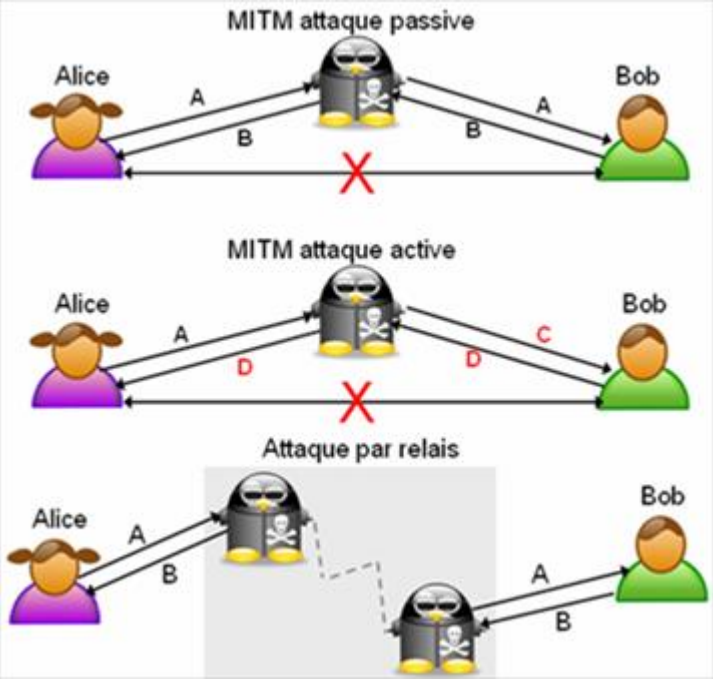
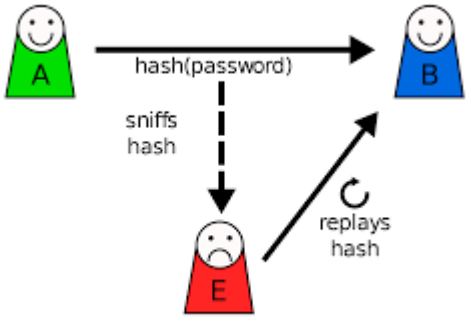
Vérification de protocoles cryptographiques

Comment s'y prendre ?

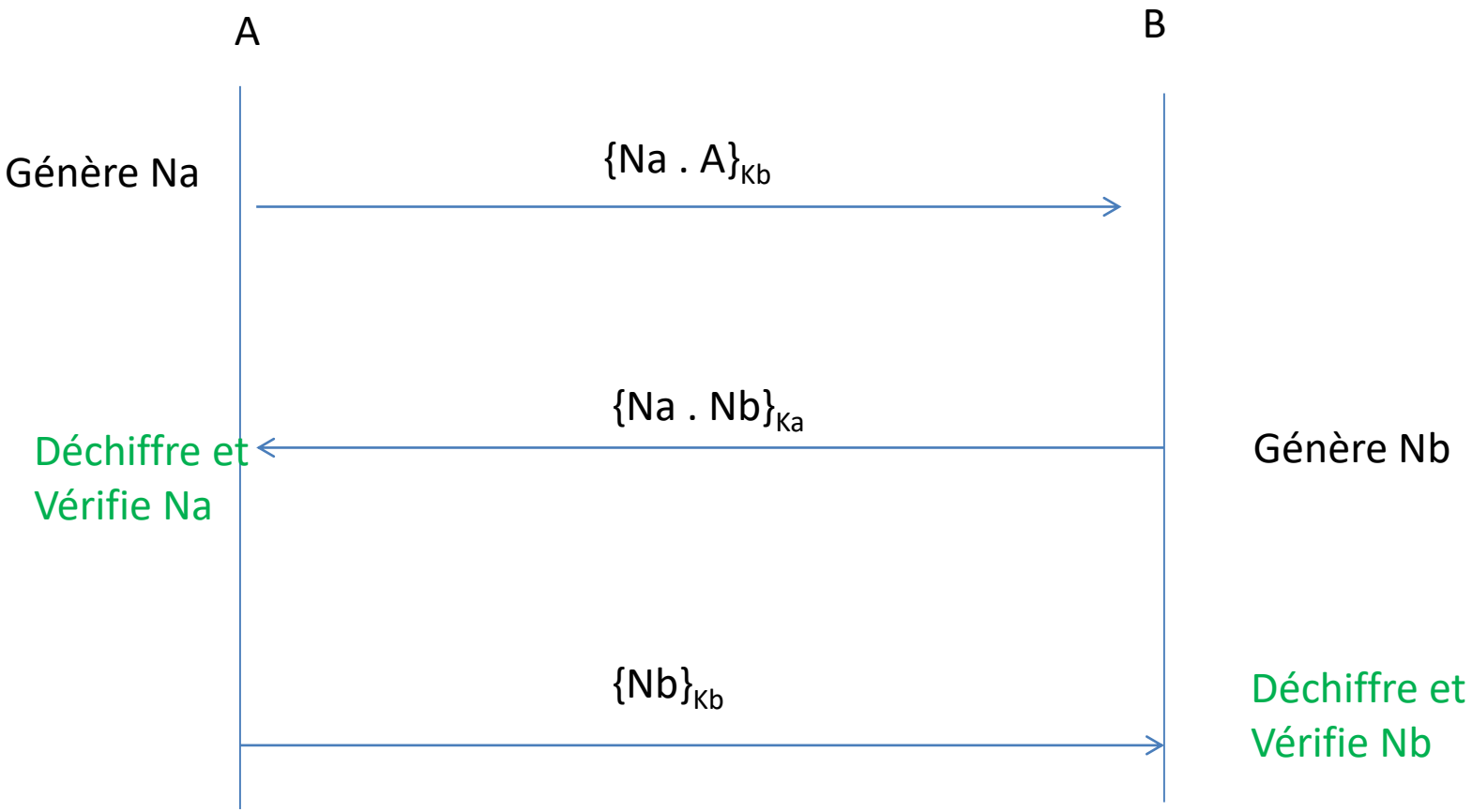




Needham-Schroeder

- 1. A → B : $\{Na . A\}_{Kb}$
- 2. B → A : $\{Na . Nb\}_{Ka}$
- 3. A → B : $\{Nb\}_{Kb}$

- Propriétés garanties :
  - secret : Na et Nb connus que de A et B
  - authentification : authentification de B auprès de A au pas 2 (par Na) et authentification de A auprès de B au pas 3 (par Nb)


A
B
Génère Na
{Na . A}Kb
{Na . Nb}Ka
Déchiffre et
Vérifie Na
Génère Nb
{Nb}Kb
Déchiffre et
Vérifie Nb

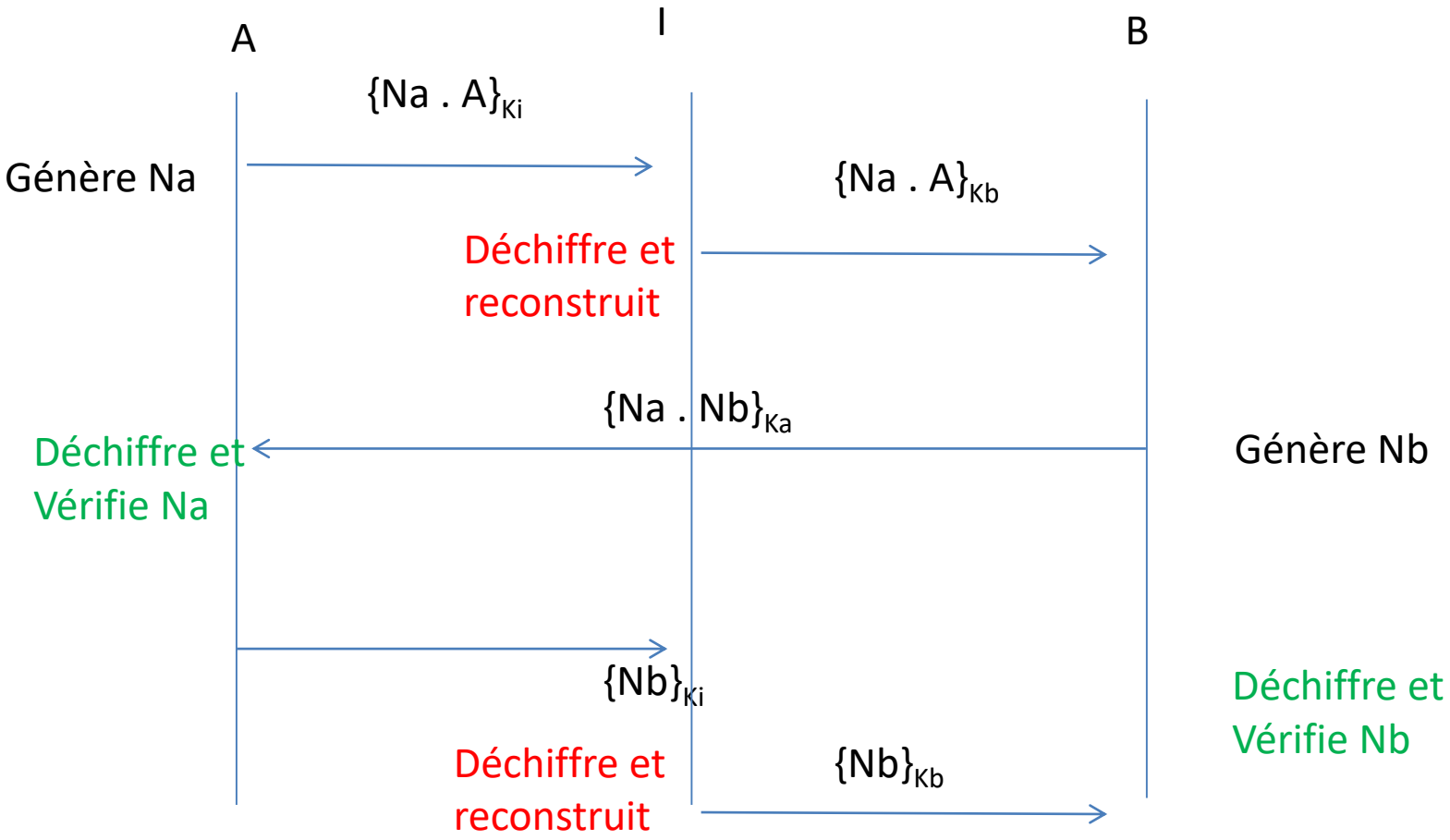
Attaque

Une session entre A et I et une session entre I est B : I malhonnête (se fait passer pour A)

S1.1	A → I :	$\{Na . A\}_{Ki}$
S2.1	I(A) → B :	$\{Na . A\}_{Kb}$
S2.2	B → I(A) :	$\{Na . Nb\}_{Ka}$
S1.2	I → A :	$\{Na . Nb\}_{Ka}$
S1.3	A → I :	$\{Nb\}_{Ki}$
S2.3	I(A) → B :	$\{Nb\}_{Kb}$

⇒ Violation du secret : I connait Nb que B pense partager avec A

⇒ Violation de l'authentification : B pense jouer le protocole avec A (et non I)



I se fait pas pour A auprès de B mais personne ne s'aperçoit de rien !

Correction

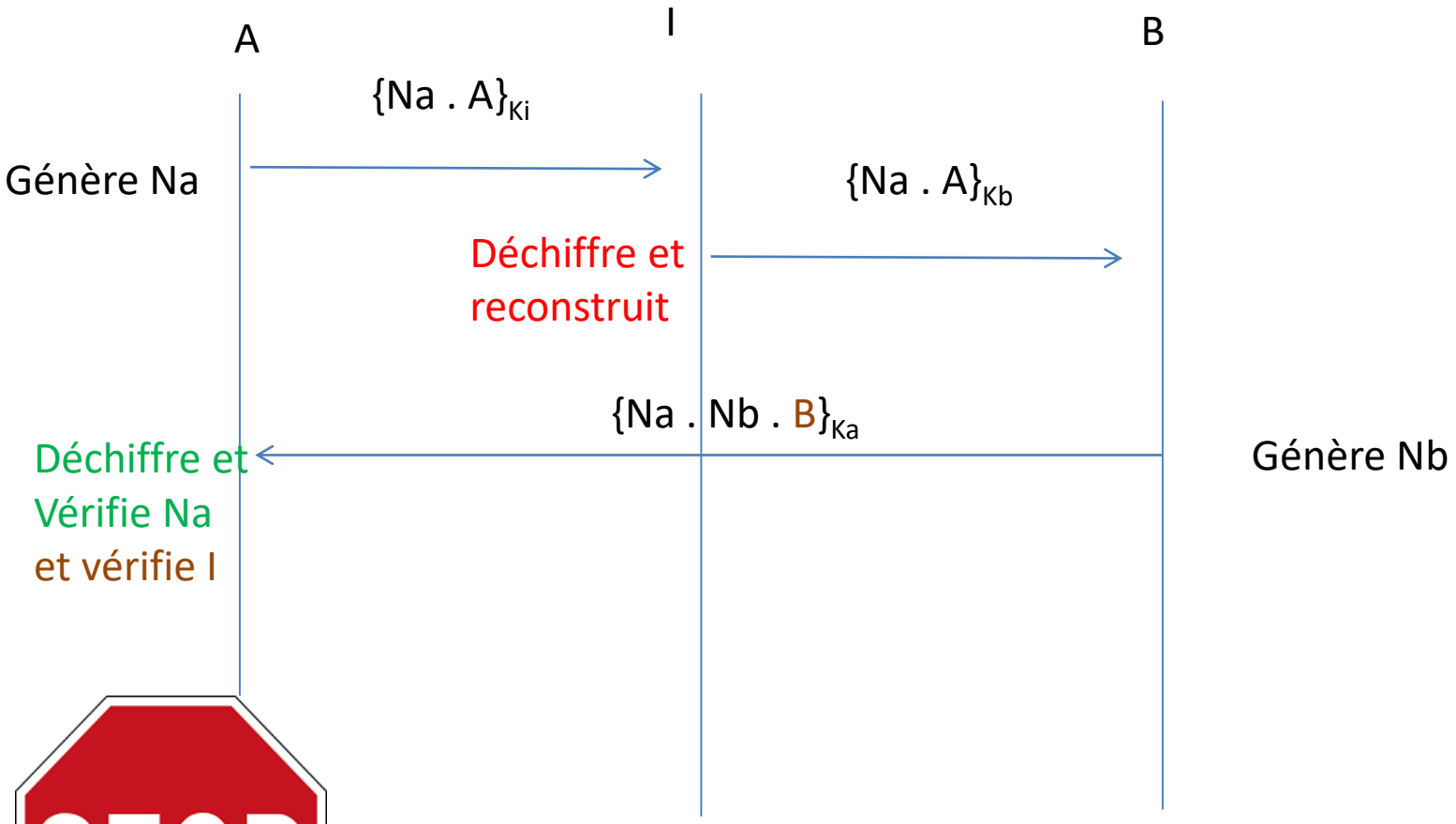
- Needham-Schroeder-Lowe :

A → B : {Na . A}Kb

B → A : {Na. Nb. B}Ka

A → B : {Nb}Kb

L'attaque précédente ne peut pas avoir lieu.



Mais est-il sûr ? Qu'est ce que ca veut dire ?

Vérifier des protocoles cryptographiques

- Modéliser les protocoles
- Modéliser l'attaquant : ce qu'il peut faire, ce qu'il peut calculer
- Enoncer des propriétés de sécurité et les vérifier

- Un outil de vérification :
  - Calcul de tous les comportements possibles
  - Répondre SAFE, attaque ou timeout

Preuve de propriétés

1. Modéliser les processus et vérifications possibles par chaque agent
2. Modéliser l'intrus
3. Combiner agents, intrus et sessions
4. Enoncer des propriétés et les vérifier

1. Modélisation processus

Session entre A (initiateur) et B (répondeur)

Processus A :

1. Envoie $\{Na.\ A\}_{Kb}$
2. Reçoit $\{Na.\ z\}_{Ka}$ -- vérifie Na
3. Envoie $\{z\}_{Kb}$

Processus B :

1. Reçoit : $\{x\ .\ A\}_{Kb}$
2. Envoie : $\{x\ .\ Nb\}_{Ka}$
3. Reçoit : $\{Nb\}_{Kb}$ -- vérifie Nb

z et x variables

Un langage pour décrire les protocoles sous forme de processus (hlpsl)

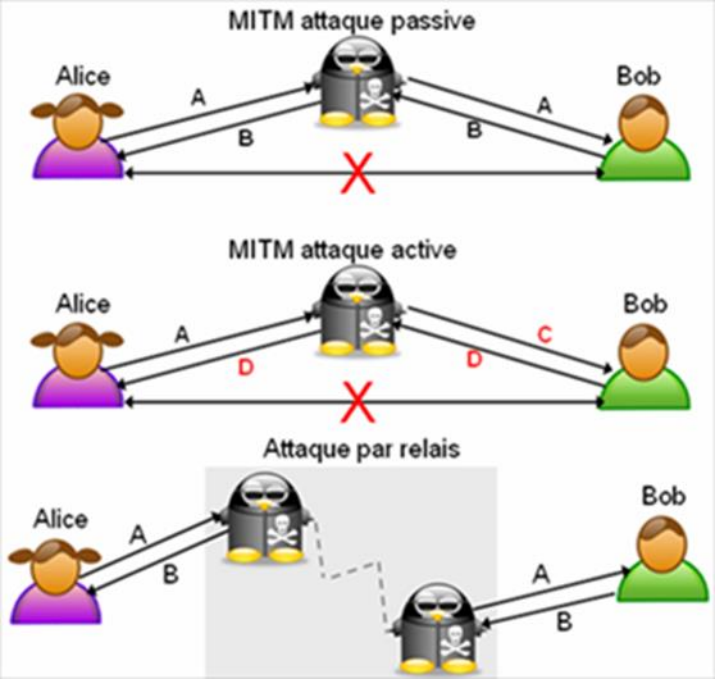
```
role alice (A, B: agent,
            Ka, Kb: public_key,
            SND, RCV: channel (dy))
  played_by A def=
    local State : nat,          % les états de l'automate alice seront numérotés 0, 2, 4…
          Na, Nb: text          %  text désigne le type des nonces
    init State := 0
    transition                                                  % Alice part de l'état 0
     0.  State  = 0 /\ RCV(start) =|>                           % puis passe à l'état 2
         State':= 2 /\ Na' := new() /\ SND({Na'.A}_Kb)          % en calculant un nonce
                                                                % et en l'envoyant chiffré avec Kb
     2.  State  = 2 /\ RCV({Na.Nb'}_Ka) =|>                     % quand elle reçoit, elle déchiffre Nb
         State':= 4 /\ SND({Nb'}_Kb)                            %  et le renvoie chiffré avec Kb
  end role
```

- Na : valeur connue, Na’ : valeur inconnue (variable)
- RCV : reçoit, SND : envoie

Preuve de propriétés

1. Modéliser les processus et vérifications possibles par chaque agent
2. Modéliser l'intrus
3. Combiner agents, intrus et sessions
4. Enoncer des propriétés et les vérifier

Intrus



- Ce qu'il peut faire
- Ce qu'il connait
- Ce qu'il peut déduire

- Exemple : Man in the middle
  - Observe les messages
  - Connait les clés publiques
  - Peut déchiffrer des messages s'il connait les clés

1) Dfférents intrus possibles

Actions possibles de l'intrus :

- Passif (man in the middle)
  - Observe les messages passant sur le réseau
- Participant :
  - peut lancer/participer à des sessions
- Actif :
  - peut forger des messages à partir de ce qu'il connait

2) Puissance de déduction de l'intrus

Que connait l'intrus ?

- Des connaissances initiales : les agents, les clés publiques, …
- Apprend à partir des messages observés et d'un pouvoir de déduction
- Peut connaitre des propriétés du protocole : par exemple sur les primitives cryptographiques
  - xor, chiffrement par bloc, …

Déduction (1)

Rappel format des messages :

- des clés : k, ka, inv(ka), kab
- du texte : m1, m2, ..
- des paires : m1 . m2
- du chiffrement : $\{m\}_{ka}$, $\{m\}_{inv(ka)}$, $\{m\}_{kab}$

Ka : clé publique, Inv(ka) : clé privée, kab clé partagée

Déduction (2)
Attaquant Dolev-Yao

- I peut manipuler les paires :
  - Si I connait m1 . m2 il peut en déduire chaque partie m1 et m2
  - Si I connait m1 et m2 il peut construire m1.m2

- Chiffrement/déchiffrement : cas symétrique
  - Si I connait m et kab il peut construire $\{m\}_{kab}$
  - Si I connait $\{m\}_{kab}$ et kab il peut en déduire m

- Chiffrement/déchiffrement : cas asymétrique
  - Si I connait m et ka il peut construire $\{m\}_{ka}$
  - Si I connait m et inv(ka) il peut construire $\{m\}_{inv(ka)}$
  - Si I connait $\{m\}_{inv(ka)}$ et ka il peut en déduire m
  - Si I connait $\{m\}_{ka}$ et inv(ka) il peut en déduire m

Exemples

Si I connait $k, \{s\}_c, a . \{c\}_k$

Il peut :

- trouver s
- construire $\{a\}_c$
- …

Chiffrement par blocs

Supposons que l'intrus sache que l'algorithme de chiffrement est un chiffrement par bloc qui vérifie donc la propriété :

$$\{x \,.\, y\}_k = \{x\}_k \,.\, \{y\}_k$$

Il peut alors construire $\{y \,.\, x\}_k$ à partir de $\{x \,.\, y\}_k$ sans connaitre la clé !

L’intrus en Avispa

```
% On va définir le terrain de jeu: quels agents, quelles sessions
role environment() def=

% dans Avispa, l’attaquant (intruder) existe sans être déclaré
const a, b        : agent,
 ka, kb, ki   : public_key,     % 3 agents, 3 clés publiques

intruder_knowledge = {a, b, ka, kb, ki, inv(ki)}          % ce que i connaît

 composition                  % On va jouer 3 fois le protocole:
        session(a,b,ka,kb)    % entre a et b
      /\ session(a,i,ka,ki)   % entre a et i
      /\ session(i,b,ki,kb)   % entre i et b
  end role
```

Preuve de propriétés

1. Modéliser les processus et vérifications possibles par chaque agent
2. Modéliser l'intrus (action + Dolev-Yao + connaissances)
3. Combiner agents, intrus et sessions : sémantique de traces
4. Enoncer des propriétés et les vérifier

Processus intrus

L'intrus devient le réseau et peut être malveillant :

- Tout message envoyé par un des agents est reçu par l'intrus
- Tout message reçu provient de l'intrus
- L'intrus enregistre tout ce qu'il voit passer (sa mémoire)

- Si intrus passif il se contente de renvoyer les messages reçus
- Si intrus actif il peut éliminer des messages, les rejouer plus tard, en fabriquer …
- Si un participant du protocole peut avoir des comportements malhonnêtes (exemple reprise d'un message)

Agents, Sessions (1)

- Un (très) gros automate !

1. $A \rightarrow B$: $\{Na \,.\, A\}_{Kb}$
2. $B \rightarrow A$: $\{Na \,.\, Nb\}_{Ka}$
3. $A \rightarrow B$: $\{Nb\}_{Kb}$

1'. $C \rightarrow B$: $\{Nc \,.\, C\}_{Kb}$
2'. $B \rightarrow C$: $\{Nc \,.\, Nb'\}_{Kc}$
3'. $C \rightarrow B$: $\{Nb'\}_{Kb}$

Les traces possibles :

- 1, 2, 3, 1', 2', 3'
- 1, 1', 2, 2', 3, 3'
- 1,2, 1', 2', 3, 3'
- …

+ l'intrus qui peut faire tout un tas de choses !

Agents, Intrus, Sessions (2)

+ l'intrus qui peut faire tout un tas de choses !

Intrus passif :

- ajouter entre chaque transition
- augmente ses connaissances (déduction Dolev-Yao)
- Déduction potentiellement infinie, prise en compte des propriétés à établir

- Intrus actif :
  - C'est plus compliqué … mais on y arrive

Preuve de propriétés

1. Modéliser les processus et vérifications possibles par chaque agent
2. Modéliser l'intrus (action + Dolev-Yao + connaissances)
3. Combiner agents, intrus et sessions : sémantique de traces
4. Enoncer des propriétés et les vérifier

Enoncer et vérifier une propriété de secret

$\Rightarrow$ s est secret entre A et B :

- exemple Na et Nb secret entre A et B

- s secret :
  - l'intrus ne peut jamais envoyer le secret s en clair sur le réseau

On raisonne sur toutes les traces :

- SAFE : aucune ne contient send(s)
- UNSAFE : une trace qui finit par l'envoi de s en clair sur le réseau

Secret en Avispa

```
transition
    0.  State  = 0 /\ RCV(start) =|>
        State':= 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
                   /\ secret(Na',secret_na,{A,B})

    2.  State  = 2 /\ RCV({Na.Nb'}_Ka) =|>
        State':= 4 /\ SND({Nb'}_Kb)

end role
```

- La valeur de Na est déclarée comme devant rester secrète entre A et B
- Secret_na est le nom qu'on donne à cette propriété
- Une attaque est une trace dans laquelle l'intrus peut envoyer en clair le secret (c'est-à-dire ici Na)

Enoncer et vérifier une propriété d'authentification

=>

- Authentifier l 'émetteur d'un message M
  - $\{m\}_{inv(k)}$
- Authentifier l'agent qui envoie un message (ou joue le protocole)
  - Plus complexe
  - On montre que seul A a pu envoyer le message

Enoncer et vérifier une propriété d'authentification

=>

- Méthode : A veut être sûr qu'il joue le protocole avec B
  - Il lui envoie $\{Na\}_{kb}$
  - Si A reçoit plus tard$\{\ldots Na \ldots\}_{ka}$ alors normalement cela vient de B

- 2 événements :
  - production du témoin Na pour B : witness(A,B, Na)
  - Vérification que B s'est bien authentifié auprès de A : request(B, A, Na)

=> Chaque trace doit contenir witness(A,B, Na) avant request(B, A, Na)

Exemple Attaque

A lire de bas en haut car Avispa « empile » les observations dans un état interne

% Reached State:

%

% request(b,a,bob_alice_na,Na(1),3)

% request(a,i,alice_bob_nb,Nb(2),6)

% witness(b,a,alice_bob_nb,Nb(2))

% contains(a,set_69)

% contains(b,set_69)

% witness(a,i,bob_alice_na,Na(1))

% …

La propriété de bob_alice_na n'est pas vérifiée, l'authentification de B par A via Na n'est pas assurée.

En effet Alice croit discuter avec l'Intrus alors qu'elle discute avec Bob. Imaginons que A croit avoir identifié l'intrus alors que c'est Bob qui lui fournit un service. L'intrus pourra alors demander d'être payé pour le travail fait par Bob !!!

TP avec l'outil Avispa

$\Rightarrow$Vérifie tous les comportements possibles (model cheking) pour secret, authentification forte, authentification faible

  - Répond SAFE ou donne une attaque (ou Timeout)

- TP :
  - Rejeu des exemples avec l'outil Avispa
  - Un protocole exploitant les propriétés du xor
  - Un protocole à corriger/vérifier
  - Une phase de handshake et ses propriétés

Pour conclure … et compléter !

⇒permet de vérifier des spécifications de protocoles contre un attaquant : les agents honnêtes/malhonnêtes, que peut faire l'attaquant …

⇒ Utilisé lors de la conception d'un protocole

- 5G, protocole carte bancaire, vote, échange de clé, authentification

- Une spécification vérifiée de TLS jusqu'à l'implémentation (miTLS):
  - www.mitls.org/
  - mitls.org/pages/attacks

Reste aussi à vérifier les programmes qui implémentent les protocoles

- Vérification de la structure des messages
- Hypothèses de la crypto parfaite

\+ :

- Correction (implémente bien le protocole)
- Absence de vulnérabilités dans le code (vuln régulière dans openssl par exemple)
- Résistance à d'autres d'attaques : canaux cachés, …

2A et 3A

Typage en Avispa

- Types : agent, public_key, symmetric_key (k and inv(k)), text (Nonce), nat, message
- Tous les types sont sous-type de messages

- Simulation typée : les agents n'acceptent que des messages correctement typés (correspondant au type attendu) (--typed model=yes (option par défaut))

- Simulation non typée : les agents ne vérifient pas le type des objets (--typed model=no)

Implémentation

- Un format de messages impliquant la structure :

11	T1	T2	m1	m2

- 11 : encodage du type paire

- Souvent mal implémenté ou mal décodé (TL !)

Crypto parfaite ?

=> Les preuves au niveau protocoles peuvent être combinées avec des preuves probabilistes sur les primitives cryptographiques

fr.wikipedia.org/wiki/Preuve_de_s%C3%A9curit%C3%A9

- Théorie de l'information
- Théorie de la complexité : logarithme discret (DH), factorisation (RSA), …

Cours 2A : Codes et Crypto

Preuve de sécurité des primitives cryptographiques

- Les problèmes utilisés par la cryptographie sont tous dans NP :
  - il est « facile » de coder un message, ou de décoder un message quand on en possède la clé.
  - En revanche*, en l'état actuel des connaissances*, toutes les méthodes existantes pour casser ces codes sont exponentielles en la taille de la clé.

C'est la disproportion pratique entre le temps de codage ou décodage avec clé d'une part, et de cassage d'autre part, qui rend les méthodes utiles.

Classes de complexité : P (polynomial), NP (non déterministe polynomial), P=NP est un problème ouvert.

Cryptographie post quantique

- Il n'y a pour l'instant pas d'objection théorique à l'existence d'algorithmes polynomiaux de cassage des codes utilisés actuellement, mais juste le constat pratique que ces problèmes résistent aux efforts soutenus de la communauté depuis suffisamment longtemps.

- Notons par ailleurs que les ordinateurs quantiques, si on arrive à en construire de « taille » (nombre de qbits) suffisante, permettraient de casser des systèmes comme RSA via l'algorithme de Shor.

Autres propriétés

- Autres propriétés sur les traces : non-répudiation, …

- Propriétés plus complexes comparant des traces d'exécution :
  - Strong secret (un attaquant ne voit pas de différence si les secrets sont différents)
  - Anonymat

Un TD sur le vote électronique !

Notre message 1A

- La sécurité se pense en amont (et jusqu'à son implémentation)
- Importance des outils : permet d'établir/rejouer des vérifications
- Même démarche sur les programmes (les vulns), les réseaux, les OS, …