

Si Φ n'est pas un littéral, elle est la conjonction ou la disjonction de deux formules plus petites. $\Phi = \Phi_1 \wedge \Phi_2$ ou $\Phi = \Phi_1 \vee \Phi_2$. On va traiter le cas disjonction, sachant que le raisonnement est similaire pour la conjonction. Par hypothèse de récurrence, on a bien $[\![\Phi_1]\!]_G = 1 \Rightarrow\ [\![\Phi_1]\!]_{G^+} = 1$ et $[\![\Phi_2]\!]_G = 1 \Rightarrow\ [\![\Phi_2]\!]_{G^+} = 1$. Donc on a $[\![\Phi]\!]_G = 1 \Rightarrow\ ([\![\Phi_1]\!]_G = 1 \vee [\![\Phi_2]\!]_G = 1) \Rightarrow ([\![\Phi_1]\!]_{G^+} = 1 \vee [\![\Phi_2]\!]_{G^+} = 1) \Rightarrow [\![\Phi]\!]_{G^+} = 1$.

On conclue par le principe de récurrence.

Soit $\Phi = \neg\mathsf{way}(l)$. Ce test est vrai dans le graphe G, car il y a une $\mathcal{M}$-attaque sur G. Or, par construction de G^+, l n'est pas une route valide selon way dans G^+. De ce fait, on a aussi $[\![\neg\mathsf{way}(l)]\!]_{G^+} = 1$.

Dans tous ces cas, on a $[\![\Phi]\!]_G = 1 \Rightarrow [\![\Phi]\!]_{G^+} = 1$, et donc $K \rightarrow_{G^+,\mathcal{M}} K'$, par la même règle IF-THEN.

S'il s'agit d'une règle COMM, alors on a :

$$\begin{array}{ll} \textsc{Comm} & (K = \{\lfloor \mathsf{in}\ u_j[\Phi_j].P_j \rfloor_{n_j} \mid \mathsf{mgu}(t,u_j) \neq \bot, [\![\Phi_j\sigma_j]\!]_G = 1, (n,n_j) \in E\} \\ & \quad \cup \lfloor \mathsf{out}(t).P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \\ & \quad \rightarrow_{G,\mathcal{M}} \quad (\{\lfloor P_j\sigma_j \rfloor_{n_j}\} \cup \lfloor P \rfloor_n \cup \mathcal{P}; \mathcal{I}') = K' \end{array}$$

où $\sigma_j = \mathsf{mgu}(t,u_j)$, $\mathcal{I}' = \mathcal{I} \cup \{t\}$ si $(n,n_I) \in E$ pour un $n_I \in \mathcal{M}$ et $\mathcal{I}' = \mathcal{I}$ sinon.

On doit prendre en compte le sous graphe sous-jacent pour les évaluations des formules et aussi dans les liaisons. Les formules sont dans la logique $\mathcal{L}_{noeud}$. De ce fait, on a comme pour la règle IF-THEN, $[\![\Phi]\!]_G = 1 \Rightarrow [\![\Phi]\!]_{G^+} = 1$. D'autre part, si $(n,n_j) \in E$, alors étant donné que $E \subseteq E^+$, alors $(n,n_j) \in E^+$. Les autres conditions pour effectuer la transition COMM sont indépendantes du graphe sous-jacent, et ont donc même valeur quel que soit le graphe sous-jacent. Donc $K \rightarrow_{G^+,\mathcal{M}} K'$ aussi.

S'il s'agit d'une règle IN, les conditions dépendantes du graphe sous-jacent sont les mêmes que pour la règle COMM, et donc on a aussi $K \rightarrow_{G,\mathcal{M}} K' \Rightarrow K \rightarrow_{G^+,\mathcal{M}} K'$. □

Lemme 1. *Le projeté d'une $\mathcal{Q}_{route}$-configuration est une $\mathcal{Q}_{route}$-configuration.*

Démonstration. Soient $G = (N,E)$ un graphe et $\mathcal{M}$ un ensemble de noeuds malicieux. On considère une $\mathcal{Q}_{route}$-configuration $K = (\mathcal{P};\mathcal{I})$ pour G, $\mathcal{M}$, a et b honnêtes et la propriété $\mathsf{way}(x_L)$. On considère une fonction de projection $\rho^G_{\mathcal{M}}$. On vérifie que $\rho^G_{\mathcal{M}}(K)$ est bien une $\mathcal{Q}_{route}$-configuration pour le graphe $\rho^G_{\mathcal{M}}(G)$, pour les noeuds honnêtes $\rho^G_{\mathcal{M}}(a)$ et $\rho^G_{\mathcal{M}}(b)$, et $\rho^G_{\mathcal{M}}(\mathcal{M})$ et la même propriété. $\rho^G_{\mathcal{M}}(K) = \rho^G_{\mathcal{M}}((\mathcal{P};\mathcal{I})) = (\rho^G_{\mathcal{M}}(\mathcal{P}); \rho^G_{\mathcal{M}}(\mathcal{I}))$.

- Soient n, P tels que $\lfloor P \rfloor_n \in \rho^G_{\mathcal{M}}(\mathcal{P})$. Il existe $m \in N$ et P' tel que $\lfloor P \rfloor_n = \lfloor \rho^G_{\mathcal{M}}(P') \rfloor_{\rho^G_{\mathcal{M}}(m)}$ et $\lfloor P' \rfloor_m \in \mathcal{P}$. Donc $\exists a_2, ..., a_k \in N, P'' \in \mathcal{Q}_{route}$ tel que $P' = !P''(m, a_2, ..., a_k)) \vee (m = a \wedge P' = P_i(a,b).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})$ où error n'apparaît nulle par ailleurs dans la configuration. On a que si $P \in \mathcal{Q}_{route}$, alors $\rho^G_{\mathcal{M}}(P(x)) = P(\rho^G_{\mathcal{M}}(x))$ car les processus paramétrés ne contiennent pas de termes $t \in \mathcal{N}_{\mathsf{loc}}$ clos.
  Donc $\exists \rho^G_{\mathcal{M}}(a_2), ..., \rho^G_{\mathcal{M}}(a_k) \in \rho^G_{\mathcal{M}}(N), P'' \in \mathcal{Q}_{route}$ tel que $P = !P''(n, \rho^G_{\mathcal{M}}(a_2), ..., \rho^G_{\mathcal{M}}(a_k))) \vee (n = \rho^G_{\mathcal{M}}(a) \wedge P = P_i(\rho^G_{\mathcal{M}}(a), \rho^G_{\mathcal{M}}(b)).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})$ où error n'apparaît nulle par ailleurs dans la configuration.
- Soient $b_1, ..., b_k \in \rho^G_{\mathcal{M}}(N)$, il existe $a_1, ..., a_k \in N$ tel que $\rho^G_{\mathcal{M}}(a_i) = b_i$. Et on a que $P' \in \mathcal{Q}_{route} \Rightarrow \lfloor !P'(a_1, ..., a_k) \rfloor_{a_1} \in \mathcal{P}$, et donc, $P' \in \rho^G_{\mathcal{M}}(\mathcal{Q}_{route}) \Rightarrow \lfloor !P'(b_1, ..., b_k) \rfloor_{b_1} \in \rho^G_{\mathcal{M}}(\mathcal{P})$ car les processus paramétrés ne contiennent pas de termes $t \in \mathcal{N}_{\mathsf{loc}}$ clos.
- il existe $P \in \mathcal{P}$ tel que $P = P_i(a,b).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})$ où error n'apparaît nulle par ailleurs dans la configuration, donc il existe $\rho^G_{\mathcal{M}}(P) \in \rho^G_{\mathcal{M}}(\mathcal{P})$ et $\rho^G_{\mathcal{M}}(P)$ est tel que $\rho^G_{\mathcal{M}}(P) = P_i(\rho^G_{\mathcal{M}}(a), \rho^G_{\mathcal{M}}(b)).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})$, car les processus paramétrés ne contiennent pas de termes $t \in \mathcal{N}_{\mathsf{loc}}$ clos.

De plus, $\rho^G_{\mathcal{M}}(\mathcal{I}) = \rho^G_{\mathcal{M}}(N) \cup \{\mathsf{shk}(n,p) \mid n \in \rho^G_{\mathcal{M}}(\mathcal{M}) \vee p \in \rho^G_{\mathcal{M}}(\mathcal{M})\} \cup \{\mathsf{priv}(n) \mid n \in \rho^G_{\mathcal{M}}(\mathcal{M})\}$.

Toutes les conditions sont satisfaites pour que le projeté d'une $\mathcal{Q}_{route}$-configuration soit une $\mathcal{Q}_{route}$-configuration. □

Lemme 2. *Soit G un graphe et $\mathcal{M}$ un ensemble de noeuds malhonnêtes, $\rho^G_{\mathcal{M}}$ une fonction de projection, Φ une formule d'un test de protocole souple. Si $[\![\Phi]\!]_G = 1$, alors $[\![\rho^G_{\mathcal{M}}(\Phi)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1$.*

Démonstration. On démontre cette implication par récurrence sur la taille de la formule.

Cas de base, Φ est un littéral. Soit Φ est un test de voisinage : check ou checkl ou $\neg\mathsf{way}$, alors étant donné que le voisinage est conservé : $\forall(x,y) \in E, \rho^G_{\mathcal{M}}((x,y)) \in \rho^G_{\mathcal{M}}(E)$, on a bien $[\![\rho^G_{\mathcal{M}}(\Phi)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1$. Sinon, il s'agit d'un test $\mathsf{loop}(l)$, s'il y a redondance dans la liste l, alors il y aura redondance dans la liste $\rho^G_{\mathcal{M}}(l)$, car $\rho^G_{\mathcal{M}}$ est une fonction.

Si une formule n'est pas un littéral, elle est la conjonction ou la disjonction de deux formules plus petites. $\Phi = \Phi_1 \wedge \Phi_2$ ou $\Phi = \Phi_1 \vee \Phi_2$. On va traiter le cas disjonction, sachant que le raisonnement est similaire pour la conjonction. Par hypothèse de récurrence, on a bien $[\![\Phi_1]\!]_G = 1 \Rightarrow [\![\rho^G_{\mathcal{M}}(\Phi_1)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1$ et $[\![\Phi_2]\!]_G = 1 \Rightarrow [\![\rho^G_{\mathcal{M}}(\Phi_2)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1$. Donc on a $[\![\Phi]\!]_G = 1 \Rightarrow ([\![\Phi_1]\!]_G = 1 \vee [\![\Phi_2]\!]_G = 1) \Rightarrow ([\![\rho^G_{\mathcal{M}}(\Phi_1)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1 \vee [\![\rho^G_{\mathcal{M}}(\Phi_2)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1) \Rightarrow [\![\rho^G_{\mathcal{M}}(\Phi)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1$.

On conclue par le principe de récurrence. □

Théorème 2. *Soit $G = (N, E)$ un graphe et $\mathcal{M}$ un ensemble de noeuds malicieux, $\mathcal{Q}_{route}$ un protocole de découverte de route souple. On considère une $\mathcal{Q}_{route}$-configuration $K = (\mathcal{P}; \mathcal{I})$ pour G, $\mathcal{M}$, a et b honnêtes et la propriété $\mathsf{way}(x_L)$. S'il y a une $\mathcal{M}$-attaque sur K et $\neg\mathsf{way}(x_L)$ pour la topologie G, alors quelque soit la fonction de projection $\rho^G_{\mathcal{M}}$, il existe une $\rho^G_{\mathcal{M}}(\mathcal{M})$-attaque sur la $\mathcal{Q}_{route}$-configuration $\rho^G_{\mathcal{M}}(K)$ et $\rho^G_{\mathcal{M}}(G)$.*

Démonstration. Soit $G = (N, E)$ un graphe, $\mathcal{M} \subseteq N$ un ensemble fini de noeuds, a et b agents honnêtes, $\mathcal{Q}_{route}$ un protocole de découverte de route souple. On considère une $\mathcal{Q}_{route}$-configuration $K = (\mathcal{P}; \mathcal{I})$ pour G, $\mathcal{M}$, a et b honnêtes et la propriété $\mathsf{way}(x_L)$. On considère une fonction de projection $\rho^G_{\mathcal{M}}$. On considère le graphe projeté $\rho^G_{\mathcal{M}}(G) = (N', E')$ et $\rho^G_{\mathcal{M}}(K) = (\mathcal{P}_p; \mathcal{I}_p)$ qui est une configuration adéquate d'après le lemme 1. Par définition d'une $\mathcal{M}$-attaque, il existe une séquence de transitions :

$$K = (\lfloor P_i(a,b).\mathsf{if}\ \neg\mathsf{way}(x_L)\ \mathsf{then}\ \mathsf{out}(\mathsf{error})\rfloor_a \cup \mathcal{P}; \mathcal{I}) \ \rightarrow^*_{G,\mathcal{M}} \ (\lfloor \mathsf{out}(\mathsf{error})\rfloor_a \cup \mathcal{P}'; \mathcal{I}') = K'$$

On vérifie que

$$\rho^G_{\mathcal{M}}(K) \ \rightarrow^*_{\rho^G_{\mathcal{M}}(G), \rho^G_{\mathcal{M}}(\mathcal{M})} \ \rho^G_{\mathcal{M}}(K')$$

Plus précisément, on va montrer que si :

$$K = (\mathcal{P}; \mathcal{I}) \rightarrow_{G,\mathcal{M}} (\mathcal{P}'; \mathcal{I}') = K'$$

alors on a aussi que :

$$\rho^G_{\mathcal{M}}(K) \rightarrow_{\rho^G_{\mathcal{M}}(G), \rho^G_{\mathcal{M}}(\mathcal{M})} \rho^G_{\mathcal{M}}(K')$$

Par induction, on aura le résultat escompté.

S'il s'agit d'une règle Par, on a alors :

$$K = (\lfloor P_1 \mid P_2\rfloor_n \cup \mathcal{P}; \mathcal{I}) \quad \rightarrow_{G,\mathcal{M}} \quad (\lfloor P_1\rfloor_n \cup \lfloor P_2\rfloor_n \cup \mathcal{P}; \mathcal{I}) = K'$$

On montre maintenant que de la configuration projetée peut aussi effectuer la transition Par :

$$\begin{aligned}
&(\lfloor \rho^G_{\mathcal{M}}(P_1 \mid P_2)\rfloor_{\rho^G_{\mathcal{M}}(n)} \cup \rho^G_{\mathcal{M}}(\mathcal{P}); \rho^G_{\mathcal{M}}(\mathcal{I})) \\
&= (\lfloor \rho^G_{\mathcal{M}}(P_1) \mid \rho^G_{\mathcal{M}}(P_2)\rfloor_{\rho^G_{\mathcal{M}}(n)} \cup \rho^G_{\mathcal{M}}(\mathcal{P}); \rho^G_{\mathcal{M}}(\mathcal{I})) \\
&\rightarrow_{\rho^G_{\mathcal{M}}(G), \rho^G_{\mathcal{M}}(\mathcal{M})} (\lfloor \rho^G_{\mathcal{M}}(P_1)\rfloor_{\rho^G_{\mathcal{M}}(n)} \cup \lfloor \rho^G_{\mathcal{M}}(P_2)\rfloor_{\rho^G_{\mathcal{M}}(n)} \cup \rho^G_{\mathcal{M}}(\mathcal{P}); \rho^G_{\mathcal{M}}(\mathcal{I})) = K''
\end{aligned}$$

De plus, on note que $K'' = \rho^G_{\mathcal{M}}(K')$. Donc on a bien que :

$$\rho^G_{\mathcal{M}}(K) \ \rightarrow_{\rho^G_{\mathcal{M}}(G),\rho^G_{\mathcal{M}}(\mathcal{M})} \ \rho^G_{\mathcal{M}}(K')$$

S'il s'agit d'une règle Repl ou New, la correspondance est tout aussi naturelle.
En ce qui concerne la règle If-Then,

$$\begin{array}{l} K = (\lfloor \mathsf{if}\ \Phi\ \mathsf{then}\ P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \\ \quad \rightarrow_{G,\mathcal{M}} \quad (\lfloor P \rfloor_n \cup \mathcal{P}; \mathcal{I}) = K' \text{ si } [\![\Phi]\!]_G = 1 \end{array}$$

On a :

$$\rho^G_{\mathcal{M}}(K) = (\lfloor \mathsf{if}\ \rho^G_{\mathcal{M}}(\Phi)\ \mathsf{then}\ \rho^G_{\mathcal{M}}(P) \rfloor_{\rho^G_{\mathcal{M}}(n)} \cup \rho^G_{\mathcal{M}}(\mathcal{P}); \rho^G_{\mathcal{M}}(\mathcal{I}))$$

On a $[\![\Phi]\!]_G = 1$, et donc $[\![\rho^G_{\mathcal{M}}(\Phi)]\!]_{\rho^G_{\mathcal{M}}(G)} = 1$ d'après le lemme 2. De ce fait, on a aussi :

$$\rho^G_{\mathcal{M}}(K) \ \rightarrow_{\rho^G_{\mathcal{M}}(G),\rho^G_{\mathcal{M}}(\mathcal{M})} \ \rho^G_{\mathcal{M}}(K')$$

S'il s'agit d'une règle Comm, alors on a :

$$\begin{array}{l} (K = \{\lfloor \mathsf{in}\ u_j[\Phi_j].P_j \rfloor_{n_j} \mid \mathsf{mgu}(t, u_j) \neq \bot, [\![\Phi_j \sigma_j]\!]_G = 1, (n, n_j) \in E\} \\ \quad \cup \lfloor \mathsf{out}(t).P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \\ \quad\quad \rightarrow_{G,\mathcal{M}} \quad (\{\lfloor P_j \sigma_j \rfloor_{n_j}\} \cup \lfloor P \rfloor_n \cup \mathcal{P}; \mathcal{I}') = K' \end{array}$$

où $\sigma_j = \mathsf{mgu}(t, u_j)$, $\mathcal{I}' = \mathcal{I} \cup \{t\}$ si $(n, n_I) \in E$ pour un $n_I \in \mathcal{M}$ et $\mathcal{I}' = \mathcal{I}$ sinon.

On considère maintenant la configuration $\rho^G_{\mathcal{M}}(K)$:.

$$\begin{array}{l} (\{\lfloor \mathsf{in}\ \rho^G_{\mathcal{M}}(u_j)[\rho^G_{\mathcal{M}}(\Phi_j)].\rho^G_{\mathcal{M}}(P_j) \rfloor_{\rho^G_{\mathcal{M}}(n_j)} \mid \mathsf{mgu}(t, u_j) \neq \bot, [\![\Phi_j \sigma_j]\!]_G = 1, (n, n_j) \in E\} \\ \quad \cup \lfloor \mathsf{out}(\rho^G_{\mathcal{M}}(t)).\rho^G_{\mathcal{M}}(P) \rfloor_{\rho^G_{\mathcal{M}}(n)} \cup \rho^G_{\mathcal{M}}(\mathcal{P}); \rho^G_{\mathcal{M}}(\mathcal{I})) \end{array}$$

Dans ce cas, si $\mathsf{mgu}(t, u_j) \neq \bot$, alors on a aussi $\mathsf{mgu}(\rho^G_{\mathcal{M}}(t), \rho^G_{\mathcal{M}}(u_j)) \neq \bot$:
soit $\sigma_j = \mathsf{mgu}(t, u_j) = \{^{t_1}/_{x_1}, \ldots, ^{t_n}/_{x_n}\}$. On montre que $\rho^G_{\mathcal{M}}(\sigma) = \mathsf{mgu}(\rho^G_{\mathcal{M}}(t), \rho^G_{\mathcal{M}}(u_j))$.

En effet, $\rho^G_{\mathcal{M}}(t)\rho^G_{\mathcal{M}}(\sigma) = \rho^G_{\mathcal{M}}(t)\{^{\rho^G_{\mathcal{M}}(t_1)}/_{x_1}, \ldots, ^{\rho^G_{\mathcal{M}}(t_n)}/_{x_n}\} = \rho^G_{\mathcal{M}}(t\sigma)$ car σ a comme domaine des variables libres, et le domaine de $\rho^G_{\mathcal{M}}$ est les noms d'agents (terme clos). Donc on a :
$\rho^G_{\mathcal{M}}(t)\rho^G_{\mathcal{M}}(\sigma) = \rho^G_{\mathcal{M}}(u_j\sigma) = \rho^G_{\mathcal{M}}(u_j)\rho^G_{\mathcal{M}}(\sigma)$.
En nommant $\sigma'_j = \mathsf{mgu}(\rho^G_{\mathcal{M}}(t), \rho^G_{\mathcal{M}}(u_j))$, si on a $[\![\Phi_j \sigma_j]\!]_G = 1$, alors on a aussi $[\![\rho^G_{\mathcal{M}}(\Phi_j)\sigma'_j]\!]_{\rho^G_{\mathcal{M}}(G)}$, d'après le lemme 2
Si on a $(n, n_j) \in E$ on a aussi $(\rho^G_{\mathcal{M}}(n), \rho^G_{\mathcal{M}}(n_j)) \in \rho^G_{\mathcal{M}}(E)$, parce que la projection conserve les voisinages.

De ce fait, on peut effectuer la transition de la règle Comm, et on arrive à la configuration suivante :

$$\begin{array}{l} (\{\lfloor \rho^G_{\mathcal{M}}(P_j)\sigma'_j \rfloor_{\rho^G_{\mathcal{M}}(n_j)}\} \cup \lfloor \rho^G_{\mathcal{M}}(P) \rfloor_{\rho^G_{\mathcal{M}}(n)} \cup \rho^G_{\mathcal{M}}(\mathcal{P}); \mathcal{I}'') = K'' \end{array}$$

où $\sigma'_j = \mathsf{mgu}(\rho^G_{\mathcal{M}}(t), \rho^G_{\mathcal{M}}(u_j))$, $\mathcal{I}' = \rho^G_{\mathcal{M}}(\mathcal{I}) \cup \{\rho^G_{\mathcal{M}}(t)\}$ si $(\rho^G_{\mathcal{M}}(n), \rho^G_{\mathcal{M}}(n_I)) \in \rho^G_{\mathcal{M}}(E)$ pour un $\rho^G_{\mathcal{M}}(n_I) \in \rho^G_{\mathcal{M}}(\mathcal{M})$ et $\mathcal{I}'' = \rho^G_{\mathcal{M}}(\mathcal{I})$ sinon.

On peut noter que $\rho^G_{\mathcal{M}}(\mathcal{I}') = \mathcal{I}''$, car s'il existe $n_I \in \mathcal{M}$ tel que $(n, n_I) \in E$, alors il existe $\rho^G_{\mathcal{M}}(n_I) \in \rho^G_{\mathcal{M}}(\mathcal{M})$ tel que $(\rho^G_{\mathcal{M}}(n), \rho^G_{\mathcal{M}}(n_I)) \in \rho^G_{\mathcal{M}}(E)$. Dans ce cas, $\rho^G_{\mathcal{M}}(\mathcal{I}') = \rho^G_{\mathcal{M}}(\mathcal{I}) \cup \rho^G_{\mathcal{M}}(t)$. Sinon, s'il n'existe pas de $n_I \in \mathcal{M}$ tel que $(n, n_I) \in E$, alors il n'existe pas de n_J tel que $(\rho^G_{\mathcal{M}}(n), n_J) \in \rho^G_{\mathcal{M}}(E)$, car la projection conserve les voisinages.

De plus, on a que $\rho^G_{\mathcal{M}}(P_j\sigma_j) = \rho^G_{\mathcal{M}}(P_j)\sigma'_j$, car, on a $\mathsf{mgu}(\rho^G_{\mathcal{M}}(t), \rho^G_{\mathcal{M}}(u_j)) = \rho^G_{\mathcal{M}}(\mathsf{mgu}(t, u_j))$. De ce fait, on a $\rho^G_{\mathcal{M}}(K') = K''$, et on obtient la relation, il existe une transition Comm qui lie $\rho^G_{\mathcal{M}}(K)$ à $\rho^G_{\mathcal{M}}(K')$.

Finalement, s'il s'agit d'une règle In, alors on a :

$$\text{In} \quad (\lfloor \mathsf{in}\ u[\Phi].P \rfloor_n \cup \mathcal{P}; \mathcal{I}) \quad \rightarrow_{G,\mathcal{M}} \quad (\lfloor P\sigma \rfloor_n \cup \mathcal{P}; \mathcal{I})$$
$$\text{si } (n_I, n) \in E \text{ pour un } n_I \in \mathcal{M},\ \mathcal{I} \vdash t,\ \sigma = \mathsf{mgu}(t, u) \text{ et } [\![\Phi\sigma]\!]_G = 1$$

Le système d'inférence que l'on considère est indépendant des noms d'agents et donc on peut remarquer que si $\mathcal{I} \vdash t$, alors $\rho^G_{\mathcal{M}}(\mathcal{I}) \vdash \rho^G_{\mathcal{M}}(t)$. On peut ajouter à cela les mêmes raisons que pour la règle Comm, et donc, s'il y a une transition In entre K et K', alors il existe une transition In entre $\rho^G_{\mathcal{M}}(K)$ et $\rho^G_{\mathcal{M}}(K')$. □

A.1 Résultats pour SRP appliqué à DSR

Le tableau ci-dessous représente les résultats que donne ProVerif sur les différentes configurations testées.

Source	Destination	Noeuds Malicieux	Résultat
A	A	D	Pas d'attaque trouvée
A	B	D	Pas d'attaque trouvée
A	C	D	Pas d'attaque trouvée
B	A	D	attaque trouvée
B	B	D	attaque trouvée
A	A	D et C	Pas d'attaque trouvée
A	B	D et C	Pas d'attaque trouvée
B	A	D et C	attaque trouvée
B	B	D et C	attaque trouvée

Dans le tableau, pas d'attaque trouvée signifie que ProVerif n'arrive pas à montrer la sécurité du protocole dans ces cas là, mais échoue à trouver une dérivation. Dans le cas d'une attaque trouvée, ProVerif a réussi à exhiber une trace d'attaque.

On peut remarquer que des attaques sont trouvées dès que la source est B. Ceci est du au fait que B ne peut pas vérifier directement que A et C ne sont pas voisins, et accepte d'importe quelle liste comme étant un route valide. Le scénario est alors très simple : l'attaquant construit aisément une liste comprenant le A et C voisins.

Le code et les sorties complètes pour ce protocole peuvent être trouvées sur la page
`http://eleves.mines.inpl-nancy.fr/~degri2983/fichier/SRPtoDSR.html`

A.2 SDMSR : modélisation et résultats

A.2.1 modélisation

On modélise le protocole dans notre formalisme de la façon suivante : Le processus initiateur exécuté par la source S vers une destination D est :

$$P_{\mathsf{init}}(S, D) = \mathsf{new}\ id.\mathsf{out}(u_1).\mathsf{in}\ u_2[\Phi_S].0$$

$$\text{où} \begin{cases} u_1 = \langle \mathsf{req}, S, D, id, S :: [], [\![\langle \mathsf{req}, S, D, id\rangle]\!]_{\mathsf{priv}(S)}\rangle \\ u_2 = \langle \mathsf{rep}, D, S, id, x_A, x_L, [\![\langle \mathsf{rep}, D, S, id, x_L\rangle]\!]_{\mathsf{priv}(x_A)}\rangle \\ \Phi_S = \mathsf{check}(S, x_A) \wedge \mathsf{checkl}(S, x_L) \end{cases}$$

Les noms des noeuds intermédiaires sont accumulés dans la liste présente dans la requête. Les noeuds intermédiaires retransmettent la requête à travers le réseau, en vérifiant que la signature. Dans la spécification du vrai protocole, les noeuds ne retransmettent pas les requêtes comportant une liste plus grande que celle qu'ils ont vu. On considère que tous les paquets sont retransmis, pour simplifier la modélisation.

Ci dessous, $V \in \mathcal{N}_{\mathsf{loc}}$, x_S, x_a et x_D sont des variables de type loc tandis que x_r est une variable de type lists et x_{id} est une variable de type term. Le processus de transmission de requête exécuté par un noeud intermédiaire V est comme suit :

$$P_{\mathsf{req}}(V) = \mathsf{in}\ w_1[\Phi_V].\mathsf{out}(w_2)$$

$$\text{où} \begin{cases} w_1 = \langle \mathsf{req}, x_S, x_D, x_{id}, x_a :: x_r, [\![\langle \mathsf{req}, x_S, x_D, x_{id}\rangle]\!]_{\mathsf{priv}(x_S)}\rangle \\ \Phi_V = \mathsf{check}(V, x_a) \\ t = \langle x_S, x_D, x_{id}, x_a :: x_r\rangle \\ w_2 = \langle \mathsf{req}, x_S, x_D, x_{id}, V :: x_a :: x_r, [\![\langle \mathsf{req}, x_S, x_D, x_{id}\rangle]\!]_{\mathsf{priv}(x_S)}\rangle \end{cases}$$

Quand une requête atteint la destination D, elle vérifie que la requête provient d'un de ses voisins, contient une signature valide. Puis, la destination construit une requête réponse, en calculant une signature sur le route accumulée avec sa clé privée $\mathsf{priv}(D)$, et la renvoie sur le réseau.

Le processus exécuté par la destination D est $P_{\mathsf{dest}}(D) = \mathsf{in}\ v_1[\Phi_D].\mathsf{out}(v_2).0$ où :

$$\begin{aligned} v_1 &= \langle \mathsf{req}, x_S, D, x_{id}, x_b :: x_l, [\![\langle \mathsf{req}, x_S, D, x_{id}\rangle]\!]_{\mathsf{priv}(x_S)}\rangle \\ \Phi_D &= \mathsf{check}(D, x_b) \\ v_2 &= \langle \mathsf{rep}, D, x_S, x_{id}, D, D :: x_b :: x_l, [\![\langle \mathsf{rep}, D, x_S, x_{id}, D :: x_b :: x_l\rangle]\!]_{\mathsf{priv}(D)}\rangle \end{aligned}$$

Puis la réponse travers à nouveau le réseau vers S. Les noeuds intermédiaires vérifient que la signature dans la réponse convient, et qu'il s'agit d'un route possible, avant de la transférer. Chaque noeud remplace la signature dans le réponse par la sienne. Le processus exécuté par un noeud V pour transférer une réponse est la suivante :

$$P_{\mathsf{rep}}(V) = \mathsf{in}\ w'[\Phi'_V].\mathsf{out}(w'').0$$

$$\text{où} \begin{cases} w' = \langle \mathsf{rep}, x_D, x_S, x_{id}, x_a, x_r, [\![\langle \mathsf{rep}, x_D, x_S, x_{id}, x_r\rangle]\!]_{\mathsf{priv}(x_a)}\rangle \\ \Phi'_V = \mathsf{checkl}(V, x_r) \wedge \mathsf{check}(V, x_a) \\ w'' = \langle \mathsf{rep}, x_D, x_S, x_{id}, V, x_r, [\![\langle \mathsf{rep}, x_D, x_S, x_{id}, x_r\rangle]\!]_{\mathsf{priv}(V)}\rangle \end{cases}$$

A.2.2 résultat de ProVerif

Le tableau ci-dessous représente les résultats que donne ProVerif sur les différentes configurations testées.

Source	Destination	Noeuds Malicieux	Résultat
A	A	D	Pas d'attaque trouvée
A	B	D	Pas d'attaque trouvée
A	C	D	Pas d'attaque trouvée
B	A	D	attaque trouvée
B	B	D	attaque trouvée
A	A	D et C	Pas d'attaque trouvée
A	B	D et C	Pas d'attaque trouvée
B	A	D et C	attaque trouvée
B	B	D et C	attaque trouvée

Dans le tableau, pas d'attaque trouvée signifie que ProVerif n'arrive pas à montrer la sécurité du protocole dans ces cas là, mais échoue à trouver une dérivation. Dans le cas d'une attaque trouvée, ProVerif a réussi à exhiber une trace d'attaque.

On peut remarquer que des attaques sont trouvées dès que la source est B. Ceci est du au fait que B ne peut pas vérifier directement que A et C ne sont pas voisins, et accepte d'importe quelle liste comme étant un route valide. Le scénario est alors très simple : l'attaquant construit aisément une liste comprenant le A et C voisins.

C'est exactement le même type d'attaque que pour le protocole SRP appliqué à DSR, et les attaques se produisent dans les mêmes configurations.

Le code et les sorties complètes pour ce protocole peuvent être trouvées sur la page
`http://eleves.mines.inpl-nancy.fr/~degri2983/fichier/SDMSR.html`