Avispa un outil de vérification de protocoles

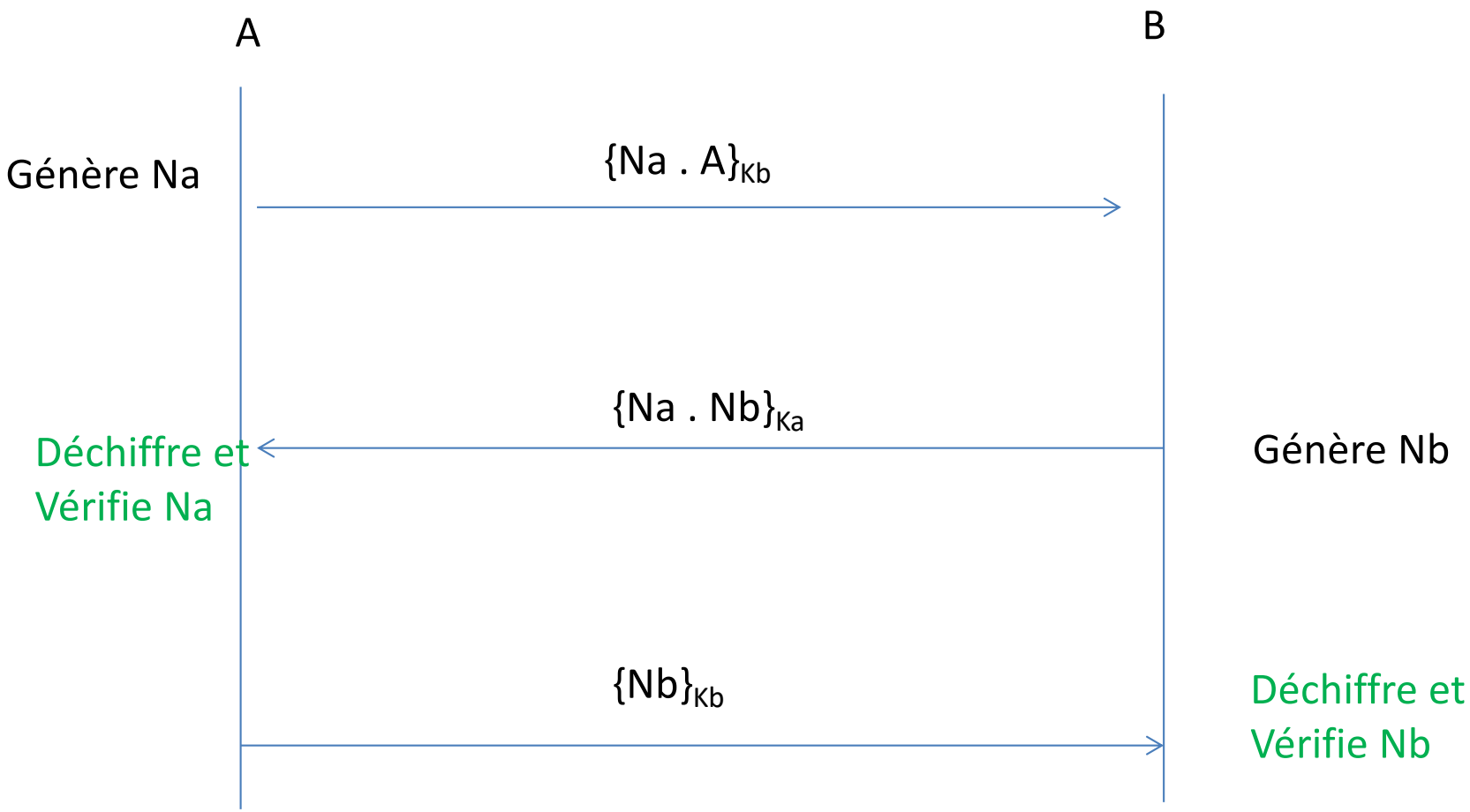
Format des fichiers hpls
de description des protocoles cryptographiques
(+ propriétés attendues)

Rappel: 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)

Rappel: Needham-Schroeder



Description des rôles de chaque agent

```
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 : une constante
- Na’ : une nouvelle valeur

- Types : agent, public_key, symmetric_key (k and inv(k)), text (Nonce), nat, message

- Tous les types sont sous-type de message

De manière symétrique

```
role bob(A, B: agent,
         Ka, Kb: public_key,
         SND, RCV: channel (dy))
  played_by B def=

    local State : nat,
        Na, Nb: text

    init State := 1

    transition

      1.  State  = 1 /\ RCV({Na'.A}_Kb) =|>
          State':= 3 /\ Nb' := new() /\ SND({Na'.Nb'}_Ka)

      3.  State  = 3 /\ RCV({Nb}_Kb) =|>
          State':= 5

  end role
```

Les sessions

```
role session(A, B: agent, Ka, Kb: public_key) def=
  local SA, RA, SB, RB: channel (dy)
  composition                                  % Une session de Needham-Schroeder va composer
              alice(A,B,Ka,Kb,SA,RA)           % un automate alice avec un automate bob
    /\ bob  (A,B,Ka,Kb,SB,RB)                  % paramétrés avec les identités des agents
end role                                       % A jouera le rôle d'alice, B celui de bob
role environment() def=          % On va définir le terrain de jeu: quels agents, quelles sessions
      const a, b       : agent,  % dans Avispa, l'attaquant (intruder) existe sans être déclaré
            ka, kb, ki  : public_key,  % 3 agents, 3 clés publiques
            secret_na, secret_nb,      % 4 étiquettes qui serviront pour les propriétés
            alice_bob_nb,
            bob_alice_na : protocol_id
      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
```

Les propriétés- le secret

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)

Les propriétés – l'authentification

Propriété attendue : l'agent A veut authentifier l'agent B par le témoin Na = c'est bien B qui lui répond.

Autre formulation : l'agent B demande à être authentifié auprès de A par le témoin Na

Rôle A :

State':= 2 /\ Na' := new() /\ SND({Na'.A}_Kb) /\
witness(A, B, bob_alice_na, Na')

- La valeur de Na doit permettra à A d'authentifier B : seul B peut normalement retourner ce nonce
- La propriété s'appelle bob_alice_na

Rôle B :

State':= 5 /\ request(B,A,bob_alice_na,Na)

A la fin de son processus B veut être authentifié auprès de A par Na

Authentification - suite

- Trace correcte : chaque request est précédée du witness associé :

  … witness(A, B, prop_B_A_na, Na) … request (B, A, prop_B_A_na, Na)

- Une attaque : la condition ci-dessus n’est pas respectée

Exemple

A lire de bas en haut pour remonter de chaque witness vers une request

% 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 !!!

Vérification

On déclare à la fin du fichier les propriétés qu'on veut vérifier :

```
goal
    secrecy_of secret_na, secret_nb
    authentication_on alice_bob_nb
    authentication_on bob_alice_na
end goal
```

=> Arrêt sur la première attaque

Résultat possible

- Safe : pas d’attaque possible
- Unsafe + une trace d’attaques
- Timeout