Security Model

# Examen

60% d’examen terminal + 40% de CC

CC 🡪 60% d’examen intermédiaire + 40% de TP

Exam intermédiaire : 1h sur le TD4 le 7 février

TP noté : 21 février

# Rappel sur la cryptographie

## Schéma de chiffrement

* Clefs symétrique (AES)
* Clefs asymétrique (RSA, Elgammal)
* Signature
* Fonction de hash

# Computationnal World

Message = chaine de bits (10100011001100)

Attaquant : Probabilité polynomiale machine de Turing (PPTM)

Dans le Symbolic World 🡪 message chaine de termes ({n, H(B)}k(A))

Attaquant : système de déduction (Dlev-Yoz)

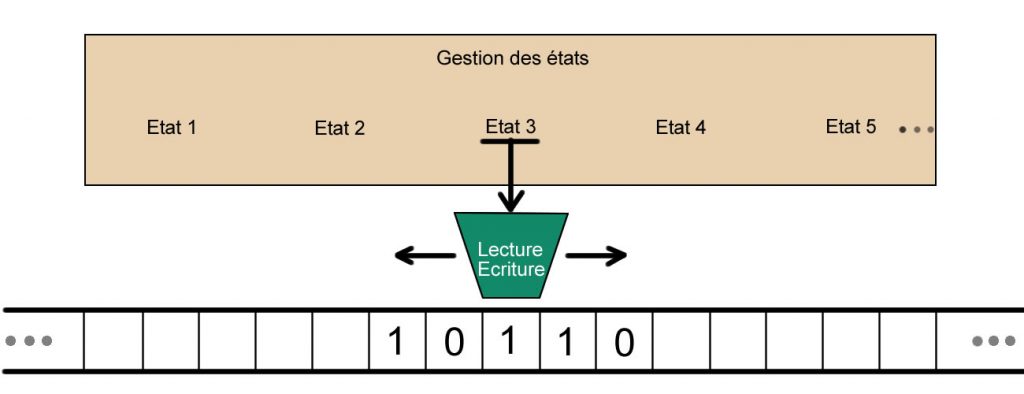
## Alan Turing

En 1936, il a designé le Machine de Turing 🡪 faire une machine qui pouvait résoudre n’importe quel problème.

Le premier prototype : LA Bombe pour casser ENIGMA.

Machine de Turing :

* Ruban infini qui contient les données et les programmes
* Une machine



* READ le ruban
* MOVE à droite ou gauche le ruban ou IDLE
* WRITE 0 ou 1

Probabilité : la machine de Turing a un générateur de nombre aléatoire.

Tous les systèmes de chiffrement peuvent être casser par l’approche force brut, donc de tester toutes les clefs possibles 🡪 2^|k| avec |k| la taille de la clef.

Pour RSA il faut essayer 2^|k| clefs.

### 2 books :

* The handbook of cryptography, Menezes Etal
* Introduction to the modern cryptography, Jonathan Katz / Yehudra Lindell

## Fonction négligeable

### Définition

On appelle une fonction u : N 🡪 R\* négligeable, si pour tout polynôme positive P, il existe un N tel que :

Pour tout n > N

U(n) < 1/P(n)

Exemple : (1/2)^n est une fonction négligeable

n^-logn est négligeable

### Problèmes difficiles

Factorisation :

p\*q = n -> p et q différent et premiers

Trouver p et q à partir de n.

* RSA sécurité est basé sur ce problème.

### Logarithme discret

n, g, g^y modn 🡪 ? y

### Composition de Diffre-Hellman (CDH)

g, ,g^x, g^y 🡪 ? g^xy

### Deciionnel Diffre-Hellman (DDH)

(G, g^a, g^b, g^ab) r (g, g^a, g^b, g^r)

R est un nombre aléatoire

Si on résout le premier on résout le 2e et on résout le 3e.

La technique utilisé est appelée la preuve par réduction !

### Comment ?

Prouver que la sécurité de schéma de E est en dessous de l’assumption de H ?

⬄

Si l’assumption de H est vraie alors E est sécurisé

⬄

Si E n’est pas sécurisé alors l’assumption H est fausse.

Remarque logique :

(A => B)

⬄

(nonB => nonA)

⬄

Il existe A un adversaire qui casse le schéma E alors on peut construire un adversaire B tel que B casse H.

## One wayness

Depuis un cipher c = {m}pk

Il est difficile de retrouver un message m.

Pr[it(c) -> m] est négligeable dans le périmètre de la sécurité, où c = {m}pk

Clefs RSA de taille 1 bit :

C = {m}pk 🡪 proba de casser ½

Clefs RSA de taille 2 bits :

* ½^2 = ¼

Clefs RSA de taille k bits :

* (1/2)^k est une fonction négligeable

Propriété asymptotique.

## Indistinguishability (IND)

M0 et m1 2 textes différents avec la même taille.

C = {mb}pk

B <- {0,1}

Pouvez-vous déterminer lequel message est chiffré dans c ?

Oui c’est possible, on a une chance sur 2, il suffit de choisir aléatoirement.

## Non Malleability (1991)

C = {m}pk

R « selector »

C’ = {m’}pk tel que m’ != m et m’ = R(m)

Si on prend la chiffrement fully homomorphique schéma :

{a}pk \* {b}pk = {a\*b}pk

C -> c\*c’ = {m\*m’}pk

C’ = {m’}pk

Donc fully homomorphique est malléable. (Pour cet exemple)

**NM => IND => OW**

## Adversaires

Modéliser par PPTM :

« Malin »

« Puissance des ordinateurs limitées »

Adversaire a accès à un Oracle. Un programme qui donne tout le temps la bonne réponse.

Oracle de chiffrement E

Oracle de déchiffrement D

E prends un message m et retourne le cipher de m.

D prends un cipher c = {m}pk et retourne le message correspondant ou un symbole d’erreur si le cipher est pas bon.

3 Intruders :

* CPA : Choosen Plaintext Attack
  + L’adversaire has access to polynomial of time pour l’Oracle de chiffrement avant et après la réception du challenge C.
* CCA 1 : Non adaptative Chosen Cipher Text Attack
  + CPA + Access, pour l’Oracle de déchiffrement D avant d’avoir le challenge (« Lunch Time Attack »)
* CCA 2 : Adaptative Chosen Cipher Text Attack
  + CCA 1 + accès à l’oracle de déchiffrement après avoir reçu le challenge C

CCA 2 => CCA 1 => CPA

NM – CPA 🡨 NM – CCA 1 🡨 NM-CC2

IND – CPA 🡨 IND- CCA 1 🡨 IND – CCA 2

Sécurité forte (OHEP-RSA)

Sécurité minimum (ElGammal)

OW - CPA

Sécurité faible (RSA)

## IND

A = (A1, A2)

A1 prends pk

A1 choisis m0 et m1 différents mais de même taille.

B 🡨 {0,1}, c = {mb}pk donné à A2

A2 répond b’

Eis IND-XXX securisé si

Pr[b = b’] – ½ est négligeable

# Apologize

<a,b> ou (a,b) ou a||b est la pair de a et b -> on peut extraire facilement a ou b.

# Recall

OW -> Plastic Translucide Bag

IND -> Black Plastic Bag

Non Malleability -> Black Box

# IND-XXX Game

Soit S = (k,E,D) un schema de chiffrement.

Un adversaire A = (A1, A2)

b <-$ {0,1}

INDxxxb (A1) :

* Génère (pk, sk) = K(y) -> paramètre de sécurité
* (s, m0, m1) <- A1(y, pk)
* M1 != m0 et |m1| = |m0|
* B’ <- A2(s,y,pk, {mb}pk)
* Return b’

AdvINDxxx (y) :

Pr [b’ <- INDxxx0(A,y) : b’ = 1] – Pr [b’ <- INDxxx1(A, y) : b’ = 1]

S is INDxxx secure

Si AdvINDxxxs,A(y) est négligeable

XXX :

* CPA : O1 = O2 = {E} (oracle de chiffrement)
* CCA1 : O1 = {E,D} et O2 = {E}
* CCA2 : O1 = O2 = {E,D}

On peut prouver que :

AdvINDxxxs,A(y) = 2 Pr[b’ <- INDxxxb(A) : b’ = b ] – 1

## Definition of Non-malleability

S = (k, E, D)

A = (A1, A2)

* On donne a A1 la clef publique pk
* A1 choisi M une liste de textes
* M et m\* sont 2 texte choisi aléatoirement dans M et c = {m}pk : le tout donner à A2
* Les sortis de A2 sont des relations binaire R et un cipher text c’
* M’ = D(c’)
* Pr [R(m, m’)] – Pr [R(m, m\*)] est négligeable

ExpATK-bS,A (n) :

(pk, sk) = K(n)

(M,s) = AO1(pk)

X0, x1 <- M

y = {xb}pk

(R, y->) = AO22(s, n M, y)

If (y appartient pas à y-> ^ | appartient pas à x-> ^ R(xb, x->) alors

Sortie 1

Sinon

Sortie 0

RSA est seulement OW

RSA-OAEP est DND-CCA2 et NM-CCA2

CrammerShouq est IND-CCA2 et NM-CCA2

# Crammer Shouq (1998)

## Rappel El-Gammal

* Clefs secrete c
* Clefs publique k = g^x mod
* C = (g^r/u , h^r \* M/v)
* M = v/u^x

## Crammer shouq

Algorithme de génération de clef

On prends de manière aléatoire (x, x2, y1, y3, z)

C = g1^x1 g2^x2

D = g1^y1 g2^y2

H = g1^z

Clefs publique : (c, d, h)

Clefs secrete : (x1, x2, y1, y2, z)

## Cyclic order groupe

Z/pZ avec p premier

* Element de Z mod p

Z/5Z exemple

Generation est un element tel que

G1, g2, … gd

Ou d est l’ordre du groupe are all element of the group

Avec 2 : 2^1 = 2 / 2^2 = 4 / 2^3 = 8 mod 5 = 3 / 2^4 = 1 / 2^5 = 2

## Algorithme de chiffrement pour un message m

K un nombre aléatoire

On prend u1 = g1^k et u2 = g2^k

e = h^k \* n

alpha = H(u1, u2, e)

v = c^k \* d^kalpha

cipherText est (u1, u2, e, v)

## Algorithme de déchiffrement (clefs secrète connu)

Avec le cipher text on ajoute :

Alpha = H(u1, u2, e)

On vérifie v = u1^x2 \* u2^x1 \* (u1^y1 \* u2^y2)^alpha

m = 2 / (u1^z)

2/(u1^z) = h^k \* m/g1^kz = g1^kz \* n /g1^kz = m

## Key anonymity

Quelle clef utilisée ? (pour un même cadenas par exemple)

Avec un cadenas à code on a la key anonimity

RSA est key anonymous ?

pk, pk2, m

c = {m}pk -> ? b

c = m^eb mod nb pour RSA

pk = {e, n}

sk = d

c = x^e mod n

n = c^d mod n

Pour trouver quelle clef de RSA on chiffre m avec la première on regarde si c’est égal à c et si non on test la deuxième. Donc RSA n’est pas clef anonyme.

Les algorithmes de chiffrement déterministe ne sont pas clef anonyme.

El-Gammal et Crammer Shouq sont clef anonyme sous la DDH assumption, RSA-OEP aussi.

IKAxxx^b(A,n) :

(pk0, sk0) = k(y)

(pk1, sk1) = k(y)

(s,m) = A1^O1(pk0, pk1,n)

B’ <- A2^O2 (s, pk, n, {m}pkb)

Return b’

Adv^IKAxxx s,H (n) = Pr [b’ <- IKAxx^1 (A, n) : b’ = 1] – Pr[b’ – IKAxxx^0 (A,n) : b’ = 1]

# Signature numérique

Une signature (K, S, V) avec S un algo de signature, V un algo de vérification

(sk, pk) = K(n) un générateur de clefs

Sigma = sign(m, sk)

V(sigma, pk, m) = OK ou KO

Donc on a :

V(sign(m, sk), pk, m) = OK

Une signature ne doit pas être reproductible en anglais 🡪 UNFORGEABILITY

On donne à un adversaire A la clef publique pk, le message m et sigma la signature.

A peut-il trouver un sigma’ une signature de m’ valid

Une signature naïve est reproductible. Les sceaux des rois sont reproductibles.

(sk, pk) = K(y)

(m, sigma) = A^sign(pk,m) avec sign un oracle de signature qui maintient une liste Q de toutes les requetes de A.

A gagne si et seulement si

(m, sigma) appartient pas à Q

Et V(pk, m, sigma) = OK

Pr [(m, sigma) = A^sign(pk, n) : V(pk, m, sigma) = 0] est négligeable

## RSA signature

Sk = (n, e)

Pk = d

Sigma = sign(m, sk) = m^d mod n

Verify(sigma, m, pk) = m = ? sigma^pk mod n

RSA signature est-elle reproductible (unforgeable -> EUF-CMA : Existentiel Unforgeability under Chosen Messages Attack)

Algo de vérification :

Sigma, m, pk

M = ? sigma^pk mod n = sigma^e mod n

Peut-on forger une nouvelle signature valide sans sk 🡪 (m’, sigma’) valide ?

Attack 1 :

On prends un nombre aléatoire s

On prend m’ = s^e mod n

Sigma’ = s

N’ = s^e mod n

C’est une signature valide §

Attack 2 :

(sigma1, m1) une signature valide de m1

On choisit m2 tel que :

M1 M2 = m mod n

On demande une signature de m2

Sigma2 = m2^d mod n

Sigma = sigma1 sigma2 est une signature valide de m

### Probabilistic Full Domain-Hash RSA

Signature :

Sigma = (r, s) = (r, y^d mod n)

Ou y = H(r || m) et r un nombre aléatoire

Vérification : s^e mod n = H(r||m)

# Symbolic verification for cryptographic protocol

On considère le schéma parfait de chiffrement.

La seule façon d’ouvrir un message chiffré avec un chiffrement parfait est de connaitre la clef.

On cherche les attaques logiques.

Exemple naïf :

Alice 🡪 {12h30}kAB 🡪 Bob avec kAB une clef partagée entre A et B

Bad guy (renvoie le message d’avant)

* {11h30}kAB 🡪

On connait 2 types d’attaquants :

* Attaquant passif
  + Ecoute les messages du réseau
* Attaquant actif
  + Ecoute, stop, réponds, renvoie, forge, joue avec les messages et sessions

Les attaquants sont le Réseau :

Capacité des attaquant : pour ça un modèle a été crée avec un système de déduction.

On utilise le système de deduction de Dolev-Yao (1983)

Soit T un set de tous les messages que l’attaquant a later over the network.

Règle de deduction : H1 . H2 . H3 (R) / C

H les hypothèses / C la conclusion / R la règle

A partir de T, l’ataquant peut déduire le message u 🡪 T |- u

Dolex-Yao deduction système :

(u appartient à T / T |- u) (A) 🡪 Axium

(T |- n . T |- k / T |- {m}k) (C) 🡪 cipher

(T |- x . T |- y / T |- <x, y>) (P) 🡪 Pairing

(T |- m{k} . T |- k) / T |- m) (D) 🡪 Decryption (symetric)

(T |- <x, y> / T |- x) (U) 🡪 Unpairing

Exercice :

Depuis T = ({m}k, <a, k>

Peut-on déduire M ? T |- m ?

1/ ({m}k appartient à T / T |- {m}k) (A)

2/ (<a,k> appartient à T / T |- <a, k>)(A) et (T |- <a,k> / T |- k) (U)

Donc

1/ . 2/ /T |- m (A)

VICTOIRE !!!

### Définition :

La taille d’une preuve P, note |P| est le nombre de nœuds de la preuves (exemple précédent taille de 6)

### Définition :

A prouve P of T |- u est minimal, s’il n’existe pas de preuve P’ qui prouve T |- u tel que |P’| < |P|

### Définition

Une preuve est simple si un nœud n’apparait qu’une seule fois dans chaque branche.

### Définition synthactic subterms

S(t) la plus petite liste tel que :

* t appartient à S(t)
* (x, y) appartient S(t) => x appartient à S(t) et y appartient à S(t)
* {x}y appartient à S(t) => x appartient à S(t) and y appartient à S(t)

T = ( {m}k, <a, k> }

S(T) = (m, a, k, T)

T = (<a, {b}k>, ({m}a<k,b>)

S(T) = {T, a, {b}k, b, k, {m}a, <k, b>, m}

Le nombre de terme peut être infinis.

## Idée de la localité de Marc Halleston

But trouvé T|-w

Un carré qui représente S(T, w) et à l’intérieur des cercles concentrique de plus en plus grand (mais jamais plus que le carré) Un cercle correspond à Ti

Une preuve (T |- w) est S-local si tous les nœuds appartiennent à S(T, w) = S(T) U S(w)

Un système de déduction est S-local si pour une preuve P de T |- w alors il existe une preuve P’ S-local de T |- w.

Doles-Yao est S-Local

### Théorème de la localité de Marc Alleston

Si un système de preuve est S-local, alors il existe une procédure en temps polynomial pour décider la déductabilité INP

**Preuve** : On construit un algorithme physique pout T0 |- w.

1/ Compute S(T0, w)

2/ While (il existe s appartenant à S(T0, w) tel que T |- s et s n’appartient pas à T

T <- T U (s)

Output u appartient à T.

Exemple :

T = ({a}k, <b, a>)

Preuve : T |- k est impossible !

Delta = S(T, k) = ({a}k, <a,b>, k, a, b)

1. Tout les termes déductible à partir de T en 1 étape :

({a}k appartient à T / T |- {a}k)(A)

(<b,a> appartient à T / T |- <a,b>) (A)

1. Tous les termes en 2 étapes

(D) pas possible

(U) (T |- <b ,a> / T |- b)

(U2) (T |- <b,a> / T |- a)

A et B appartiennent à Delta

(T |- {a}k . T |- {a}k / T |- {{a}k{a}k}) (C)

(T |- {a}k . ™ |- <a,b> / T |- {{a}k}<a,b>) (C)

(T |- <a,b> . T |- {a}k / T |- {<a,b>}{a}k}) (C)

(T |- <a,b> . T |- <a,b> / T|- {<a,b>}<a,b>)

No new terms for C aucun n’est dans Delta.

1. Tous les termes deductible en 3 étapes

(D) no

(U1) oui mais tous les termes sont déjà trouvé

(U2) pareil

(C) On peut mais pas de nouveau termes

(P) des termes sont produit mais ne sont pas dans Delta

On arrive à un point fixe.

## Active intruder

Un intruder actif contrôle le réseaux, joue et rejoue les messages, stop et notifie les communications.

## Needham Schroeder

Nounce Na generating Nounce NB générer par B

A(skA, pkA) 🡪 {Na, A}pkb 🡪 B (skb, pkb)

🡨{Na, Nb}pka 🡨

🡪{Nb}pkb 🡪

On vient de partager Nb

17 ans plus tard on trouve une attaque :

### Man in the middle attack

A(skA, pkA) I(skI, pkI) B(skB, pkB)

🡪{Na, A}pkI🡪 🡪 {Na, A}pkB 🡪

🡨 🡨 🡨 {Na, Nb}pkA 🡨

🡪{Nb}pkI🡪 Nb 🡪{Nb}pkB

Attaque du côté de Bob, Bob pense parler avec Alice.

A lack oh authentification

A B

🡪{Na, A}pkB 🡪

🡨 {Na, Nb, B}pkA 🡨

🡪{Nb}pkB 🡪

Pourquoi une vérification formelle des protocoles de chiffrement est-elle difficile ?

* Branches infinie (taille des messages pas bornée)
* Profondeur infinie (nombre de session non bornée) A – Talk 🡪 I == une session

## Definition

### Action

* Une action est un couple (recv(u), send(v) tel que n appartient à T(F, X) U {Init}

Et v appartient à T(F, X) U {stop}

Denoted u 🡪 v

* T(F, X) un set de termes qui peuvent construire depuis un set de fonction F et le set de variable X

Exemple : T(h :x -> h(x), y) = {y, h(y), h(h(y)), h(h(h(y))), …)

Remarque : Une constante est juste une fonction de ?? 0

Exemple : a :🡪 a une constante

Init 2 termes spéciaux

Stop

### Rôle

* Un rôle est une séquence finie d’actions. (u1 🡪 v1)(u2🡪 v2)…tel que var(vi) inclus dans U(j de 1 à i)var(j)

### Protocole

* Un protocole P est un set fini de rôles P = {R1, …, R4}

Exemple :

A 🡪(Na, A)pkB 🡪 B

🡨 (Na, Nb)pka🡨

🡪(Nb)pkb 🡪

Donner les roles de A et B pour le Weedhome Schroeder protocole :

Rôle A :

((Init, Xb 🡪 {Na, A}pk(Xb)

(Na, XNb}pkA 🡪 {XNb}pk(Xb))

Rôle B :

({XNb, XA}pk 🡪 {XNA, NB}pk(XA))

({Nb}pkB 🡪 Stop)

### State

* T est un set de grounds termes (termes sans variables)
* P un protocole
* Un état est un couple (T, P)

### Transition

* C’est une relation entre des états

(T, P) 🡪 sigma🡪 (T’, P’)

Sigma est un substitu sigma :X🡪T{F, X}

P = Union(pour i allant de 1 à k)Ri (Rôle de i)

Sigma possible : T |- sigma(ui)

Don(sigma) = vars(ui)

* T’ = Tu {sigma(vi)}
* Pour tout j <> i, Rj appartient à P’
* P’ = ( P\Rj) U sigma(Rj)\*

### Secrecy

C’est quelque chose que l’intruder ne peut pas déduire.

Let T un set de grounds termes, un protocole P qui ne préserve pas the secrecy du grounds terme sigma pour T si il existe un état (T’, P’) tel que

* T’ |- s
* (T, P) 🡪sigma\*🡪 (T’, P’)

S’il n’existe pas tel que (T’, P’), on dit que P préserve le secret.

### Protocol order <q

Un protocol P définie un ordre partiel <p en action

P tel que (ui, vi) <p (uj, vj)

R inclus dans P, R = (u1, v1) (ui, vi) (uj, vj) (un, vn)

### Exécution order <E

C’est un ordre total sur le subset A of action de P un protocole compatible avec <p stable predecessor si b appartient à A, a <p b alors a appartient à A et a <e b.

### Secret over <E

Soit une exécution order <E de P. On assume que (u1🡪v1) <E .. <E (un 🡪 vn)

<E ne preserve pas le secret de s, pour un T donné si il existe sigma1, …, sigman.

(T, P) 🡪sigma1 🡪(T1, P1) … 🡪 sigma🡪(Tn, Pn)

T U {sigma1(v1), …, sigman(vn)) |- S

On considère un nombre finis de session.

### Logiciel pour faire des trucs

* Scyther
* AVISPA

## Systèmes de contraintes

### Definition :

Une contrainte est une expression :

T ||- u

Où T est un set de termes et u un terme.

Un système de contrainte C est un set finis de contraintes

U(i allant de 1 à n) Ti ||- ui tel que

* Monotonicité : Ti inclus dans Ti+1
* Si Ti ||- ui appartient à C alors

X appartient à Vars(Ti) alors

Tj = min(T’ | T’ ||- v appartient à C, X appartient à Vars(i), j< i)

* Transformation d’un protocole en un système de contraintes :

T1 ||- u1

T2 = T1 U {v1} ||- u2

T3 = T2 U {v2} ||-

Tn+1 = Tn ||- {un} = Secret

(u1 🡪 v1) <E … <E (un, vn)

C a une solution si <E ne prouve pas le secret des termes de S.

## Unification

### Definition

2 termes t et s sont unifiable if il existe une substitution J telle que J(t) = J(s)

### Exercice

|  |  |  |
| --- | --- | --- |
| S = a | T = x | Sigma :x 🡪a sigma(s) = a = sigma(t)  YES |
| S = a | T = p(a) | NO |
| S = p(a, x) | T = p(y, b) | Sigma : (x🡪b, y 🡪a)  YES |
| S = p(f(x), g(sigma)) | T = p(f(a), y) | Sigma x 🡪a, y 🡪g(z)  Sigma2 x 🡪 a, z 🡪b, y 🡪f(b) |

s = p(f(x), g(z))

t = p(f(a), y)

De façon à ce que sigma1(s) = sigma1(t) et sigma2(s) = sigma2(t)

Sigma1 = {x -> a ; y -> g(z))

Sigma2 = {x -> a ; z -> b ; y -> g(b))

## Most Generated Unifer (MGU)

Le MGU entre 2 termes s et t est mgu(s,t) si pour tout sigma tel que sigma(s) = sigma(t)

Il existe O tel que O = sigma(t,s)O

Sigma2 = sigma1\*O où O :{z 🡪 b}

### Exemple :

s = f(x, g(a), g(z))

t = f(g(y), g(y), g(g(x)))

mgu(s, t) = {x -> g(y) ; y -> a ; z->g(x)}

sigma = {x -> g(a), y -> a ; z -> g(g(a))}

sigma = Omgu(s,t)

R1 : C U {T ||- u} 🡪 C si T U {x | T’ ||- x € C, T’ inclus dans T} ||- u

R2 : C U {T ||- u} 🡪 sigma(C) U | sigma(T) ||- sigma(u)

* Sigma = mgu(t, u), t € s(T) avec t et u no variable

R3 : C U {T ||- u} 🡪 sigma(C) U (sigma(T) ||- sigma(u)

* Sigma = mgu(t, t2), t, t2 € S(T) avec t et t1 no variable

R4 : C U {T ||- (u)v} 🡪 C U {T ||- u} U {T ||- V}

* Séparation du chiffrement u | v en 2

R5 : C U {T ||- <u, v>} 🡪 C U {T ||- u} U {T ||- v}

R6 : C U {T ||- u} 🡪 impossible si T = vide ou var(T) = var(u) = vide and T |-/ u)

Soit P un protocole 🡪 Roles 🡪Système de contraintes C

C Apply R1 to R6 autant de fois que l’on veut.

Si on trouve une solution alors on peut trouver une attaque sinon c’est sécurisé.

### Lemmes (Préservation)

Le règles de simplifications transforme un système de contraintes en un autre système de contraintes.

### Lemmes (Correctness)

Si C ~>sigma C’ alors si O est une solution de C’ alors sigmaO est aussi une solution de C.

### Lemme (Terminaison)

Les règles de terminaisons se termine toujours

### Definition (Solved form)

Un système de contrainte C est un solved form si C = bottom ou si pour chaque contraintes elles sont de la forme T ||- x où x est une variable et T != vide.

### Lemme

Tous les systèmes de contraintes dans une forme résolus différentes de « bottom » ont au moins une solution

### Lemme (Complétude)

Si C un système de contraintes qui n’est pas dans une forme résolue et si sigma est une solution de C alors il existe O, z tel que C~>sigma C’ et sigma = Oz et z est une solution de C’.

### Theorem :

La préservation du secret pour un protocole avec un nombre borné de session est décidable.

### Preuve :

1/ deviner un interleaving

2/ Construire le système de contrainte C

3/ utiliser les règles de simplifications

On choisi une solution, s’il existe C’ sous forme résolue tel que C’ != bottom et C ~>sigma C’

4/ Cela se termine toujours

# Outils

Pour vérifier les protocoles cryptographiques

3 outils :

* Avispa
  + Composé de plusieurs logiciel
  + OFMC
  + cl-ATSE
  + SATMC
  + TA4SP
* Scyther
* Proferif

## AVISPA

A 🡪 {Na, A}pkB 🡪 B

🡨 {Na, Nb}pkA 🡨

🡪{Nb}okA 🡪

Rôle Alice (A, B : agent,

Ka, kb : public key,

SND, RCV : channel(dy))

Played\_by A def =

Local state : nat,

Na, Nb : text

Init state := 0

Transition

O. state = 0 /\ RCV(state) =>

State’ := 2 /\ N’a = row() /\ SND({Na’} = k5}

/\ secret(Na’n na} A, B}

2. state = 2 /\ RCV({Na.Nb’} -ka) => state’ := 4 /\ SND({Nb ‘kb)

End Rôle

Rôle session (A, B : agent,

Ka, Kb public key)

Def = local SA, RA, SB RB

**Composition :** alice(A, B, Ka, Kb, Sa, Sb)

/\ Bob (A, B, Ka, Kb, Sb, Rb)

End rôle

Rôle environnement() def =

Cout a, b agent

Ka, Kb, K : public key

na, nb : protocole

Connaissances de l’intrus : {a, b, Ka, Kb, Ki, inv(Ki)}

Composition :

Session(a, b, Ks, Kb) /\

Session(a, i, Ka, Ki) /\

Session (i, b, ky, Kb)

End rôle

Le secret qui est le but, na, nb

End goal

Environnement ?

## Scyther (Ias commun)

Protocole NS3(I, R) {

Rôle I {

Count : Nonce,

Var nr = Nonce,

Send\_1(I, R, {n, Ipk(R).

Read\_c (R, I, (ni, nr)pk(I)

Send\_3 (I, R, {nr}pk(R))

Claim\_1(I, Secret, n)

Claim\_2(I, N, synch)

}

Rôle R {

Var n := nonce ;

Const nr : nonce ;

Read\_1 (I, R, {ni, I}pk(R)) ;

Send\_2 (R, I, {ni, nr}pk(I)) ;

Read\_3 (I, R, {nr} pk(R)) ;

Claim\_R1 (R, Secret, nr) ;

Claim\_R2 (R, Ni, synch) ;

## Proverif

Utiliser des clauses d’Horn

H1 … Hn => C

* Il existe différentes résolution rapide et efficace

2 formats d’entrée

* Clauses d’Horn
* Pi-calcul

Proverif fonctionne pour un nombre non borné de session

* Il fait une approximation
* Peut produire de fausse attaque
* A partir de la modélisation des nonces

Un nonce est modelisé de la façon suivante : Na [x, A, B]

Fonction de parsion values

Dans proverif l’identité sont les clefs public :

pk(SA[]) 🡪 clefs publique d’Alice

avec SA la clef privée

pred c/1 elimVar, decompData

nonverif c :x

fun pk/1

fun encrypt/2

query c : secret []

reduce

(\*Init\*)

C = c[]

C = pk(SA[]) ;

C = pk(SB[]) ;

C = x & c : encrypt(m, pk(x)) -> c :m ;

C :x & c : y -> c ; encrypt(x,y) ;

(\*A\*)

C : pk(x) -> c : encrypt(Na [Pk(x)], pk(SA[])), pk(x)) ;

C : pk(x) & c :encrypt((NA[pk(x)], y), pk(SA[]))

* C : encrypt(y\*), pk(x))

(\*B\*)

C : encrypt((X, y), pk(SB[])) 🡪 c : encrypt((x, Nb[pk(x,y)]), y),

C : encrypt((x, pk(SA[])))

C : encrypt(Nb[x, pk(SA[])], z) pk(SB[])

* C : encrypt(secret [])