Plan

1. Sniffer
2. Outils de Pentest
3. Tamarin

Plan

1. Sniffer
2. Outils de Pentest
3. Tamarin

Bluetooth 4.0

- nRF51 DK & nRF-Sniffer
  `https://www.nordicsemi.com/eng/Products/nRF51-DK`
  `https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF-Sniffer/`
- Adafruit Bluefruit LE Sniffer
  `https://www.adafruit.com/product/2269`
- Ubertooth One
  `http://ubertooth.sourceforge.net/hardware/one/`
- TI BLE Sniffer (Texas Instruments)
  `http://www.ti.com/tool/packet-sniffer`

Bluetooth 5.0

- nRF52 DK & nRF-Sniffer-v2
  `https://www.nordicsemi.com/eng/Products/nRF52-DK`
  `https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF-Sniffer/`
- infor-link
  www.infor-link.com

Plan

1 Sniffer

2 Outils de Pentest

3 Tamarin

Outils de Pentest

- Crackle : crack BLE Link Layer encryption
- BtleJuice : framework complet pour faire des Man in the Middle attack
- Gattack : outil permettant de tester plusieurs attaques
- BlueBorne exploit : exploite la faille BlueBorne
- Autres outils utilisées : wireshark, hcitool, hcidump, gatttool,...

- Bluelog : scanner Bluetooth
- BlueRanger : Permet de localiser les devices.
- BlueSnarfer : Permet de récupérer les fichiers a partir du Bluetooth
- Btscanner : Permet de lister les périphériques et d'extraire des informations sans pairage
- RedFang : Permet de trouver les périphériques Bluetooth cachés
- Spooftooph : Permet d'usurper le nom, la classe, et l'adresse d'un autre appareil

Plan

1 Sniffer

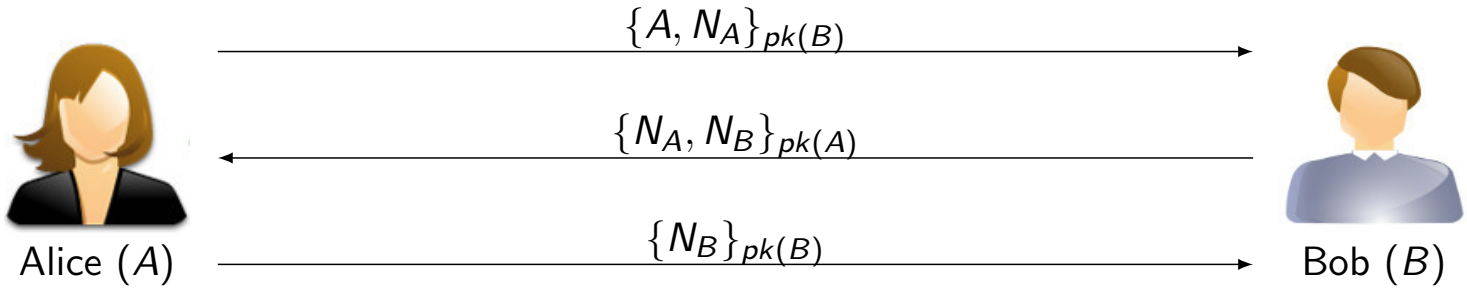
2 Outils de Pentest

3 Tamarin

Tamarin est un outil de vérification de protocole de sécurité. Il permet de modéliser un protocole, ainsi que des propriétés de sécurités, qui seront vérifiées automatiquement par l'outil.

L'outil prend en entrées la modalisation du protocole, ainsi que des propriétés a vérifier.

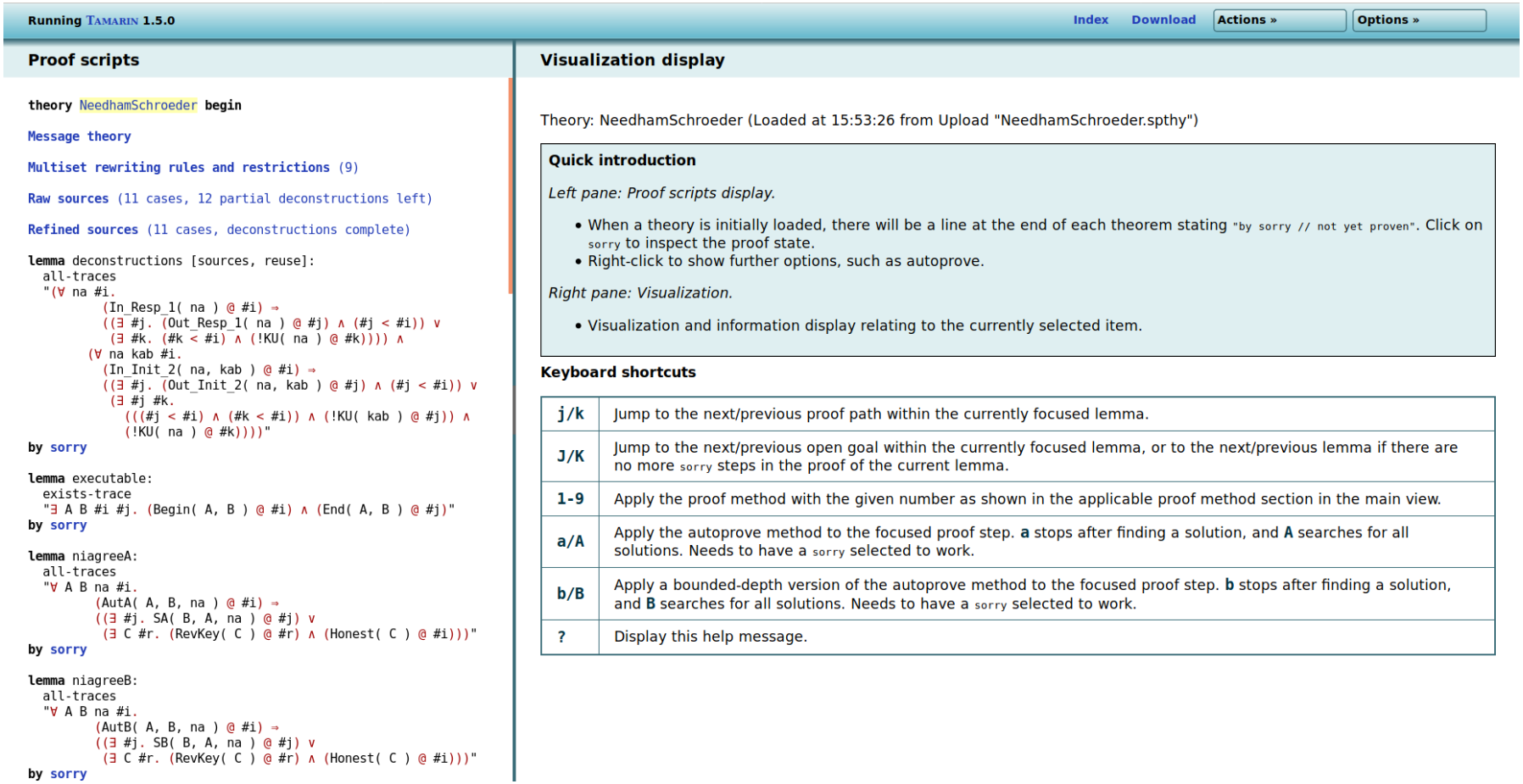
Exemple de protocole : Needham Schroeder



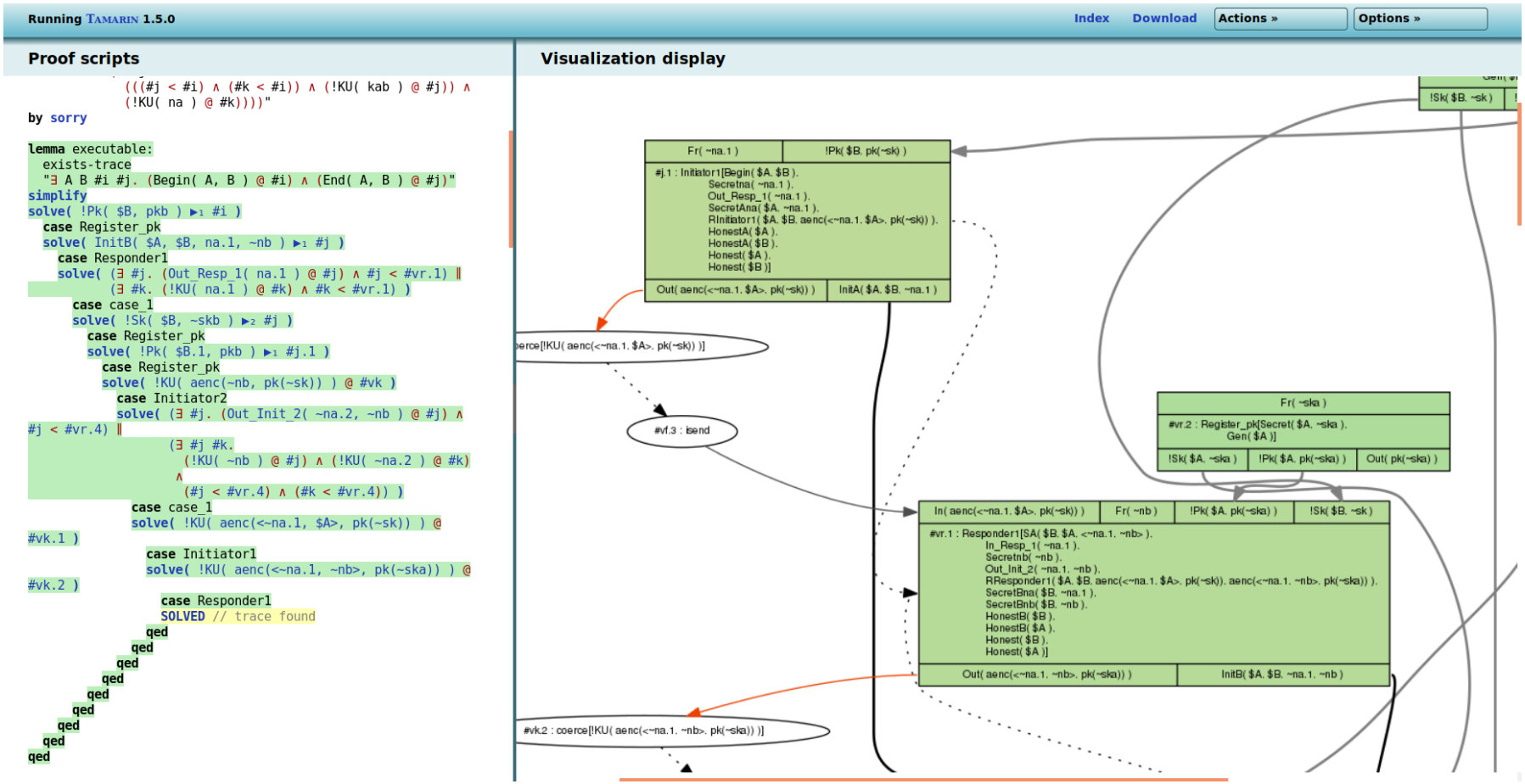
Avec N_A (respectivement N_B) un nombre aléatoire généré par Alice (respectivement Bob), et $pk(A)$ la clé publique d'Alice.

2 types de propriétés possibles :

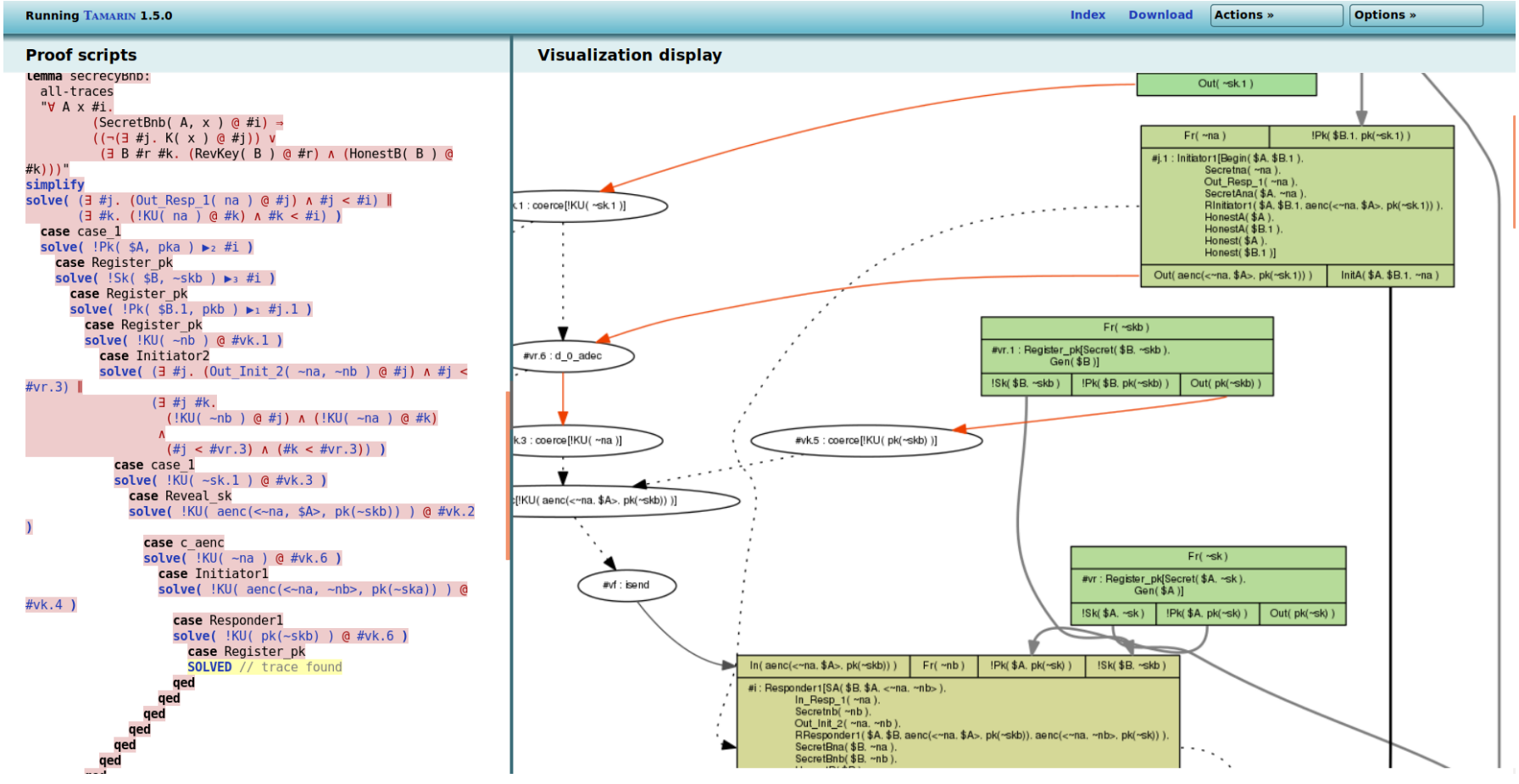
- la propriété doit être vrai dans tous les cas (par exemple, pour toutes exécutions, si Bob reçoit un message d'Alice, alors Alice a bien envoyé le message). Tamarin affiche un contre exemple, s'il en trouve un.
- la propriété doit être vrai dans au moins un cas (par exemple, il existe une exécution où Alice envoie un message, et Bob lui répond). Tamarin affiche un exemple de trace d'exécution correspondant s'il en trouve un.


Running TAMARIN 1.5.0
Index
Download
Actions »
Options »
Proof scripts
theory NeedhamSchroeder begin
Message theory
Multiset rewriting rules and restrictions (9)
Raw sources (11 cases, 12 partial deconstructions left)
Refined sources (11 cases, deconstructions complete)
lemma deconstructions [sources, reuse]:
all-traces
"(∀ na #i.
(In_Resp_1(na) @ #i) ⇒
((∃ #j. (Out_Resp_1(na) @ #j) ∧ (#j < #i)) ∨
(∃ #k. (#k < #i) ∧ (!KU(na) @ #k)))) ∧
(∀ na kab #i.
(In_Init_2(na, kab) @ #i) ⇒
((∃ #j. (Out_Init_2(na, kab) @ #j) ∧ (#j < #i)) ∨
(∃ #j #k.
(((#j < #i) ∧ (#k < #i)) ∧ (!KU(kab) @ #j)) ∧
(!KU(na) @ #k))))"
by sorry
lemma executable:
exists-trace
"∃ A B #i #j. (Begin(A, B) @ #i) ∧ (End(A, B) @ #j)"
by sorry
lemma niagreeA:
all-traces
"∀ A B na #i.
(AutA(A, B, na) @ #i) ⇒
((∃ #j. SA(B, A, na) @ #j) ∨
(∃ C #r. (RevKey(C) @ #r) ∧ (Honest(C) @ #i)))"
by sorry
lemma niagreeB:
all-traces
"∀ A B na #i.
(AutB(A, B, na) @ #i) ⇒
((∃ #j. SB(B, A, na) @ #j) ∨
(∃ C #r. (RevKey(C) @ #r) ∧ (Honest(C) @ #i)))"
by sorry
Visualization display
Theory: NeedhamSchroeder (Loaded at 15:53:26 from Upload "NeedhamSchroeder.spthy")
Quick introduction
Left pane: Proof scripts display.
When a theory is initially loaded, there will be a line at the end of each theorem stating "by sorry // not yet proven". Click on sorry to inspect the proof state.
Right-click to show further options, such as autoprove.
Right pane: Visualization.
Visualization and information display relating to the currently selected item.
Keyboard shortcuts
j/k
Jump to the next/previous proof path within the currently focused lemma.
J/K
Jump to the next/previous open goal within the currently focused lemma, or to the next/previous lemma if there are no more sorry steps in the proof of the current lemma.
1-9
Apply the proof method with the given number as shown in the applicable proof method section in the main view.
a/A
Apply the autoprove method to the focused proof step. a stops after finding a solution, and A searches for all solutions. Needs to have a sorry selected to work.
b/B
Apply a bounded-depth version of the autoprove method to the focused proof step. b stops after finding a solution, and B searches for all solutions. Needs to have a sorry selected to work.
?
Display this help message.

Tamarin


Running TAMARIN 1.5.0
Index
Download
Actions »
Options »
Proof scripts
(((#j < #i) ∧ (#k < #i)) ∧ (!KU(kab) @ #j)) ∧
(!KU(na) @ #k))))"
by sorry
lemma executable:
exists-trace
"∃ A B #i #j. (Begin(A, B) @ #i) ∧ (End(A, B) @ #j)"
simplify
solve(!Pk($B, pkb) ▶₁ #i)
case Register_pk
solve(InitB($A, $B, na.1, ~nb) ▶₁ #j)
case Responder1
solve((∃ #j. (Out_Resp_1(na.1) @ #j) ∧ #j < #vr.1) ∥
(∃ #k. (!KU(na.1) @ #k) ∧ #k < #vr.1))
case case_1
solve(!Sk($B, ~skb) ▶₂ #j)
case Register_pk
solve(!Pk($B.1, pkb) ▶₁ #j.1)
case Register_pk
solve(!KU(aenc(~nb, pk(~sk))) @ #vk)
case Initiator2
solve((∃ #j. (Out_Init_2(~na.2, ~nb) @ #j) ∧
#j < #vr.4) ∥
(∃ #j #k.
(!KU(~nb) @ #j) ∧ (!KU(~na.2) @ #k)
∧
(#j < #vr.4) ∧ (#k < #vr.4)))
case case_1
solve(!KU(aenc(<~na.1, $A>, pk(~sk))) @
#vk.1)
case Initiator1
solve(!KU(aenc(<~na.1, ~nb>, pk(~ska))) @
#vk.2)
case Responder1
SOLVED // trace found
qed
qed
qed
qed
qed
qed
qed
qed
qed
Visualization display
!Sk($B, ~sk)
Fr(~na.1)
!Pk($B, pk(~sk))
#j.1 : Initiator1[Begin($A, $B).
Secretna(~na.1).
Out_Resp_1(~na.1).
SecretAna($A, ~na.1).
RInitiator1($A, $B, aenc(<~na.1, $A>, pk(~sk))).
HonestA($A).
HonestA($B).
Honest($A).
Honest($B)]
Out(aenc(<~na.1, $A>, pk(~sk)))
InitA($A, $B, ~na.1)
erce[!KU(aenc(<~na.1, $A>, pk(~sk)))]
#vf.3 : isend
Fr(~ska)
#vr.2 : Register_pk[Secret($A, ~ska).
Gen($A)]
!Sk($A, ~ska)
!Pk($A, pk(~ska))
Out(pk(~ska))
In(aenc(<~na.1, $A>, pk(~sk)))
Fr(~nb)
!Pk($A, pk(~ska))
!Sk($B, ~sk)
#vr.1 : Responder1[SA($B, $A, <~na.1, ~nb>).
In_Resp_1(~na.1).
Secretnb(~nb).
Out_Init_2(~na.1, ~nb).
RResponder1($A, $B, aenc(<~na.1, $A>, pk(~sk)), aenc(<~na.1, ~nb>, pk(~ska))).
SecretBna($B, ~na.1).
SecretBnb($B, ~nb).
HonestB($B).
HonestB($A).
Honest($B).
Honest($A)]
Out(aenc(<~na.1, ~nb>, pk(~ska)))
InitB($A, $B, ~na.1, ~nb)
#vk.2 : coerce[!KU(aenc(<~na.1, ~nb>, pk(~ska)))]

Tamarin


Running Tamarin 1.5.0
Index
Download
Actions »
Options »
Proof scripts
lemma secrecyBnb:
all-traces
"∀ A x #i.
(SecretBnb(A, x) @ #i) ⇒
((¬(∃ #j. K(x) @ #j)) ∨
(∃ B #r #k. (RevKey(B) @ #r) ∧ (HonestB(B) @
#k)))"
simplify
solve((∃ #j. (Out_Resp_1(na) @ #j) ∧ #j < #i) ∥
(∃ #k. (!KU(na) @ #k) ∧ #k < #i))
case case_1
solve(!Pk($A, pka) ▶2 #i)
case Register_pk
solve(!Sk($B, ~skb) ▶3 #i)
case Register_pk
solve(!Pk($B.1, pkb) ▶1 #j.1)
case Register_pk
solve(!KU(~nb) @ #vk.1)
case Initiator2
solve((∃ #j. (Out_Init_2(~na, ~nb) @ #j) ∧ #j <
#vr.3) ∥
(∃ #j #k.
(!KU(~nb) @ #j) ∧ (!KU(~na) @ #k)
∧
(#j < #vr.3) ∧ (#k < #vr.3)))
case case_1
solve(!KU(~sk.1) @ #vk.3)
case Reveal_sk
solve(!KU(aenc(<~na, $A>, pk(~skb))) @ #vk.2
)
case c_aenc
solve(!KU(~na) @ #vk.6)
case Initiator1
solve(!KU(aenc(<~na, ~nb>, pk(~ska))) @
#vk.4)
case Responder1
solve(!KU(pk(~skb)) @ #vk.6)
case Register_pk
SOLVED // trace found
qed
qed
qed
qed
qed
qed
Visualization display
Out(~sk.1)
Fr(~na)
!Pk($B.1, pk(~sk.1))
#j.1 : Initiator1[Begin($A, $B.1),
Secretna(~na),
Out_Resp_1(~na),
SecretAna($A, ~na),
RInitiator1($A, $B.1, aenc(<~na, $A>, pk(~sk.1))),
HonestA($A),
HonestA($B.1),
Honest($A),
Honest($B.1)]
Out(aenc(<~na, $A>, pk(~sk.1)))
InitA($A, $B.1, ~na)
.1 : coerce[!KU(~sk.1)]
#vr.6 : d_0_adec
Fr(~skb)
#vr.1 : Register_pk[Secret($B, ~skb),
Gen($B)]
!Sk($B, ~skb)
!Pk($B, pk(~skb))
Out(pk(~skb))
k.3 : coerce[!KU(~na)]
#vk.5 : coerce[!KU(pk(~skb))]
[!KU(aenc(<~na, $A>, pk(~skb)))]
#vf : isend
Fr(~sk)
#vr : Register_pk[Secret($A, ~sk),
Gen($A)]
!Sk($A, ~sk)
!Pk($A, pk(~sk))
Out(pk(~sk))
In(aenc(<~na, $A>, pk(~skb)))
Fr(~nb)
!Pk($A, pk(~sk))
!Sk($B, ~skb)
#i : Responder1[SA($B, $A, <~na, ~nb>),
In_Resp_1(~na),
Secretnb(~nb),
Out_Init_2(~na, ~nb),
RResponder1($A, $B, aenc(<~na, $A>, pk(~skb)), aenc(<~na, ~nb>, pk(~sk))),
SecretBna($B, ~na),
SecretBnb($B, ~nb),