Validation de BALADE Protocole de gestion de clés de groupe dans les réseaux ad hoc

Mohamed Salah Bouassida, Isabelle Chrisment, Olivier Festor

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Validation de BALADE
Protocole de gestion de clés de groupe dans les réseaux ad hoc

Mohamed Salah Bouassida, Isabelle Chrisment et Olivier Festor

N° 5896

Avril 2006

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5896--FR

Validation de BALADE
Protocole de gestion de clés de groupe dans les réseaux ad hoc

Mohamed Salah Bouassida, Isabelle Chrisment et Olivier Festor

Thème 1 — Réseaux et systèmes
Project Madynes

Research Report n° 5896 — Avril 2006 — 71 pages

Résumé : Dans ce rapport, nous validons notre protocole de gestion de clés de groupe multicast, dénommé BALADE, afin de détecter les éventuelles failles de sécurité qui peuvent exister. Notre approche de validation suit trois étapes essentielles : (1) spécification des sous protocoles de BALADE de manière générale, (2) spécification des scénarii de ces protocoles en HLPSL, et finalement (3) vérification de ces scénrii avec l'outil AVISPA.

Mots-clés : BALADE, validation, HLPSL, AVSIPA

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Validation of BALADE

Abstract: In this report, we validate our group key management protocol, called BALADE, in order to detect the possible security faults, which can exist. Our validation approach follows three essential steps: (1) specifying the sub-protocols of BALADE, (2) specifying scenarios of these protocols in HLPSL, and finally (3) checking these scenarios with the AVISPA tool.

Key-words: BALADE, validation, HLPSL, AVSIPA

Table des matières

Chapitre 1

Introduction

Pour garantir la sécurité des communications de groupe dans les réseaux ad hoc, nous avons conçu un protocole de gestion de clés de groupe, dénommé BALADE [BCF05a], dédié aux réseaux mobiles ad hoc. Pour vérifier la robustesse et l'applicabilité de notre protocole, nous avons entrepris sa validation afin de détecter les éventuelles failles qu'un intrus peut exploiter pour pirater des données ou compremettre des membres du groupe multicast.

Notre approche de validation suit les trois étapes suivantes :

1. **Spécifier les sous protocoles de BALADE de manière générale**

   Cette étape consiste à spécifier d'un façon complète notre protocole BALADE, à commencer par les entités qui participent au fonctionnement de BALADE, les messages échangés par ces entités, les connaissances que peut avoir chaque entité (statique ou dynamique), pour aboutir à la spécification des sous protocoles de BALADE suivants :

   - Initialisation de BALADE
   - Renouvellement périodique de la clé de chiffrement de données TEK
   - Authentification et contrôle d'accès
   - Join et leave d'un nouveau membre
   - Ré-intégration d'un membre du groupe
   - Exclusion d'un membre du groupe
   - Envoi périodique des CL_Queries
   - Gestion des listes ACL et RL
   - Retransmission de la TEK par les contrôleurs de BALADE
   - Clusterisation OMCT

2. **Spécifier des scénarii de ces protocoles en HLPSL**

   Dans cette étape, nous spécifions les différents sous protocoles définis à la première étape, à l'aide du langage de spécification HLPSL [CCC+04]. Ce langage fournit un

haut niveau d'abstraction et offre plusieurs interfaces requises pour la spécification de protocoles de sécurité à large déploiement dans Internet.

3. **Vérifier les spécifications avec l'outil AVISPA (CL-ATSE)**

   Cette dernière étape consiste à utiliser l'outil AVISPA pour vérifier les différents sous protocoles de BALADE. AVISPA se distingue des outils existants par les nouvelles fonctionnalités qu'il intègre. Tout d'abord, il propose une analyse entièrement automatique. Ensuite, il recouvre un large spectre de protocoles et peut notamment prendre en compte certaines propriétés des opérateurs cryptographiques. Enfin, il combine plusieurs techniques de vérification de protocoles. Ces différentes techniques reposent soit sur la vérification de modèles, soit sur la résolution de contraintes extraites par exploration «symbolique» des protocoles. Avec la méthode de résolution de contraintes il s'agit de tester si l'intrus peut décrypter un message avec les informations qu'il a en sa possession, ou qu'il peut acquérir en se faisant passer pour un autre agent.

Ce rapport est structuré de la façon suivante. Le chapitre 2 présente la spécification informelle des sous protocoles de BALADE. Le chapitre 3 aborde quelques exemples de scénarii de sous protocoles de BALADE, écrits en HLPSL. Le chapitre 4 expose quelques exemples de vérifications et validations de sous protocoles via l'outil de validation AVISPA. L'ensemble des spécifications des différents sous protocoles de BALADE, ainsi que leur validation avec AVISPA, est présenté dans l'annexe.

Chapitre 2

Spécification des sous protocoles de BALADE

2.1 Contexte

BALADE est un protocole de gestion de clés de groupe dans les réseaux ad hoc. Il est composé de différents sous protocoles, formés chacun d'un ensemble de messages et de connaissances échangés, et faisant intervenir différentes entités du groupe.

2.1.1 Notation adoptée

Dans ce qui suit, nous présentons le glossaire des acronymes que nous utilisons tout au long de ce rapport :

- G_k : désigne le groupe multicast k
- ACL : liste de contrôle d'accès (*Access Control List*)
- RL : liste de révocation (*Revocation List*)
- LML : liste des membres locaux
- SG_{ik} : sous groupe i appartenant au groupe k
- GCL : Groupe des contrôleurs locaux
- TEK : clé de chiffrement des données du groupe (*Traffic Encryption Key*)
- KEK_CSG_{ik} : clé locale du sous groupe i du groupe k, jouant le rôle d'une clé de chiffrement de la clé TEK (*Key Encryption Key*).
- KEK_CCL : clé du groupe des contrôleurs locaux, jouant également le rôle d'une clé de chiffrement de la clé TEK.
- CBID$_mg_{ik}$: identificateur cryptographique (*Crypto Based IDentifier*) du membre mg_{ik}

- Pub_mg_{ik} : clé publique du membre mg_{ik}
- Pri_mg_{ik} : clé privée du membre mg_{ik}
- Coord_nœud : les coordonnées GPS du nœud.
- $\rightarrow$: message envoyé en unicast
- $\Rightarrow$: message envoyé en multicast
- $\Rrightarrow$: message envoyé en broadcast

2.1.2 Entités

Nous distinguons trois types de participants aux différents sous protocoles de BALADE. Le premier type de participant correspond à un membre ordinaire du groupe multicast. Le deuxième type correspond au contrôleur global de l'ensemble du groupe multicast. Et finalement, le dernier type d'acteurs de BALADE est le contrôleur local, qui a à sa charge d'assurer la sécurité au sein d'un sous-groupe ou cluster.

- mg_k : désigne un membre du groupe k,
- mg_{ik} : désigne un membre du groupe k, appartenant au cluster i,
- CG_k : désigne le contrôleur global du groupe k,
- CL_{ik} : désigne le contrôleur local du cluster i appartenant au groupe k,
- *Admin* : désigne l'administrateur du groupe multicast, responsable de la création et de l'initialisation de la session du groupe.

2.1.3 Messages

Les messages échangés entre les différentes entités de BALADE seront définis dans la spécification des différents sous protocoles, en utilisant la notation suivante :

- $\{m\}k$: désigne le message m crypté avec la clé k.
- (m_1, m_2) : désigne la concaténation de deux messages m_1 et m_2.

2.1.4 Connaissances

Nous distinguons pour chaque entité de BALADE des connaissances statiques, invariables au cours de la session, et des connaissances dynamiques.

Connaissances statiques

Tout nœud du réseau mg détient une clé publique Pub_mg et une clé privée Pri_mg, qu'il a générées de façon autonome, ou qu'il a reçu d'une autorité de certification hors-ligne. Pour assurer l'authentification des membres et des sources dans le cadre de BALADE, tout membre du réseau mg construit son identificateur unique CBID_mg, à partir de ses clés publique et privée, de la façon suivante [MC02]:

$$CBID\_mg = hmac\_sha1\_128(sha1(imprint), sha1(Pub\_mg))$$

avec imprint un entier de 8 octets qui peut être choisi aléatoirement.

Les identifiants cryptographiques sont statistiquement uniques et cryptographiquement vérifiables, ce qui signifie que de par leur nature, il est très peu probable que deux entités aient le même identifiant, et qu'il est possible de vérifier la validité de l'identifiant présenté par une entité grâce à des techniques cryptographiques. L'idée de base de ces identificateurs est d'avoir une forte liaison cryptographique avec leurs composants (clés privée et publique). C'est exactement le but des certificats. Cette technique d'authentification ne doit pas compter sur une tierce partie, à savoir une PKI globale, ou un serveur central de distribution de clés. Ceci implique que deux entités qui ne se connaissent pas ne peuvent pas communiquer. L'authentification d'un nouveau nœud n'est donc pas possible. Seuls les nœuds qui se connaissent au préalable peuvent s'identifier et communiquer.

Durant une session de communication multicast, tout membre mg connaît également les identificateurs des groupes multicast auxquels il est adhérent.

Connaissances dynamiques

Tout membre du groupe k, noté mg_k, connaît son contrôleur global actuel CG_k, qui est la source du flux multicast à l'instant courant. Tout membre mg_k connaît également l'identificateur i de son sous-groupe SG_{ik}, ainsi que son contrôleur local CL_{ik}.

La clé de chiffrement de données TEK est distribuée par le contrôleur global du groupe à tous ses membres. En plus, chaque membre mg_{ik} connaît la clé de chiffrement de clé de son sous-groupe (clé locale) KEK_CSG_{ik}.

Tous les contrôleurs (global et locaux) connaissent à tout moment la liste de leurs membres locaux, appelée LML. Ils coopèrent et collaborent également pour gérer la liste de contrôle d'accès ACL et celle des membres exclus RL.

2.2 Sous protocoles de BALADE

2.2.1 Initialisation de BALADE

Mise en place de l'ACL

La liste de contrôle d'accès d'un groupe multicast est construite hors ligne, et contient tous les membres autorisés à joindre ce groupe, dès le début de la session ou ultérieurement. C'est l'administrateur qui est responsable de construire cette ACL et de l'envoyer au premier contrôleur global du groupe correspondant. Cet administrateur n'a aucune charge en ligne, c'est à dire que la gestion et la sécurité des membres du groupe est assurée par les membres eux même sans aucune entité centrale veillante.

Génération et distribution de la TEK

TEK Distribution
$\forall mg_k,\ CG_k \longrightarrow mg_k : \{TEK, Num\_Seq, KEK\_CSG_{0k}, IDG, ID_{CG}, Pub\_CG,$ $imprint, \{CBID - CG\}Pri\_CG\}Pub\_mg_k$

avec :

1. TEK : clé de chiffrement de trafic,
2. Num_Seq : Numéro de séquence correspondant à la clé de chiffrement de données TEK,
3. KEK_CSG_{0k} : clé locale du cluster 0,
4. IDG : identité du groupe,
5. ID_{CG} : identité du CG, qui peut être son adresse IP,
6. CBID-CG : CBID du CG ,
7. Pub_mg_k : clé publique du membre mg_k

Pour améliorer la fiabilité de la distribution de la clé de chiffrement de données TEK, un numéro de séquence correspond à chaque TEK. Ce même numéro de séquence sera ajouté également au flux multicast chiffré par cette TEK. Num_Seq est un entier à un octet, incrémenté à chaque renouvellement de la TEK. Ainsi, un membre qui ne reçoit pas la clé de chiffrement de données lors du renouvellement de la TEK, à cause d'un problème de connexion, peut détecter que la TEK qu'il détient ne correspond pas au flux chiffré qu'il reçoit, et demander ainsi à son contrôleur local de la lui re-transmettre (cf 2.2.9).

Pour s'identifier auprès des membres du groupe, le contrôleur global envoie son CBID chiffré avec sa clé privée, ainsi que sa clé publique et l'entier imprint qu'il a utilisé pour générer son CBID. Chaque membre du groupe recevant ce message commence par vérifier l'authenticité du CG en déchiffrant son CBID et le comparant avec le CBID calculé via la clé publique et l'entier imprint du CG.

En cas de succès, le membre du groupe stocke la clé de chiffrement des données, le numéro de séquence et la clé locale de son sous-groupe 0.

Connaissances initiales

CG_k	$IDG, ID_{CG}, TEK, Num\_Seq, \forall Pub\_mg_k$
$\forall mg_k$	$Pub\_mg_k, Pri\_mg_k, IDG, ID_{CG}$

2.2.2 Renouvellement périodique de la TEK

On se place dans le cadre où le processus de clusterisation a été enclenché (voir section 2.2.8), dû à ce qu'un nombre important de membres ont joint le groupe multicast, avec une

faible cohésion. La clusterisation divise le groupe multicast en des clusters, gérés chacun par un CL responsable de la gestion et de la sécurité de son sous-groupe.

Le renouvellement de la clé de chiffrement de trafic TEK est enclenché après chaque unité sémantique des données multicast, et suit le processus de distribution suivant :

TEK Renewal
$\forall mg_{0k} \in LML\_CG$:
$CG_k \Rightarrow mg_{0k} : \{ID_{CG}, TEK, Num\_Seq\} KEK\_CSG_{0k}$
$\forall CL_{ik} \in GLC$:
$CG_k \Rightarrow CL_{ik} : \{ID_{CG}, TEK, Num\_Seq\} KEK\_CCL$
$\forall CL_{ik} \in GCL, \forall mg_{ik} \in LML\_CL_{ik}$:
$CL_{ik} \Rightarrow mg_{ik} : \{ID_{CL}, TEK, Num\_Seq\} KEK\_CSG_{ik}$

avec :

1. ID_{CG} : identité du CG
2. ID_{CL} : identité du CL

On note que ce processus de distribution de clé reste valable dans le cas où le groupe multicast est constitué d'un seul cluster, géré par le CG.

Connaissances initiales

CG_k	$IDG, ID_{CG}, TEK, Num\_Seq, KEK\_CSG_{0k}, KEK\_CCL$
$\forall CL_i$	IDG, ID_{CG}, KEK_CCL
$\forall mg_{ik}$	$Pub\_mg_{ik}, Pri\_mg_{ik}, IDG, ID_CG, KEK\_CSG_{ik}$

2.2.3 Ajout d'un nouveau membre (Procédure Join)

Quand un nouveau membre mg_k veut joindre le groupe multicast, il doit s'authentifier auprès d'un contrôleur local CL_{ik} du groupe qui vérifie en plus s'il est autorisé ou non à joindre le groupe. Le nouveau membre peut connaître le contrôleur local CL_{ik} en recevant un message CL_Query de la part de CL_{ik}, l'informant de sa localisation géographique et de l'identité du groupe multicast pour lequel il joue le rôle d'un CL (cf section 2.2.7).

Dans le cas où un nouveau membre ne connaît aucun contrôleur local auprès duquel il peut entamer une procédure de Join, il envoie une requête en broadcast à tous les membres du réseau, cherchant à connaître un contrôleur. Le premier membre du groupe qui reçoit cette requête lui répond avec un message contenant l'identité de son contrôleur local.

Dans ce qui suit, nous considérons qu'un nouveau membre connaît l'identité du CL du cluster auquel il veut adhérer.

Node Authentication and Access Control
$mg_k \longrightarrow CL_{ik} : Pub\_mg_k, imprint, \{CBID\_mg_k\}Pri\_mg_k$

Le membre envoie sa clé publique et l'entier imprint qu'il a utilisé pour générer son identificateur CBID. Le contrôleur local, ayant reçu ce message, commence par authentifier le membre, en déchiffrant le CBID à l'aide de la clé publique, ensuite en calculant de nouveau le CBID via la clé publique et l'entier imprint. En cas de succès, le CL vérifie si le membre est bien autorisé à joindre le groupe multicast, via l'ACL du groupe, gérée par le groupe des contrôleurs locaux (cf section 2.2.8). À ce stade, si le contrôle d'accès réussit, la demande d'adhésion du nouveau membre est acceptée au sein du cluster concerné, et ce membre nommé mg_{ik}, sera ajouté à la liste des membres locaux LML du CL_{ik}.

Pour faire face aux rejeux de messages et aux attaques de déni de service (attaques DoS), le contrôleur local doit demander à un nouveau membre, voulant joindre le groupe, de répondre à un puzzle (puzzle-request), avant d'entamer la vérification de son authentification et de son contrôle d'accès.

Node Authentication and Access Control
$mg_k \longrightarrow CL_{ik} : Join - Request, Pub\_mg_k$
$CL_{ik} \longrightarrow mg_k : Puzzle - Request$
$mg_k \longrightarrow CL_{ik} : Puzzle - Reply, Pub\_mg_k, imprint, \{CBID\_mg_k\}Pri\_mg_k$

Un puzzle-request est efficace s'il a les propriétés suivantes [ANL01] :

1. la création du puzzle et la vérification ne sont pas coûteuses pour le contrôleur local,
2. le coût pour résoudre le puzzle est facile à ajuster,
3. le puzzle peut être calculé par des nœuds de faible capacité de calcul,
4. le résultat du puzzle ne peut pas être pré-calculé à l'avance.

Un exemple de puzzle-request est donné dans [ANL01], et est basé sur une fonction à sens unique (*one way function*)(MD5 ou SHA). Le Puzzle-Request envoyé par le serveur est une valeur aléatoire N_s, et un niveau de difficulté k. Pour répondre au puzzle, le client génère une valeur aléatoire N_c, et calcule X et Y, de sorte que :

$$(i) \quad h(C, N_s, N_c, X) \; = \; 0_1 0_2 ... 0_k \; Y$$

h étant une fonction cryptographique de hachage, et C étant l'identité du client. Le Puzzle-Reply contient donc l'identité du client C, les deux valeurs aléatoires N_s et N_c, ainsi que la solution X. Pour valider cette réponse, le serveur n'a besoin que de vérifier l'égalité (i).

L'ajout d'un nouveau membre au sein d'un cluster nécessite le renouvellement de la clé locale de ce cluster, et sa distribution à tous les membres locaux, y compris le nouvel

adhérent, comme cela est illustré dans le tableau suivant. Un mot de passe est également envoyé au nouveau membre, pour pouvoir l'utiliser en cas de ré-intégration (voir la section 2.2.4).

Join Procedure
$for\ ancien\_mg_{ik} : anciens\ membres\ du\ cluster$
$CL_{ik} \Rightarrow ancien\_mg_{ik} : \{ID_{CL}, KEK\_CSG_{ik}\} ancienne\_KEK\_CSG_{ik}$
$mg_{ik} :\ nouveau\ membre$
$CL_{ik} \longrightarrow mg_{ik} : \{ID_{CL}, TEK, KEK\_CSG_{ik}, Passwd\} Pub\_mg_{ik}$

Connaissances initiales

CL_{ik}	$ID_{CL}, TEK, KEK\_CSG_{ik}, ancienne\_KEK\_CSG_{ik}, Passwd$
mg_{ik}	$Pub\_mg_{ik}, Pri\_mg_{ik}, imprint, CBID_{m}g_{k}$

2.2.4 Ré-intégration d'un membre du groupe

Un membre du groupe mg_k qui se déplace dans le réseau ad hoc, peut perdre sa connectivité vers son cluster et essayer ensuite de rejoindre le groupe multicast. Un membre ad hoc peut également perdre son appartenance au groupe multicast à cause de problèmes de ressources (batterie), et essayer de le ré-intégrer ensuite.

Dans ces deux cas d'utilisation, pour éviter de refaire la procédure de Join présentée dans 2.2.3, l'ancien membre $ancien\_mg_k$ peut ré-intégrer son groupe en envoyant un message de ré-authentification à n'importe quel membre du groupe mg_{ik}, via le ticket de ré-authentfication, comme suit :

Re – authentication Procedure
$ancien\_mg_k \longrightarrow mg_{ik} : Pub\_ancien\_mg_k, CBID\_ancien\_mg_k, \{\{Passwd\}TEK\} Pri\_ancien\_mg_k$
$mg_{ik} \longrightarrow ancien\_mg_k : \{KEK\_CSG_{ik}\} Pub\_ancien\_mg_k$

Le membre mg_{ik} peut être un contrôleur local du groupe multicast, connu via les messages CL_Queries envoyés périodiquement par tous les contrôleurs locaux (cf section 2.2.7). Si l'ancien membre $ancien\_mg_k$ ne connaît aucun membre du groupe (ou aucun contrôleur local), il envoie un message en broadcast à tous les membres du réseau. Le premier membre du groupe recevant cette requête, lui répond avec son identité et l'identité de son contrôleur local.

Le ticket de ré-authentification est un mot de passe connu par tous les membres du groupe, et qui doit être envoyé, chiffré avec la clé de chiffrement de données courante (TEK). Dès que le membre du groupe mg_k reçoit le message de ré-authentification de $ancien\_mg_k$,

il vérifie si le ticket est valable, c'est à dire si le mot de passe est bien chiffré avec la clé de chiffrement de données courante. Le double chiffrement du mot de passe est nécessaire pour assurer la non-répudiation.

Cette première version du sous protocole de réintégration n'est pas valide, puisqu'elle permet à un intrus de pouvoir intercepter {Passwd}TEK, et par suite pouvoir récupérer la clé locale du sous groupe KEK_CSG_{ik}, et la clé de chiffrement de données TEK. Cette faille a été trouvée par l'outil de validation AVISPA. La nouvelle version de ce sous protocole (validée par AVISPA) est la suivante :

Re authentication Procedure
$ancien\_mg_k \longrightarrow mg_{ik} : Pub\_ancien\_mg_k, Imprint, \{CBID\_ancien\_mg_k\}Pri\_ancien\_mg_k,$ $\{\{Passwd\}TEK\}Pub\_mg_{ik}$
$mg_{ik} \longrightarrow ancien\_mg_k : \{KEK\_CSG_{ik}\}Pub\_ancien\_mg_k$

Connaissances initiales

$ancien\_mg_k$	$Pub\_ancien\_mg_k, Pri\_ancien\_mg_k, CBID\_ancien\_mg_k, Imp, Pub\_mg_{ik}, Passwd, TEK$
mg_{ik}	$Pub\_mg_{ik}, Passwd, TEK, KEK\_CSG_{ik}$

2.2.5 Départ d'un membre du groupe (Procédure Leave)

Une demande de quitter le groupe Leave est toujours acheminée vers le contrôleur local du cluster concerné.

Node Authentication
$mg_k \longrightarrow CL_{ik} : Pub\_mg_k, imprint, \{CBID\_mg_k\}Pri\_mg_k$

Comme dans le cas de Join, le CL procède à l'authentifiction du membre voulant quitter le groupe multicast, ensuite, et en cas de succès, il enlève le membre sortant de sa liste LML, et enclenche un processus de renouvellement de la clé locale du cluster, comme suit :

Leave Procedure
$mg_{ik}\_sortant$ *est le membre quittant le groupe* *for* mg_{ik} : *membre local,* $mg_{ik} \neq mg_{ik}\_sortant$ $CL_{ik} \longrightarrow mg_{ik} : \{ID_{CL}, KEK\_CSG_{ik}\}Pub\_mg_{ik}$

avec ID_{CL} : identité du CL.

Connaissances initiales

$mg_k\_sortant$	$Pub\_mg_k, Pri\_mg_k, imprint, CBID\_mg_k$
CL_{ik}	ID_CL, KEK_CSG_{ik}
$\forall mg_{ik}\_restant$	Pri_mg_{ik}

Leave d'un contrôleur local du groupe multicast

Lorsqu'un contrôleur local veut quitter le groupe multicast, il demande à ses membres locaux de joindre d'autres clusters, dans une durée maximale notée Merge_Timer. Le choix des nouveaux clusters est présenté en 2.2.7. Ensuite, il envoie un message leave au groupe des contrôleurs locaux, qui procèdent au changement de la clé de leur groupe KEK_CCL (par le contrôleur global)

Merge
$\forall mg_{ik}, CL_{ik} \Rightarrow mg_{ik} : \{ID\_Cluster, LML\_CL_{ik}\}KEK\_CSG_{ik}$
Notification_Départ
$CL_{ik} \Rightarrow GCL : \{ID\_CL_{ik}\}KEK\_CCL$ $\forall j \neq i,\ CG_k \Rightarrow CL_{jk} : \{ID\_CG, new\_KEK\_CCL\}Pub\_CL_{jk}$

Dans le cas où un contrôleur local quitte le groupe sans notifier de son départ (à cause d'un problème de batterie par exemple), ses membres locaux vont se rendre comte après un certain délai qu'il ne peuvent plus recevoir la clé de chiffrement de données émise par la source. Ils joindront donc un autre contrôleur local (voir 2.2.7).

2.2.6 Exclusion d'un membre du groupe

La révocation ou l'exclusion d'un membre du groupe est réalisée quand un membre mettant la sécurité du groupe en danger, ou ne respectant pas les consignes du bon déroulement de l'application multicast en question, est détecté[1].

Dans ce cas, le contrôleur local ajoute le membre exclu à la liste de révocation RL, et l'enlève de la liste LML de ses membres locaux. Ensuite, il enclenche un processus de renouvellement de la clé locale de son cluster, comme est décrit dans le cas de départ volontaire. De plus, la révocation d'un membre du groupe, nécessite en plus, le renouvellement total de la clé de chiffrement du trafic TEK, comme cela est défini dans la section 2.2.2. Le CL envoie ainsi une demande de renouvellement de la clé TEK au contrôleur global.

1. La présence d'un mécanisme de détection de comportements malicieux ou égoïstes est assumée au sein du réseau ad hoc

Node Exclusion
$mg_{ik}\_sortant$ *est le membre exclu du groupe* *for* mg_{ik} : *membre local,* $mg_{ik} \neq mg_{ik}\_sortant$ $CL_{ik} \longrightarrow mg_{ik} : \{ID_{CL}, KEK\_CSG_{ik}\}Pub\_mg_{ik}$
$CL_{ik} \longrightarrow CG_k : \{ID_{CL}, Pub\_CL_k, imprint, \{CBID\_CL_k\}Pri\_CL_k\}$
$\forall mg_{0k} \in LML\_CG$: $CG_k \Rightarrow mg_{0k} : \{ID_{CG}, TEK, Num\_Seq\}KEK\_CSG_{0k}$ $\forall CL_{ik} \in GLC$: $CG_k \Rightarrow CL_{ik} : \{ID_{CG}, TEK, Num\_Seq\}KEK\_CCL$ $\forall CL_{ik} \in GCL, \forall mg_{ik} \in LML\_CL_{ik}$: $CL_{ik} \Rightarrow mg_{ik} : \{ID_{CL}, TEK, Num\_Seq\}KEK\_CSG_{ik}$

Connaissances initiales

$\forall mg_{ik} \neq mg_{ik}\_sortant$	Pub_mg_{ik}
CL_{ik}	$ID\_CL, KEK\_CSG_{ik}, \forall Pub\_mg_{ik}$
CG_k	$IDG, ID_{CG}, TEK, Num\_Seq, KEK\_CSG_{0k}, KEK\_CCL$
$\forall CL_i$	IDG, ID_{CG}, KEK_CCL
$\forall mg_{ik}$	$Pub\_mg_{ik}, Pri\_mg_{ik}, IDG, ID_CG, KEK\_CSG_{ik}$

2.2.7 Envoi périodique des CL_Queries

Périodiquement, et à chaque intervalle noté CL_Query_Interval, chaque contrôleur local diffuse un message nommé CL_Query, avec un TTL=2. Le CL_Query a la forme suivante :

CL_Query
$CL_{ik} \Rrightarrow (TTL = 2) : \{ID\_CL_{ik}, Pub\_CL_{ik}, imprint, \{CBID\_CL_{ik}\}Pri\_CL_{ik}, Coord\_CL_{ik}\}$

Connaissances initiales

CL_{ik}	$ID\_CL_{ik}, Pub\_CL_{ik}, imp, CBID\_CL_{ik}, Coord\_CL_{ik}$

La diffusion des CL_Queries se fait à deux sauts, et ce afin de permettre à tous les membres du groupe multicast de connaître tous les contrôleurs locaux à deux sauts de leur

localisation géographique, et ainsi de choisir de rejoindre le cluster dont le CL est le plus proche d'eux en cas de déplacement ou de perte de connectivité. En effet, tout membre du groupe a seulement besoin d'avoir une connaissance restreinte de la localisation des contrôleurs locaux, qu'il pourrait éventuellement joindre (en cas de perte de connectivité avec son contrôleur courant).

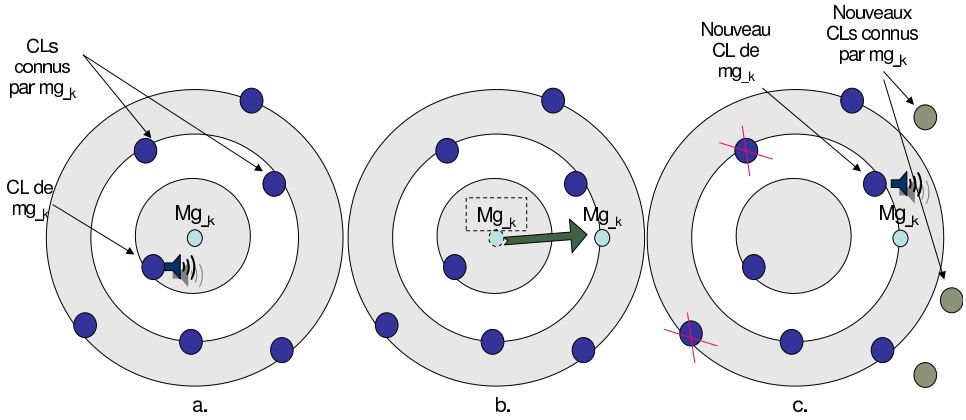

FIG. 2.1 – *Envoi périodique des CL_ Queries à 2 sauts*

Le cas d'utilisation présenté dans la figure 2.1 illsutre l'efficacité du choix de l'envoi des CL_Queries avec un TTL=2, nécessaire et suffisant pour assurer un bon fonctionnement de BALADE sans avoir recours à un broadcast couteûx vers tous les membres du réseau.

Dans la figure 2.1.a, le membre du groupe mg_k connaît son contrôleur local (à un saut) et les autres contrôleurs locaux à deux sauts. Au cours de son déplacement (figure 2.1.b), ce membre perd sa connectivité avec son contrôleur local et va essayer de joindre un autre contrôleur local le plus proche de sa localisation géographique. Dans la figure 2.1.c, mg_k connaît son nouveau contrôleur local, et les nouveaux contrôleurs locaux à deux sauts, et perd la connaissance des contrôleurs locaux qui ne sont plus à deux sauts de lui.

2.2.8 Gestion des listes ACL et RL

L'objectif de cette gestion est de pouvoir organiser une coopération entre les contrôleurs de BALADE, partager les listes ACL et RL entre ces nœuds, et en assurer la maintenance, la disponibilité, l'accessibilité et la cohérence[2]. Deux principaux algorithmes ont été mis en

2. Ce travail a été réalisé par des étudiants de l'École Supérieure d'Informatique et d'Applications de Lorraine (ESIAL), dans le cadre de leur projet 2A

place, l'algorithme de recherche de l'information, et l'algorithme de maintenance et de mise à jour de l'information.

- Algorithme de recherche de l'information des ACLs, permettant à tout contrôleur local de pouvoir demander l'autorisation d'un nouveau membre du groupe au nœud qui détient l'entrée ACL correspondante.
- Algorithme de maintenance et de mise à jour de l'information, permettant d'ajouter ou de supprimer des entrées dans l'ACL distribuée, tout en assurant sa cohérence et sa disponibilité. Une redondance des entrées est donc requise pour pouvoir faire face aux problèmes de ressources limitées et de perte de connectivité dans un réseau ad hoc. Considérons n : nombre de membres du groupe multicast, k : nombre des contrôleurs locaux du groupe, et f : nombre de redondances requises par la politique de gestion de l'application concernée. Le nombre d'entrées que chaque contrôleur local doit stocker dans son fichier ACL est donc : $f * \frac{n}{k}$.

Les requêtes que peut émettre un contrôleur local concernant les deux listes ACL et RL sont la demande d'accès d'un nœud, et/ou la demande d'ajout ou de suppression d'une entrée dans une liste. Ces requêtes sont envoyées vers le groupe des contrôleurs locaux GCL, chiffrées avec leur clé locale KEK_CCL.

Un système de gestion de confiance peut être mis au point entre les contrôleurs du groupe multicast. Initialement, c'est le contrôleur du groupe CG qui détient les deux listes ACL et RL. Ensuite, au cours de la création des contrôleurs locaux, le CG attribue à chaque contrôleur local un niveau de confiance lui permettant de mettre à jour les deux listes ou bien de seulement les consulter.

Dans ce qui suit, et pour des raisons de simplification, nous considérons que tout contrôleur local a le droit de consulter et de modifier les deux listes ACL et RL.

Demande d'autorisation d'accès
$CL_k \Rightarrow GCL : \{ID\_CL_k, mg_{new}\}KEK\_CCL$
$CL_{rk} \Rightarrow CL_k : \{ID\_CL_{rk}, ID\_CL_k, mg_{new}, Acceptation\}KEK\_CCL$

avec : mg_{new} : nouveau membre voulant joindre le groupe, et CL_{rk} est le contrôleur local qui détient l'information recherchée et qui répond à la requête d'autorisation d'accès. Si le nouveau membre n'est pas autorisé à joindre le groupe, i.e : il n'existe pas d'entrée ACL correspondante à ce membre, le contrôleur local en question CL_k refuse sa demande d'adhésion au groupe, après l'expiration d'un délai noté Join_Request_Timer.

Ajouter entrée
$CL_k \Rightarrow GCL : \{ID\_CL_k, mg_{à\_ajouter}, Liste\}KEK\_CCL$

avec : $mg_{à\_ajouter}$: membre à ajouter dans la liste ACL ou RL.

Supprimer entrée
$CL_k \Rightarrow GCL : \{ID\_CL_k, mg_{à\_supprimer}, Liste\} KEK\_CCL$

avec : $mg_{à\_supprimer}$: membre à supprimer de la liste ACL ou RL.

Idée : *Combiner les deux listes ACL et RL (un membre exclu sera supprimé de la liste ACL), l'avantage est le gain en mémoire et la simplicité, l'inconvénient est de ne plus pouvoir refuser l'accès d'un membre dès qu'il appartient à la liste RL et de devoir chercher son accès dans le fichier ACL qui est plus grand en mémoire.*

2.2.9 Retransmission de la TEK par les contrôleurs de BALADE

Pour améliorer la fiabilité de la transmission de la TEK, et assurer la synchronisation entre la distribution de la clé de chiffrement de données et la diffusion des données chiffrées, nous utilisons les numéros de séquence, comme expliqué dans 2.2.1. Quand un membre se rend compte qu'il ne détient pas la clé de chiffrement de données appropriée, il la demande auprès de son contrôleur local, comme suit :

TEK Retransmission
$mg_{ik} \longrightarrow CL_{ik} : \{ID\_mg_{ik}\} KEK\_CSG_{ik}$
$\forall mg_{ik}, CL_{ik} \Rightarrow mg_{ik} : \{ID\_CL, TEK, Num\_Seq\} KEK\_CSG_{ik}$

Connaissances initiales

$\forall mg_{ik}$	ID_mg_{ik}, KEK_CSG_{ik}
CL_{ik}	$ID\_CL, TEK, Num\_Seq, KEK\_CSG_{ik}$

La fiabilité de la gestion des clés dans BALADE, qui constitue un véritable challenge dans un milieu hostile comme les réseaux ad hoc, est ainsi assurée à deux niveaux :

1. Quand un membre du groupe perd sa connectivité avec son cluster, à cause de son déplacement ou en raison d'un problème de ressources, il entame une procédure de ré-intégration comme cela est décrit dans la section 2.2.4, à condition qu'il détienne la clé de chiffrement de données TEK courante. Cette procédure de ré-intégration lui permettra d'obtenir la clé locale de son nouveau sous-groupe KEK_CSGik.

2. Quand un membre du groupe se rend compte qu'il ne détient pas la clé de chiffrement de données appropriée, (le numéro de séquence de la TEK ne correspond pas à celui des données chiffrées), il demande à son contrôleur local de lui re-transmettre la TEK courante, à condition qu'il détienne la clé locale du cluster auquel il appartient KEK_CSGik.

2.2.10 Clusterisation avec OMCT

OMCT [BCF05b] est un algorithme de clusterisation dynamique pour la distribution de clés de groupe multicast, dédié aux réseaux ad hoc. OMCT est exécuté par la source du groupe, afin de diviser dynamiquement le groupe multicast en des clusters fortement corrélés. Et dynamiquement, chaque contrôleur de cluster créé LC, exécutera à son tour l'algorithme OMCT. OMCT est conscient de la mobilité et de la localisation des nœuds. En effet, il utilise les informations de localisation géographique de tous les membres du groupe, dans la construction de l'arbre de distribution de clés. Nous assumons ainsi la présence d'un système GPS *Global Positioning System*, ou de tout autre support à la localisation, au sein d'un réseau ad hoc.
Nous utilisons l'algorithme OMCT de clusterisation dynamique pour diviser dynamiquement le groupe multicast en clusters, et pour élire les contrôleurs locaux de BALADE, selon leurs localisations géographiques par rapport aux autres membres du groupe.

Clusterisation à l'initialisation

À l'initialisation du groupe multicast, tous les membres sont attachés au contrôleur global CG, qui vérifie périodiquement si son groupe est fortement corrélé, et par conséquent si le processus de distribution de clés est optimal. La période de vérification de la cohésion au sein d'un cluster est notée Cohésion_Timer.

L'évaluation de la cohésion du groupe multicast est déterminée en calculant le facteur de centralisation des membres autour du nœud central CG :

$$Cohésion = \frac{membres\ dans\ la\ portee\ du\ CG}{Nombre\ de\ membres\ du\ groupe} = \frac{Voisins\ à\ un\ seul\ saut\ du\ CG}{Nombre\ de\ membres\ du\ groupe}$$

Ce paramètre mesure la proximité des membres du groupe par rapport à leur CG. En dessous du seuil minimal $Min\_Cohésion$, la cohésion est considérée perdue, et la clusterisation du groupe multicast est entamée par le CG.

Clusterisation
$\forall mg_{0k} \in LML\_CG_k$ $CG_k \Rightarrow\ mg_{0k} : \{Clusterisation\} KEK\_CSG_{0k}$ $mg_{0k} \longrightarrow CG_k : \{Id\_mg_{0k}, Coord\_mg_{0k}\} Pub\_CG_k$
Exécution de l'algorithme OMCT *Election des nouveaux controleurs locaux* CL_{ik}
$\forall i$ $CG_k \longrightarrow CL_{ik} : \{LML\_CL_{ik}, KEK\_CCL\} Pub\_CL_{ik}$ $CL_{ik} \longrightarrow mg_{ik} : \{ID_{Cluster}, KEK\_CSG_{ik}, Pub\_CL_{ik}, imprint, \{CBID - CL_{ik}\} Pri\_CL_{ik}\} Pub\_mg_{ik}$

Le CG commence par envoyer un message en multicast à tous les membres de son sous groupe, pour entamer la clusterisation selon OMCT. Ces membres lui envoient un message contenant leur identité et leurs coordonnées, crypté avec sa clé publique. À cette étape, le contrôleur global CG exécute l'algorithme OMCT qui sélectionne les nouveaux contrôleurs locaux CL_{ik}, qui doivent former le groupe des contrôleurs locaux et partager la clé de groupe KEK_CCL.

Le CG envoie à tout nouveau contrôleur local CL_{ik}, un message contenant sa liste de membre locaux LML_CL_{ik}, ainsi que la clé partagée par le groupe des contrôleurs locaux KEK_CCL. Ce message est chiffré avec les clés publiques des contrôleurs locaux.

Afin que chaque membre du groupe mg_k sache à quel cluster il appartient (et devient ainsi mg_{ik}), tout nouveau contrôleur local CL_{ik} envoie à tous les membres de sa liste LML_CL_{ik}, un message contenant l'identificateur de son cluster $ID_{Cluster}$ et la clé locale de chiffrement de clé KEK_CSG_{ik}. La clé publique Pub_CL_{ik}, l'entier imprint et le CBID du CL_{ik} permettent d'assurer son authentification. Ce message est chiffré avec la clé publique de chaque membre du cluster respectivement.

Connaissances initiales

CG_k	KEK_CSG_{0k}
$\forall mg_{ik}$	$Id\_mg_{0k}, Coord\_mg_{0k}, Pub\_CG_k$

Connaissances après l'exécution de OMCT

CG_k	$LML\_CL_{ik}, KEK\_CCL, Pub\_CL_{ik}$
$\forall CL_{ik}$	$ID_{Cluster}, KEK\_CSG_{ik}, Pub\_CL_{ik}, Pri\_CL_{ik}, imprint, \forall mg_{ik} : Pub\_mg_{ik}$

À la création des nouveaux clusters dans le groupe multicast, les nouveaux contrôleurs locaux deviennent responsables de la gestion des clés locales et de leurs distributions à leurs membres locaux respectifs. En outre, ces nouveaux CLs seront responsables d'assurer la maintenance de leurs clusters fortement corrélés. Ainsi, récursivement, et à tout instant de la vie de la session multicast, le groupe multicast reste composé de clusters fortement corrélés, assurant que leur cohésion respective est toujours supérieure au seuil minimal de cohésion $Min\ \ Cohésion$.

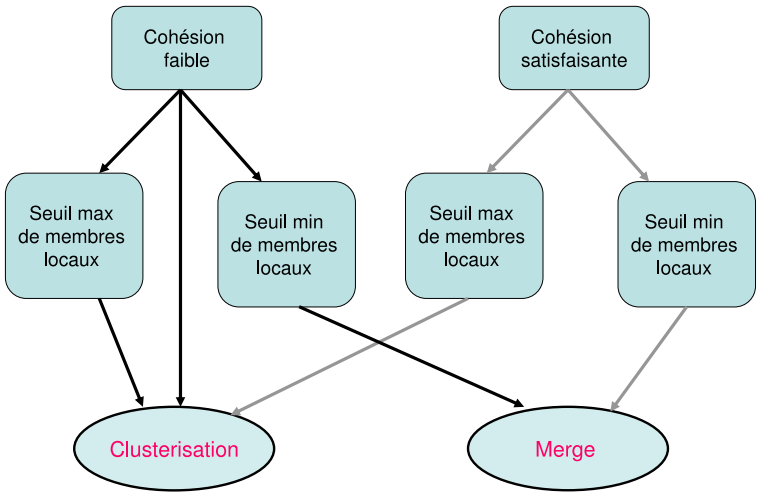

FIG. 2.2 – *Formation des clusters dans BALADE*

La maintenance de la formation de clusters fortement corrélés est assurée au cours de la session d'un groupe multicast de la façon suivante. Périodiquement, tout contrôleur local calcule le facteur de cohésion au sein de son cluster, et le nombre de ses membres locaux. Si le paramètre de cohésion atteint le seuil minimum $Min\_Cohésion$, **ou** si le nombre de membres locaux atteint le seuil maximum (MAX_Threshold), le contrôleur local entame la clusterisation de son cluster.
Et si le nombre de membres locaux atteint le seuil minimal (MIN_Threshold), le contrôleur local décide de fusionner son cluster avec d'autres clusters (cf Figure 2.2) , et ainsi de demander à ses membres locaux de joindre d'autres clusters, dans une durée maximale notée Merge_Timer, comme suit :

Merge
$\forall mg_{ik}, CL_{ik} \Rightarrow mg_{ik} : \{ID\_Cluster, LML\_CL_{ik}\} KEK\_CSG_{ik}$

Cette procédure de Merge est aussi déclenchée lorsqu'un contrôleur local veut notifier ses membres locaux de son départ du groupe multicast (cf section 2.2.5).

2.2.11 Liste des Timers et des constantes par défaut

Les valeurs des constantes et des timers dépendent des besoins fonctionnels de l'application à sécuriser par BALADE. Dans ce qui suit, nous présentons les valeurs par défaut de ces constantes et timers, qui seront validées par simulations.

1. Merge_Timer : timer permettant aux membres d'un cluster de pouvoir intégrer d'autres clusters, à cause du départ ou du Merge de leur CL. La valeur de ce timer dépend de l'application en question.
2. Join_Request_Timer : après l'expiration de ce timer, un contrôleur local considère que sa requête de demande d'adhésion d'un nœud voulant joindre le groupe a été refusée. La valeur par défaut de ce timer pourrait être fixée à 2mn.
3. Min_Cohésion : dès que la cohésion d'un cluster atteint ce seuil minimum, son CL décide d'entamer la clusterisation de son sous-groupe. La valeur par défaut de cette valeur est $\frac{1}{2}$.
4. MAX_Threshold : nombre maximum de membres d'un cluster, au delà duquel le cluster est divisé. La valeur par défaut de ce seuil est 20.
5. MIN_Threshold : nombre minimum de membres d'un cluster, au dessous duquel le cluster est fusionné avec d'autres clusters. La valeur par défaut de ce seuil est 3.
6. CL_Query_Interval : période de temps séparant deux envois successifs de message CL_Query. Cette période doit être inférieure au Merge_Timer, afin de permettre à un nœud de choisir le cluster auquel il veut adhérer, avant l'expiration d'un Merge_Timer. La valeur par défaut de cet interval est 1mn.
7. Cohésion_Timer : période de vérification de la cohésion d'un cluster. La valeur par défaut de cet interval est 4mn. Cette valeur est prouvée par le fait que la vitesse maximale des membres est définie à 3m/s (pour des raisons de stabilité [YLN03]), soit une vitesse moyenne de 1.5m/s. La portée moyenne des nœuds du réseau est de 300 à 400m. Ainsi, durant 4mn d'interval de vérification de la cohésion, un membre parcourrerait en moyenne 360m, soit une seule portée.

Chapitre 3

Spécification des sous protocoles de BALADE en HLPSL

3.1 Présentation du langage HLPSL

HLPSL (*High-Level Specification Langage*) [Tea05] est un langage de spécification, qui a les caractéristiques suivantes :

- langage modulaire, basé sur la notion de rôles (participants) et de rôles composés (sessions, instances). Un rôle simple sert à décrire les actions d'un agent lors de l'exécution d'un protocole ou d'un sous protocole. Un rôle composé instancie plusieurs rôles simples afin de modéliser l'exécution du protocole en entier. Le rôle environnement définit les agents concrets, les sessions d'exécution, ainsi que les connaissances de l'intrus et autres entités globales.
- support cryptographique varié (clés symétriques, clés publiques et privées, fonctions de hachage, ...),
- information typée (ou non), avec des types simples ou composés,
- propriétés algébriques supportées (concaténation, OU exclusif, exponentiation, ...),
- canaux pour les échanges de messages,
- flux de contrôle assurant des transitions valides,
- propriétés étudiées : confidentialité, authentification forte ou faible.

3.2 Spécification de scénarii des sous protocoles de BALADE en HLPSL

Dans cette section, nous présentons deux exemples de scénarii des sous protocoles de BALADE, initialisation du protocole et ré-intégration d'un membre du groupe. L'ensemble

des spécifications des autres sous protocoles de BALADE en HLPSL, est présenté dans l'annexe A.

3.2.1 Initialisation de BALADE

Ce sous protocole fait intervenir deux rôles, le contrôleur global CG et un membre du groupe mg. Le contrôleur global génère une nouvelle clé de chiffrement de données, et l'envoie au membre mg, chiffrée avec la clé locale. Les propriétés de sécurité que doit vérifier AVISPA sont la confidentialité des deux clés partagées par les deux entités (clé locale et clé de chiffrement de données). Deux rôles supplémentaires doivent aussi être définis: le rôle session regroupant les deux entités de la spécification (CG et mg), et le rôle environnement spécifiant les cannaux de communications et les connaissances de l'intrus.

```
%% Initialisation_TEK_Distribution

%% Premier Rôle : Le contrôleur global CG

role membre1 (CG,mg: agent,
                IDG: text,
                IDCG: text,
                Pub-CG: public_key,
                imprint: nat,
                CBID-CG: text,
                Pub-mg: public_key,
                Snd, Rcv: channel(dy))
    %% Connaissances initales du CG
     played_by CG def=

   local State: nat,
        TEK: symmetric_key,
        NumSeq: nat,
        KEK_CSG_OK: symmetric_key  %% Variables locales du CG

   const id1: protocol_id,
         id2: protocol_id  %% Constantes
   init State:=0

   transition
     step1. State=0 /\ Rcv(start)

   %% génération d'une nouvelle clé TEK
         =|> TEK' := new()
         /\ KEK_CSG_OK' := new()
```

```
         %% Incrémentation du numéro de séquence
         /\ NumSeq' := new()

         %% Envoi du message de distribution de la TEK
         /\ Snd({TEK'.NumSeq.KEK_CSG_0K'.IDG.IDCG.Pub-CG.imprint.
          {CBID-CG}_inv(Pub-CG)}_Pub-mg)
         /\ State':=1

   %% Propriétés à vérifier par l'outil AVISPA
         /\ secret(TEK',id1,{CG,mg})
         /\ secret(KEK_CSG_0K',id2,{CG,mg})
end role

%% Deuxième Rôle : un membre du groupe mg

role membre2 (CG,mg: agent,
              Snd, Rcv: channel(dy))
     played_by mg def=

   local State: nat,
       TEK: message,
       NumSeq: message,
       KEK_CSG_0K: message,
       IDG: message,
       IDCG: message,
       Pub-CG: message,
       imprint: message,
       CBID-CG: message,
       Pub-mg: public_key

    const id1: protocol_id
          id2: protocol_id
       init State:=0

   transition
     step1. State=0 /\ Rcv({TEK'.NumSeq'.KEK_CSG_0K'.IDG'.IDCG'.Pub-CG'.imprint'.
                 {CBID-CG}_inv(Pub-CG)}_Pub-mg)
       =|> State'=1
           /\ secret(TEK',id1,{CG,mg})
           /\ secret(KEK_CSG_0K',id2,{CG,mg})
```

```
%% Ce participant (membre du groupe) n'accepte les clés TEK et KEK_CSG_0K que si
   le CBID est validé authentique

end role

%% le rôle session entre les deux entités

role Initialisation_TEK_Distribution(SC, RC: channel(dy),
                CG, mg: agent,
                TEK: symmetric_key,
                NumSeq: nat,
                KEK_CSG_0K: symmetric_key,
                IDG: text,
                IDCG: text,
                Pub-CG: public_key,
                imprint: nat,
                CBID-CG: text,
                Pub-mg: public_key
          ) def=

   composition
     membre1(IDG,IDCG,Pub-CG,imprint,CBID-CG,Pub-mg,SC,RC)
     /\ membre2(CG,mg,SC,RC)

end role

%% Le rôle principal : environnement

role environment() def=
       local Snd, Rcv: channel(dy)
       const CG, mg: agent,
     IDG: text,
       IDCG: text,
       Pub-CG: public_key,
       imprint: nat,
       CBID-CG: text,
       Pub-mg: public_key

   %% Connaissances de l'intrus : identités des entités participantes
   intruder_knowledge = {CG,mg}

   composition
```

```
      Initialisation_TEK_Distribution(Snd,Rcv,CG,mg,IDG,IDCG,Pub-CG,
  imprint,CBID-CG,Pub-mg)

end role

%% L'objectif de la validation avec AVISPA est la vérification des propriétés de
%% sécurités présentées dans la section goal.

goal
secrecy_of id1
secrecy_of id2
end goal

environment()
```

3.2.2 Ré-intégration d'un membre du groupe

La spécification du scénario de ré-intégration d'un membre d'un groupe, que nous présentons dans ce qui suit, correspond à la nouvelle version du sous protocole, décrit dans la section 2.2.4.

```
%% Reintegration

%% First Role: Amgk ancien membre voulant réintégrer son groupe

role membre1 (Amgk,Mgik: agent,
              PubAmgk,PubMgik: public_key,
     CBIDAmgk,Imp: text,
     TEK: symmetric_key,
     Passwd: text,
     Snd, Rcv: channel(dy))
     played_by Amgk def=

   local State: nat,
         KEKCSGik2: message

   const id1,id2: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv(start)
```

```
       =|> Snd(PubAmgk.Imp.{CBIDAmgk}_inv(PubAmgk).{{Passwd}_TEK}_PubMgik)
           /\ secret({Passwd}_TEK,id1,{Amgk,Mgik})
           /\ State':=1
     step2. State=1 /\ Rcv({KEKCSGik2'}_PubAmgk)
       =|> State':=2
           /\ secret(KEKCSGik2,id2,{Amgk,Mgik})


end role

%% The second role: Mgik membre du groupe multicast

role membre2 (Amgk,Mgik: agent,
  PubMgik: public_key,
  KEKCSGik,TEK: symmetric_key,
              Snd, Rcv: channel(dy))
     played_by Mgik def=

   local State: nat,
         PubAmgk2: public_key,
         CBIDAMg: message,
 Passwd2: message,
 Imp2: message

    const id1,id2: protocol_id
    init State:=0

   transition
     step1. State=0 /\ Rcv(PubAmgk2'.Imp2'.{CBIDAMg'}_inv(PubAmgk2').
   {{Passwd2'}_TEK}_PubMgik)
       =|> State'=1
           /\ Snd ({KEKCSGik}_PubAmgk2')
           /\ secret({Passwd2'}_TEK,id1,{Amgk,Mgik})
   /\ secret(KEKCSGik,id2,{Amgk,Mgik})

end role

%%The role session between the two participants

role integrer(SC, RC: channel(dy),
          Amgk,Mgik: agent,
          PubAmgk,PubMgik: public_key,
```

```
  CBIDAmgk,Imp: text,
  TEK,KEKCSGik: symmetric_key,
  Passwd: text
    ) def=

   composition
     membre1(Amgk,Mgik,PubAmgk,PubMgik,CBIDAmgk,Imp,TEK,Passwd,SC,RC)
     /\ membre2(Amgk,Mgik,PubMgik,KEKCSGik,TEK,SC,RC)

end role

%%The main role

role environment() def=
   local  Snd, Rcv: channel(dy)

   const  amgk,mgik: agent,
          pubamgk,pubmgik: public_key,
  cbidamgk,imp: text,
  tek,kekcsgik: symmetric_key,
  passwd: text

   intruder_knowledge = {amgk,mgik}

   composition
      integrer(Snd,Rcv,amgk,mgik,pubamgk,pubmgik,cbidamgk,imp,tek,kekcsgik,passwd)

end role

goal
secrecy_of id1
secrecy_of id2
end goal

environment()
```

Chapitre 4

Vérification et validation du protocole BALADE à l'aide de l'outil d'analyse AVISPA

4.1 Présentation de l'outil AVISPA

AVISPA [1] [Tea05] est un analyseur de protocoles de sécurité spécifiés en HLPSL, intègrant différentes techniques d'analyse automatique de protocoles, pour un nombre fini et infini de sessions. L'interaction avec l'utilisateur est facilitée par emacs et une interface Web, illustrée dans la figure 4.1.

Sortie de AVSIPA (*Output*)

Quand l'analyse d'un protocole est réussie, en trouvant ou pas des attaques possibles, la sortie de AVISPA décrit précisément le résultat, et sous quelles conditions il a été obtenu.

- La première section *SUMMARY* indique si le protocole est sécurisé ou non, ou si l'analyse n'a pas été concluante,
- La deuxième section *DETAILS* décrit sous quelles conditions le protocole est déclaré sécurisé ou non, sous quelles conditions une attaque est trouvée, et finalement pourquoi l'analyse n'a pas été concluante,
- La section *PROTOCOL* rappelle le nom du protocole analysé,
- La section *GOAL* présente le but de l'analyse, comme par exemple la confidentialité de la clé de chiffrement des données,

1. http://www.avispa-project.org/

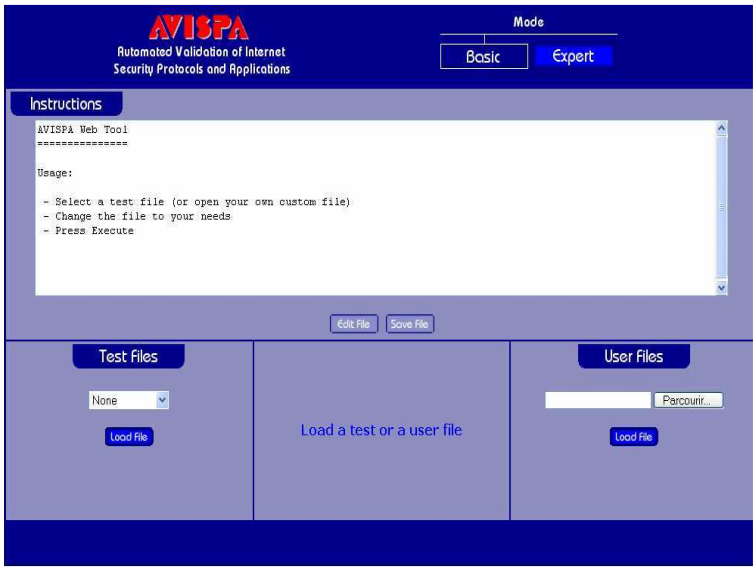

FIG. 4.1 – *Interface AVISPA*

– La section *BACKEND* désigne le traducteur des spécifications HLPSL, utilisé lors de l'analyse. Nous utilisons pour tous nos analyses le traducteur CL-ATSE (Constraint-Logic-based Attack-Searcher), comme le montre la figure 4.2.

  CL-ATSE fournit une traduction des spécifications HLPSL, en un ensemble de contraintes, qui peuvent être efficacement utilisées pour trouver des attaques malicieuses. Les opérations de traduction et de vérification sont entièrement automatiques. La validation avec CL-ATSE se fait donc de la manière suivante :

  – Chaque rôle est partiellement pré-exécuté par CL-ATSE, pour extraire la liste de contraintes minimale, le modélisant.

  – Les états et les connaissances des rôles participants sont éliminés, grâce à l'utilisation des variables globales.

  – Toute étape d'un protocole est exécutée en ajoutant des contraintes nouvelles, et en éliminant d'autres contraintes correspondantes.

  – Finalement, toute étape du système est testée pour vérifier les propriétés de sécurité fournies en objectif.

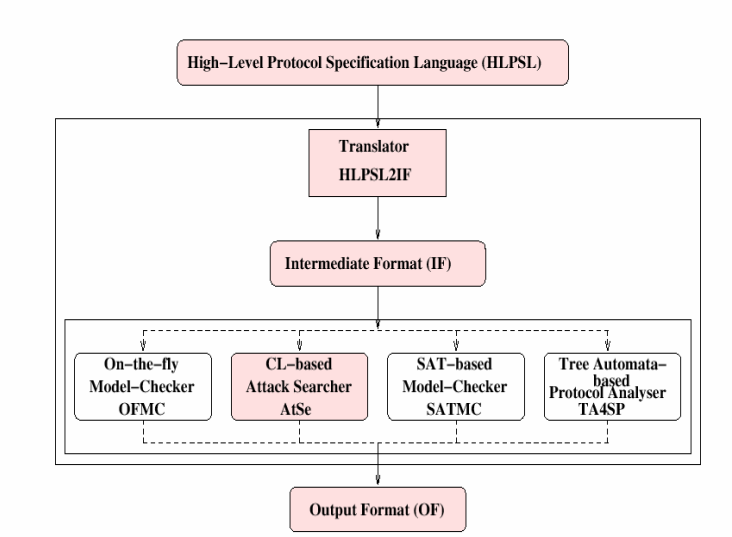

FIG. 4.2 – *AVISPA*

4.2 Validation des sous protocoles de BALADE avec AVISPA

Dans cette section, nous présentons deux exemples de validation des sous protocoles de BALADE, initialisation du protocole et ré-intégration d'un membre du groupe. L'ensemble des vérifications des autres sous protocoles de BALADE avec AVISPA, est présenté dans l'annexe B.

4.2.1 Initialisation de BALADE

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /home/avispa/web-interface-computation/./tempdir/workfileSDnRd2.if

GOAL
```

```
  As Specified

BACKEND
  CL-AtSe
```

4.2.2 Ré-intégration d'un membre du groupe

La première version de ce sous protocole présente une faille de sécurité, par ce qu'elle permet à un intrus de pouvoir intercepter $\{passwd\}\_tek$, qu'il peut ensuite utiliser pour récupérer la clé locale du cluster KEKCSGik, et récupérer la clé de chiffrement de données TEK à son prochain renouvellement. La trace de cette attaque est illustrée dans la figure 4.3.

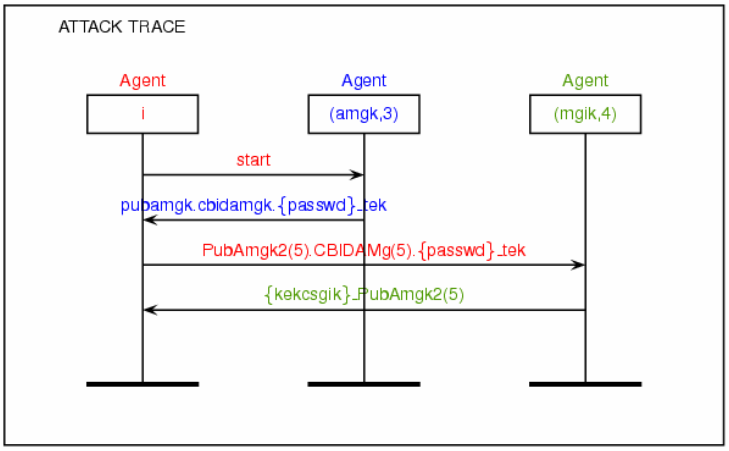

FIG. 4.3 – *Trace de l'attaque malicieuse*

La sortie AVISPA, correspondant à cette spécification éronnée de la ré intégration est présentée ci-dessous :

```
SUMMARY
  UNSAFE

DETAILS
  ATTACK_FOUND
  TYPED_MODEL
```

```
PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/Reintegration.if

GOAL
  Secrecy attack on ({passwd}_tek)

BACKEND
  CL-AtSe

ATTACK TRACE

  i -> (amgk,3):  start
  (amgk,3) -> i:  pubamgk.cbidamgk.{{passwd}_tek}_(inv(pubamgk))
                  & Secret({passwd}_tek,set_60);  Add amgk to set_60;
                  & Add mgik to set_60;
```

L'authentification et la non répudiation de l'ancien membre voulant joindre le groupe est réalisée, dans la nouvelle version de ce sous protocole, via son CBID. Le mot de passe chiffré avec TEK est maintenant chiffré avec la clé publique du membre du groupe, pour que lui seul puisse obtenir {Passwd}_TEK, et non un intrus. Le nouveau sous protocole est validé comme suit :

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/Reintegration.if

GOAL
  As Specified

BACKEND
  CL-AtSe
```

Chapitre 5

Conclusion

Dans ce rapport, nous avons validé notre protocole de gestion de clé de groupe BALADE, en utilisant l'outil AVISPA. Dans une première étape, nous avions établi la spécification informelle des différents sous protocoles de BALADE. Ensuite, nous avons spécifié des scénarii de ces sous protocoles en utilisant le langage de spécification HLPSL. Et finalement, nous avons validé ces scénarii avec AVISPA, qui procède à une recherche des attaques que peut faire un intrus, pour mettre en péril le service de sécurité que nous avons défini pour chaque sous protocole, tel que la confidentialité de la clé de chiffrement de données ou des clés de chiffrement de clés.

Notre approche de validation nous a permis de détecter une faille de sécurité au niveau du sous protocole de ré-intégration d'un ancien membre du groupe, permettant à un intrus de pouvoir intercepter et récupérer la clé de chiffrement de données. Nous avons remédié à cette faille ; la nouvelle version de ce sous protocole est désormais valide selon AVISPA.

Après avoir réaliser une validation qualitative de notre protocole avec AVISPA, nous souhaitons également mener une évaluation quantitative de celui-ci au travers de son implantation, ainsi que via des simulations dans NS2 [1].

1. http://www.isi.edu/nsnam/ns/

ANNEXES

ANNEXE A : Spécification des sous protocoles en HLPSL

A.1- Renouvellement périodique de la TEK

```
%% Renouvellement_TEK

%% Premier Rôle : Le contrôleur global CG

role membre1 (CG,mg0k,CLik: agent,
           IDCG: text,
         KEK-CSG0k : symmetric_key,
         KEK-CCL : symmetric_key,
         Snd, Rcv: channel(dy))
   played_by CG def=

   local State: nat,
          TEK: symmetric_key,
        NumSeq: nat

   const id1: protocol_id
       init State:=0

   transition
     step1. State=0 /\ Rcv(start)
       =|> TEK' := new()
           /\ Snd({IDCG.TEK'.NumSeq'}_KEK-CSG0k)
           /\ Snd({IDCG.TEK'.NumSeq'}_KEK-CCL)
           /\ State':=1
         /\ secret(TEK',id1,{CG,mg0k,CLik})
end role
```

```
%% Deuxième Rôle : mg0k appartenant à la liste des membres locaux du CG (LML_CG)

role membre2 (CG,mg0k: agent,
          KEK-CSG0k: symetric_key,
              Snd, Rcv: channel(dy))
     played_by mg0k def=

   local State: nat,
       IDCG: message,
       TEK: message,
       NumSeq: message

    const id1: protocol_id
          init State:=0

   transition
     step1. State=0 /\ Rcv({IDCG'.TEK'.NumSeq'}_KEK-CSG0k)
       =|> State'=1
           /\ secret(TEK',id1,{CG,mg0k})

end role

%% Troisième Rôle : Un contrôleur local CLik appartenant au groupe des CLs (GCL)

role membre3 (CG,CLik,mgik: agent,
           IDCL: text,
         KEK-CCL, KEK-CSGik: symetric_key,
         Snd, Rcv: channel(dy))
     played_by CLik def=

   local State: nat,
          IDCG: message,
        TEK: message,
        NumSeq: message

   const id1: protocol_id
         init State:=0

   transition
     step1. State=0 /\ Rcv({IDCG'.TEK'.NumSeq'}_KEK-CCL)
       =|> Snd({IDCL.TEK'.NumSeq'}_KEK-CSGik)
```

```
          /\ State':=1
         /\ secret(TEK',id1,{CG,CLik,mgik})
end role

%% Quatrième rôle: mgik appartenant à la liste des membres locaux du CLik (LML_CLik)

role membre4 (CLik,mgik: agent,
          kek6csgik: symmetric_key,
              Snd, Rcv: channel(dy))
     played_by mgik def=

   local State: nat,
         IDCL: message,
       TEK: message,
       NumSeq: message,

    const id1: protocol_id
         init State:=0

   transition
     step1. State=0 /\ Rcv({IDCL'.TEK'.NumSeq'}_KEK-CSGik)
       =|> State'=1
           /\ secret(TEK',id1,{CLik,mgik})

end role

%%The role session between the participants

role RenouvellementTEK(SC, RC: channel(dy),
                  CG,mg0k,CLik,mgik : agent,
                KEK_CSG_0K,KEK-CCL,KEK-CSGik: symmetric_key,
                IDCG,IDCL: text
           ) def=

   composition
     membre1(CG,mg0k,CLik,IDCG,KEK-CSG0k,KEK-CCL,SC,RC)
     /\ membre2(CG,mg0k,KEK-CSG0k,SC,RC)
     /\ membre3(CG,CLik,mgik,IDCL,KEK-CCL,KEK-CSGik,SC,RC)
     /\ membre4(CLik,mgik,KEK-CSGik,SC,RC)

end role
```

```
%%The main role

role environment() def=
   local Snd, Rcv: channel(dy)
   const CG,mg0k,CLik,mgik : agent,
       KEK_CSG_0K,KEK-CCL,KEK-CSGik: symmetric_key,
       IDCG,IDCL: text

   intruder_knowledge = {CG,mg0k,CLik,mgik}

   composition
      RenouvellementTEK(Snd,Rcv,CG,mg0k,CLik,mgik,KEK_CSG_0K,KEK-CCL,KEK-CSGik,IDCG,IDCL)

end role

goal
secrecy_of id1
end goal

environment()
```

A.2- Join d'un nouveau membre

```
%% Join

%% Premier Rôle : mgk membre voulant joindre le groupe

role membre1 (mgk,CLik: agent,
           Pub-mgk: public_key,
         imprint: text,
         CBID-mgk: text,
         Snd, Rcv: channel(dy))
   played_by mgk def=

   local State: nat,
         Puzzle-Reply: text,
       Puzzle-Request: message,
         TEK: message,
       IDCL: message,
       Passwd: message,
       KEK-CSGik: message
```

```
   const id1,id2: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv(start)
       =|> Snd(Pub-mgk)
           /\ State':=1
     step2. State=1 /\Rcv(Puzzle-Request)
     =|> Puzzle-Reply' := new()
         /\ Snd(Puzzle-Reply'.Pub-mgk.imprint.{CBID-mgk}_inv(Pub-mgk))
         /\ State'=2
     step3. State=1 /\Rcv({IDCL'.KEK-CSGik'.TEK'.Passwd'}_Pub-mgk)
     =|> State'=2
         /\ secret(TEK,id1,{CLik,mgk})
         /\ secret(KEK-CSG,id2,{CLik,mgk})

end role

%% Deuxième Rôle : CLik contrôleur local du cluster que le nouveau membre veut joindre

role membre2 (mgk,CLik,mgik: agent,
          ancienne-KEK-CSGik: symetric_key,
          IDCL: text,
          Passwd: text,
          TEK: symetric_key,
              Snd, Rcv: channel(dy))
     played_by CLik def=

   local State: nat,
       KEK-CSGik: symmetric_key,
       Pub-mgk: message,
       Puzzle-Request: text,
       Puzzle-Reply: message,
       imprint: message,
       CBID-mgk: message

   const id1,id2: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv({Pub-mgk')
```

```
       =|> Puzzle-Request' := new()
         /\Snd(Puzzle-Request)
         /\State'=1
      step2. State=1 /\ Rcv(Puzzle-Reply.Pub-mgk.imprint.{CBID-mgk}_inv(Pub-mgk))
%% Ce message n'est reçu que si le Puzzle_Reply est correct et que le CBID-mgk
est validé authentique
     =|> KEK-CSGik' := new()
         /\ Snd({IDCL.KEK-CSGik'.TEK.Passwd}_Pub-mgk)
         /\ Snd(IDCL.KEK-CSGik'}_ancienne-KEK-CSGik)
         /\ State'=2
         /\ secret(TEK,id1,{CLik,mgk})
         /\ secret(KEK-CSG,id2,{CLik,mgk,mgik})

end role

%% Troisième Rôle : mgik ancien membre appartenant au cluster géré par CLik

role membre3 (CLik,mgik: agent,
           ancienne-KEK-CSGik: symmetric_key,
         Snd, Rcv: channel(dy))
     played_by mgik def=

   local State: nat,
         IDCL: message,
       KEK-CSGik: message

   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv({IDCL'.KEK-CSGik'}_ancienne-KEK-CSGik)
       =|> State':=1
         /\ secret(KEK-CSGik',id1,{CLik,mgik})
end role

%%The role session between the participants

role Join(SC, RC: channel(dy),
                  CG,mgik,mgk : agent,
               KEK_CSG_iK,ancienne-KEK-CSGik,TEK: symmetric_key,
               IDCL,imprint,CBID-mgk,Passwd: text,
            Pub-mgk: Public_key
```

```
          ) def=

   composition
     membre1(mgk,CLik,Pub-mgk,imprint,CBID-mgk,SC,RC)
     /\ membre2(mgk,CLik,mgik,ancienne-KEK-CSGik,IDCL,Passwd,TEK,SC,RC)
     /\ membre3(CLik,mgik,ancienne-KEK-CSGik,SC,RC)

end role

%%The main role

role environment() def=
   local Snd, Rcv: channel(dy)
   const CG,mgik,mgk : agent,
       KEK_CSG_iK,ancienne-KEK-CSGik,TEK: symmetric_key,
       IDCL,imprint,CBID-mgk,Passwd: text,
       Pub-mgk: Public_key

   intruder_knowledge = {mgk,CLik,mgik}

   composition
      Join(Snd,Rcv,CG,CG,mgik,mgk,KEK_CSG_iK,ancienne-KEK-CSGik,TEK,IDCL,imprint,
      CBID-mgk,Passwd,Pub-mgk)

end role

goal
secrecy_of id1
secrecy_of id2
end goal

environment()
```

A.3- Leave d'un membre du groupe

```
%% Leave_membre_du_groupe

%% Premier Rôle : Mgk membre voulant quitter le groupe

role membre1 (Mgk,CLik: agent,
              PubMgk: public_key,
```

```
      Imp: text,
      CBIDMgk: text,
      Snd, Rcv: channel(dy))
   played_by Mgk def=

   local State: nat

   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv(start)
       =|> Snd(PubMgk.Imp.{CBIDMgk}_inv(PubMgk))
   /\ State'=1

end role

%% Deuxième Rôle : CLik contrôleur local du cluster que le nouveau membre veut quitter

role membre2 (Mgk,CLik,Mgik: agent,
  IDCL: text,
  PubMgik: public_key,
             Snd, Rcv: channel(dy))
     played_by CLik def=

   local State: nat,
   KEKCSGik: symmetric_key,
   Imp2: message,
   CBIDMgk2: message,
   PubMgk2: message
   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv(PubMgk2'.Imp2'.{CBIDMgk2'}_inv(PubMgk2'))
     %% Ce message n'est reçu que si le Puzzle_Reply est correct et que le CBIDMgk est
     validé authentique
     =|>  Snd({IDCL.KEKCSGik'}_PubMgik)
        /\ State':=1
  /\ secret(KEKCSGik',id1,{CLik,Mgik})

end role
```

```
%% Troisième Rôle : Mgik ancien membre appartenant au cluster géré par CLik

role membre3 (CLik,Mgik: agent,
      PubMgik : public_key,
              Snd, Rcv: channel(dy))
     played_by Mgik def=

   local State: nat,
         IDCL3: message,
 KEKCSGik: message

   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv({IDCL3'.KEKCSGik'}_PubMgik)
       =|> State':=1
     /\ secret(KEKCSGik',id1,{CLik,Mgik})

end role

%%The role session between the participants

role leave(SC, RC: channel(dy),
           CLik,Mgik,Mgk : agent,
   IDCL,Imp,CBIDMgk: text,
   PubMgk,PubMgik: public_key
           ) def=

   composition
     membre1(Mgk,CLik,PubMgk,Imp,CBIDMgk,SC,RC)
     /\ membre2(Mgk,CLik,Mgik,IDCL,PubMgik,SC,RC)
     /\ membre3(CLik,Mgik,PubMgik,SC,RC)

end role

%%The main role

role environment() def=
   local Snd, Rcv: channel(dy)
   const clik,mgik,mgk : agent,
```

```
   idcl,imp,cbidmgk: text,
   pubmgk,pubmgik: public_key

   intruder_knowledge = {mgk,clik,mgik}

   composition
      leave(Snd,Rcv,clik,mgik,mgk,idcl,imp,cbidmgk,pubmgk,pubmgik)

end role

goal
secrecy_of id1
end goal

environment()
```

A.4- Leave d'un contrôleur local du groupe

```
%% Leave_contrôleur_local_du_groupe

%% Premier Rôle : CLik membre voulant quitter le groupe

role membre1 (CLik,MGik,CG: agent,
              KEKCSGik,KEKCCL: symmetric_key,
      IDCLik: text,
      Snd, Rcv: channel(dy))
   played_by CLik def=

   local State: nat

   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv(start)
       =|> Snd({IDCLik}_KEKCSGik)
           /\ Snd(IDCLik}_KEKCCL)
   /\ State'=1

end role
```

```
%% Deuxième Rôle : CG contrôleur global du groupe

role membre2 (CLik,CG,CLjk: agent,
  IDCG: text,
  KEKCCL: symmetric_key,
  PubCLjk: public_key,
              Snd, Rcv: channel(dy))
     played_by CG def=

   local State: nat,
   NEWKEKCCL: symmetric_key,
   IDCLik2: message
   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv((IDCLik2}_KEKCCL))
     =|>  NEWKEKCCL' = new()
        /\Snd({IDCG.NEWKEKCCL'}_PubCLjk)
        /\ State':=1
  /\ secret(NEWKEKCCL',id1,{CG,CLjk})

end role

%% Troisième Rôle : Mgik membre appartenant au cluster géré par CLik

role membre3 (CLik,Mgik: agent,
      KEKCSGik : public_key,
               Snd, Rcv: channel(dy))
     played_by Mgik def=

   local State: nat,
         IDCLik2: message,

   init State:=0

   transition
     step1. State=0 /\ Rcv({IDCLik2}_KEKCSGik)
       =|> State':=1

end role
```

```
%% Quatrième Rôle : CLjk contrôleur local appartenant au groupe GCL, j<>i

role membre4 (CLjk,CG: agent,
      PubCLjk: public_key,
               Snd, Rcv: channel(dy))
     played_by CLjk def=

   local State: nat,
         IDCG2: message,
 NEWKEKCCL2: message

   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv({IDCG2',NEWKEKCCL2'}_PubCLjk)
       =|> State':=1
           /\ secret(NEWKEKCCL2,id1,{CG,CLjk})

end role


%%The role session between the participants

role leaveCL(SC, RC: channel(dy),
           CLik,CLjk,Mgik,CG : agent,
   IDCLik,IDCG: text,
   PubCLjk: public_key,
   KEKCCL,KEKCSGik: symmetric_key
           ) def=

   composition
     membre1(CLik,MGik,CG,KEKCSGik,KEKCCL,IDCLik,SC,RC)
     /\ membre2(CLik,CG,CLjk,IDCG,KEKCCL,PubCLjk,SC,RC)
     /\ membre3(CLik,Mgik,KEKCSGik,SC,RC)
     /\ membre3(CLjk,CG,PubCLjk,SC,RC)

end role

%%The main role

role environment() def=
```

```
   local Snd, Rcv: channel(dy)
   const clik,cljk,mgik,cg : agent,
 idclik,idcg: text,
 pubcljk: public_key,
 kekccl,kekcsgik: symmetric_key

   intruder_knowledge = {mgik,clik,cljk,cg}

   composition
      leaveCL(Snd,Rcv,clik,cljk,mgik,cg,idclik,idcg,pubcljk,kekccl,kekcsgik)

end role

goal
secrecy_of id1
end goal

environment()
```

A.5- Exclusion d'un membre du groupe

```
%% Exclusion
%% La partie Renouvellement de la TEK ne sera pas vérifiée ici puiqu'elle est similaire
au sous protocole de renouvellement périodique de la TEK, validé précedemment

%% Premier Rôle : CLik contrôleur local du membre exclu

role membre1 (CLik,CG,Mgik: agent,
           IDCLik: text,
   PubCLik,PubMgik: public_key,
   Imp:text,
   CBIDCLik: text,
   Snd, Rcv: channel(dy))
   played_by CLik def=

   local State: nat,
          KEKCSGik: symmetric_key

   const id1: protocol_id
   init State:=0
```

```
   transition
     step1. State=0 /\ Rcv(start)
       =|> KEKCSGik' := new()
           /\ Snd({IDCLik.KEKCSGik}_PubMgik)
           /\ Snd(IDCLik.PubCLik.Imp.{CBIDCLik}_inv(PubCLik))
           /\ State':=1
   /\ secret(KEKCSGik,id1,{CLik,Mgik})
end role

%% Deuxième Rôle : Mgik appartenant à la liste des membres locaux du CLik

role membre2 (CLik,Mgik: agent,
      PubMgik: public_key,
      Snd, Rcv: channel(dy))
     played_by Mgik def=

   local State: nat,
   IDCLik2: message,
   KEKCSGik2: message

    const id1: protocol_id
          init State:=0

   transition
     step1. State=0 /\ Rcv({IDCLik2'.KEKCSGik2'}_PubMgik)
       =|> State'=1
           /\ secret(KEKCSGik2,id1,{CLik,Mgik})

end role

%% Troisième Rôle : CG contrôleur global du groupe

role membre3 (CG,CLik: agent,
            Snd, Rcv: channel(dy))
     played_by CG def=

   local State: nat,
         IDCLik3: message,
 PubCLik3: public_key,
 Imp3: message,
 CBIDCLik2: message
```

```
   const id1: protocol_id
         init State:=0

   transition
     step1. State=0 /\ Rcv(IDCLik3'.PubCLik3'.Imp3'.
{CBIDCLik2'}_inv(PubCLik3'))
       =|> State':=1
%% Renouvellement de la TEK par le CG

end role

%%The role session between the participants

role exclu(SC, RC: channel(dy),
                  CG,CLik,Mgik : agent,
        IDCLik,Imp,CBIDCLik: text,
  PubMgik,PubCLik: public_key
          ) def=

   composition
     membre1(CLik,CG,Mgik,IDCLik,PubCLik,PubMgik,Imp,CBIDCLik,SC,RC)
     /\ membre2(CLik,Mgik,PubMgik,SC,RC)
     /\ membre3(CG,CLik,SC,RC)

end role

%%The main role

role environment() def=
   local Snd, Rcv: channel(dy)
   const cg,clik,mgik : agent,
 idclik,imp,cbidclik: text,
 pubmgik,pubclik: public_key


   intruder_knowledge = {cg,clik,mgik}

   composition
      exclu(Snd,Rcv,cg,clik,mgik,idclik,imp,cbidclik,pubmgik,pubclik)

end role
```

```
goal
secrecy_of id1
end goal

environment()
```

A.6- Envoi périodique des CL_Queries

```
%% CLQUERY

%% Premier Rôle : CLik contrôleur local du groupe

role membre1 (CLik,MGk: agent,
              IDCLik: text,
     PubCLik: public_key,
     Imp: text,
     CBIDCLik: text,
     COORDCLik: text,
     Snd, Rcv: channel(dy))
   played_by CLik def=

   local State: nat

   init State:=0

   transition
     step1. State=0 /\ Rcv(start)
       =|> Snd(IDCLik.PubCLik.Imp.{CBIDCLik}_inv(PubCLik).COORDCLik)
           /\ State':=1

end role

%% Deuxième Rôle : MGk membre du groupe à deux sauts du CLik

role membre2 (CLik,MGk: agent,
      Snd, Rcv: channel(dy))
     played_by MGk def=

   local State: nat,
   IDCLik2: message,
   PubCLik2: public_key,
```

```
   Imp2: message,
   CBIDCLik2: message,
   COORDCLik2: message

   init State:=0

   transition
     step1. State=0 /\ Rcv(IDCLik2'.PubCLik2'.Imp2'.
{CBIDCLik2'}_inv(PubCLik2').COORDCLik2')
     %%Ce message n'est reçu que si le CBID du Clik est validé authentique
      =|> State':=1

end role

%%The role session between the participants

role query(SC, RC: channel(dy),
                  CLik,MGk: agent,
        IDCLik: text,
        PubCLik: public_key,
          Imp: text,
          CBIDCLik: text,
          COORDCLik: text
           ) def=

   composition
     membre1(MGk,CLik,IDCLik,PubCLik,Imp,CBIDCLik,COORDCLik,SC,RC)
     /\ membre2(MGk,CLik,SC,RC)


end role

%%The main role

role environment() def=
   local Snd, Rcv: channel(dy)
   const   clik,mgk: agent,
   idclik: text,
   pubclik: public_key,
   imp: text,
   cbidclik: text,
   coordclik: text
```

```
   intruder_knowledge = {clik,mgk}

   composition
      query(Snd,Rcv,clik,mgk,idclik,pubclik,imp,cbidclik,coordclik)

end role

environment()
```

A.7- Gestion des listes ACL et RL

```
%% Demande d'autorisation d'accès

%% First Role: CLik contrôleur local qui demande l'autorisation d'accès d'un membre

role membre1 (CLik,CLrk: agent,
              KEKCCL: symmetric_key,
     IDCLik: text,
     MGnew: text,
     Snd, Rcv: channel(dy))
     played_by CLik def=

   local State: nat,
         IDCLrk2: message,
 Result2: message

   init State:=0

   transition
     step1. State=0 /\ Rcv(start)
       =|> Snd({IDCLik.MGnew}_KEKCCL)
           /\ State':=1
     step2. State=1 /\ Rcv({IDCLrk2'.MGnew.Result2'}_KEKCCL)
       =|> State':=2

end role

%% The second role: CLrk contrôleur local qui détient l'information ACL

role membre2 (CLik,CLrk: agent,
```

```
KEKCCL: symmetric_key,
IDCLrk:text,
Snd, Rcv: channel(dy))
     played_by CLrk def=

   local State: nat,
         Result: text,
         IDCLk2: message,
 MGnew2: message

    const id1: protocol_id
    init State:=0

    transition
     step1. State=0 /\ Rcv({IDCLk2'.MGnew2'}_KEKCCL)
       =|> State'=1
           /\ Result'= new()
           /\ Snd ({IDCLrk.IDCLk2'.MGnew2'.Result'}_KEKCCL)

end role

%%The role session between the two participants

role acl(SC, RC: channel(dy),
           CLik,CLrk: agent,
              KEKCCL: symmetric_key,
      IDCLik,IDCLrk: text,
      MGnew: text
    ) def=

   composition
     membre1(CLik,CLrk,KEKCCL,IDCLik,MGnew,SC,RC)
     /\ membre2(CLik,CLrk,KEKCCL,IDCLrk,SC,RC)

end role

%%The main role

role environment() def=
   local  Snd, Rcv: channel(dy)
   const  clik,clrk: agent,
          kekccl: symmetric_key,
```

```
  idclik,idclrk: text,
  mgnew: text

   intruder_knowledge = {clik,clrk}

   composition
      acl(Snd,Rcv,clik,clrk,kekccl,idclik,idclrk,mgnew)

end role

environment()
```

A.8- Retransmission de la TEK par les contrôleurs de BALADE

```
%% TEK retransmission

%% First Role: Mgik membre du groupe qui demande à son CLik de lui transmettre la TEK

role membre1 (MGik,CLik: agent,
              IDMGik: text,
     KEKCSGik: symmetric_key,
     Snd, Rcv: channel(dy))
     played_by MGik def=

   local State: nat,
     IDCL2: message,
 TEK2: message,
 NUMseq2: message

   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv(start)
       =|> Snd({IDMGik}_KEKCSGik)
           /\ State':=1
     step2. State=1 /\ Rcv({IDCL2'.TEK2'.NUMseq2'}_KEKCSGik)
       =|> State':=2
       /\secret(TEK2,id1,{MGik,CLik})

end role
```

```
%% The second role: CLik contrôleur local de MGik

role membre2 (MGik,CLik: agent,
      IDCL,NUMseq: text,
      TEK,KEKCSGik: symmetric_key,
      Snd, Rcv: channel(dy))
     played_by CLik def=

    local State: nat,
          IDMGik2: message

    const id1: protocol_id
    init State:=0

    transition
     step1. State=0 /\ Rcv({IDMGik2'}_KEKCSGik)
       =|> State'=1
           /\ Snd ({IDCL.TEK.NUMseq}_KEKCSGik)
   /\secret(TEK,id1,{MGik,CLik})

end role

%%The role session between the two participants

role tek(SC, RC: channel(dy),
        MGik,CLik: agent,
        IDMGik,IDCL,NUMseq: text,
KEKCSGik,TEK: symmetric_key
    ) def=

   composition
     membre1(MGik,CLik,IDMGik,KEKCSGik,SC,RC)
     /\ membre2(MGik,CLik,IDCL,NUMseq,TEK,KEKCSGik,SC,RC)

end role

%%The main role

role environment() def=
   local Snd, Rcv: channel(dy)
   const mgik,clik: agent,
```

```
         idmgik,idcl,numseq: text,
 kekcsgik,tek: symmetric_key

   intruder_knowledge = {mgik,clik}

   composition
      tek(Snd,Rcv,mgik,clik,idmgik,idcl,numseq,kekcsgik,tek)

end role

goal
secrecy_of id1
end goal

environment()
```

A.9- Clusterisation avec OMCT

```
%% Renouvellement_TEK

%% Premier Rôle : Le contrôleur global CG

role membre1 (CG,Mgok,CLik: agent,
  Clusterisation: text,
        KEKCSG0k: symmetric_key,
    PubCGk,PubCLik: public_key,
  Snd, Rcv: channel(dy))
   played_by CG def=

   local State: nat,
     KEKCCL: symmetric_key,
     IDmg0k2: message,
     Coordmg0k2: message,
     LMLCLik: text

   const id1: protocol_id
   init State:=0

   transition
     step1. State=0 /\ Rcv(start)
```

```
       =|> Snd({Clusterisation}_KEKCSG0k)
           /\ State':=1
     step2. State=1 /\ Rcv({IDmg0k2'.Coordmg0k2'}_PubCGk)
 =|> State':=2
     /\ KEKCCL' := new()
     /\ LMLCLik' := new()
     /\ Snd({LMLCLik'.KEKCCL'}_PubCLik)
     /\ secret(KEKCCL',id1,{CG,CLik})

end role

%% Deuxième Rôle : Mgok appartenant à la liste des membres locaux du CG (LML_CG)

role membre2 (CG,Mgok: agent,
  KEKCSG0k: symmetric_key,
  PubCGk: public_key,
  IDmg0k: text,
              Snd, Rcv: channel(dy))
     played_by Mgok def=

   local State: nat,
   Clusterisation2: message,
   Coordmg0k: text

    const id1: protocol_id
    init State:=0

   transition
     step1. State=0 /\ Rcv({Clusterisation2'}_KEKCSG0k)
       =|> State'=1
     /\ Coordmg0k' := new()
     /\ Snd({IDmg0k.Coordmg0k'}_PubCGk)

end role

%% Troisième Rôle : Un contrôleur local CLik élu par le CG

role membre3 (CG,CLik,Mgik: agent,
              PubCLik: public_key,
     Imp,CBIDCLik: text,
     Snd, Rcv: channel(dy))
     played_by CLik def=
```

```
   local State: nat,
    LMLCLik2: message,
    KEKCCL2: message,
            IDcluster: text,
    KEKCSGik: symmetric_key,
    Pubmgik: public_key

   const id1,id2: protocol_id
         init State:=0

   transition
     step1. State=0 /\ Rcv({LMLCLik2'.KEKCCL2'}_PubCLik)
       =|>   State':=1
     /\ secret(KEKCCL2,id1,{CG,CLik})
     /\ IDcluster' := new()
     /\ KEKCSGik' := new()
     /\ Pubmgik' := new()
     /\ Snd({IDcluster'.KEKCSGik'.PubCLik.Imp.{CBIDCLik}_inv(PubCLik)}_Pubmgik')
     /\ secret(KEKCSGik,id2,{CLik,Mgik})

end role

%% Quatrième rôle: Mgik appartenant à la liste des membres locaux du CLik (LML_CLik)

role membre4 (CLik,Mgik: agent,
  PubMgik: public_key,
              Snd, Rcv: channel(dy))
     played_by Mgik def=

   local State: nat,
   IDcluster2: message,
   KEKCSGik2: message,
   PubCLik2: public_key,
   Imp2,CBIDCLik2: text

    const id2: protocol_id
          init State:=0

   transition
     step1. State=0 /\ Rcv({IDcluster2'.KEKCSGik2'.PubCLik2'.Imp2'.
{CBIDCLik2'}_inv(PubCLik2')}_PubMgik)
```

```
       =|> State'=1
           /\ secret(KEKCSGik2,id2,{CLik,Mgik})

end role

%%The role session between the participants

role omct(SC, RC: channel(dy),
                  CG,Mg0k,CLik,Mgik: agent,
  Clusterisation: text,
          KEKCSG0k: symmetric_key,
    PubCGk,PubCLik,PubMgik: public_key,
         IDmg0k: text,
          Imp,CBIDCLik: text
           ) def=

   composition
     membre1(CG,Mg0k,CLik,Clusterisation,KEKCSG0k,PubCGk,PubCLik,SC,RC)
     /\ membre2(CG,Mg0k,KEKCSG0k,PubCGk,IDmg0k,SC,RC)
     /\ membre3(CG,CLik,Mgik,PubCLik,imp,CBIDCLik,SC,RC)
     /\ membre4(CLik,Mgik,PubMgik,SC,RC)

end role

%%The main role

role environment() def=
   local Snd, Rcv: channel(dy)
   const cg,mg0k,clik,mgik : agent,
         clusterisation: text,
   kekcsg0k: symmetric_key,
     pubcgk,pubclik,pubmgik: public_key,
         idmg0k: text,
   imp,cbidclik: text

   intruder_knowledge = {cg,mg0k,clik,mgik}

   composition
      omct(Snd,Rcv,cg,mg0k,clik,mgik,clusterisation,kekcsg0k,pubcgk,
           pubclik,pubmgik,idmg0k,imp,cbidclik)

end role
```

```
goal
secrecy_of id1
secrecy_of id2
end goal

environment()
```

ANNEXE B : Vérification des sous protocoles avec AVISPA

B.1- Renouvellement périodique de la TEK

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/RenouvellementTEK.if

GOAL
  As Specified

BACKEND
  CL-AtSe
```

B.2- Join d'un nouveau membre

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
```

```
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/Join.if

GOAL
  As Specified

BACKEND
  CL-AtSe
```

B.3- Leave d'un membre du groupe

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/Leave.if

GOAL
  As Specified

BACKEND
  CL-AtSe
```

B.4- Leave d'un contrôleur local du groupe

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/Leave_CL.if

GOAL
```

```
  As Specified

BACKEND
  CL-AtSe
```

B.5- Exclusion d'un membre du groupe

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/Exclusion.if

GOAL
  As Specified

BACKEND
  CL-AtSe
```

B.6- Envoi périodique des CL_Queries

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/CLQUERY.if

GOAL
  As Specified

BACKEND
```

```
  CL-AtSe
```

B.7- Gestion des listes ACL et RL

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/ACL.if

GOAL
  As Specified

BACKEND
  CL-AtSe
```

B.8- Retransmission de la TEK par les contrôleurs de BALADE

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/RETEK.if

GOAL
  As Specified

BACKEND
  CL-AtSe
```

B.9- Clusterisation avec OMCT

```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  /users/madyne/bouassid/AVISPA/avispa-1.0//testsuite/results/OMCT.if

GOAL
  As Specified

BACKEND
  CL-AtSe
```

Bibliographie

[ANL01] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant authentication with client puzzles. *Lecture Notes in Computer Science*, 2133:170+, 2001.

[BCF05a] M.S. Bouassida, I. Chrisment, and O. Festor. BALADE : Diffusion multicast sécurisée d'un flux multimédia multi-sources séquentielles dans un environnement ad hoc. In *CFIP 2005*, BORDEAUX, FRANCE, March 2005.

[BCF05b] M.S. Bouassida, I. Chrisment, and O. Festor. Efficient Clustering for Multicast Key Distribution in MANETs. In *Networking 2005, International IFIP TC6 Networking Conference*, Waterloo, CANADA, May 2005.

[CCC+04] Y. Chevalier, L. Compagna, J. Cuellar, P-H. Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. A high level protocol specification language for industrial security-sensitive protocols. In *Workshop on Specification and Automated Processing of Security Requirements (SAPS 2004)*. 2004.

[MC02] G. Montenegro and C. Castelluccia. Statistically Unique and Cryptographically Verifiable Identifiers and Addresses. In *ISOC Network and Distributed System Security Symposium (NDSS)*, February 2002.

[Tea05] AVISPA Team. Avispa v1.0 user manual, automated validation of internet security protocols and applications, June 2005.

[YLN03] J. Yoon, M. Liu, and B. Noble. Random Waypoint considered harmful. In *IEEE INFOCOM - San Fransisco, California, USA*, March 2003.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399