

Università degli studi del Molise

Corso di laurea in Sicurezza dei Sistemi Software

Anno accademico 2020/2021

**Laboratorio di Sicurezza (modulo 2)**

**Analisi protocolli**

Autori:

Ciccotelli Andrea – 170210

Simeone Jonathan – 167622

Sommario

[Exercise 1.1 3](file:///D:\Shared\Sicurezza\Model%20checking\Ciccotelli-Simeone\Model%20checking.docx#_Toc69905938)

[Exercise 1.2 5](file:///D:\Shared\Sicurezza\Model%20checking\Ciccotelli-Simeone\Model%20checking.docx#_Toc69905939)

[Exercise 2 7](file:///D:\Shared\Sicurezza\Model%20checking\Ciccotelli-Simeone\Model%20checking.docx#_Toc69905940)

# Esercizio 1

**Descrivere e confrontare le implementazioni e l’output di Proverif per il protocollo di Denning-Sacco e per la sua versione corretta.**

## Denning-Sacco

In questo esempio lo scenario mostra la vulnerabilità del protocollo Denning-Sacco contro attacchi replay.

### File Horn

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Cryptographic protocol verifier \*

\* \*

\* Bruno Blanchet, Vincent Cheval, and Marc Sylvestre \*

\* \*

\* Copyright (C) INRIA, CNRS 2000-2020 \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

(\*

This program is free software; you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation; either version 2 of the License, or

(at your option) any later version.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details (in file LICENSE).

You should have received a copy of the GNU General Public License

along with this program; if not, write to the Free Software

Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*)

pred c/1 elimVar,decompData.

nounif c:x.

(\*Definizione costruttori\*)

fun pk/1. (\*Definizione costruttore chiave pubblica con un argomento (chiave privata) \*)

fun encrypt/2. (\*Definizione costruttore cifratura 2 argomenti (chiave pubblica e messaggio) \*)

fun sign/2. (\*Definizione costruttore firma 2 argomenti (messaggio e chiave privata) \*)

query c:secret[]. (\*Definizione del goal: può il canale (che rappresenta l'avversario) entrare a conoscenza del secret?\*)

reduc

(\* Initialization \*)

(\* Assuzioni di base canale avversario \*)

c:c[];

c:pk(sA[]); (\*Vede la chiave pubblica di A\*)

c:pk(sB[]); (\*Vede la chiave pubblica di B\*)

(\* The attacker \*)

(\*Definizione capacità dell'avversario\*)

c:x & c:encrypt(m,pk(x)) -> c:m; (\*se l'avversario conosce il termine "x" (potrebbe essere qualsiasi cosa, in questo contesto è un'identità intesa come chiave privata) ed è a conoscenza di una cifratura, allora può anche conoscere il messaggio\*)

c:x -> c:pk(x); (\*se l'avversario conosce la chiave segreta di un attore può ottenere la chiave pubblica \*)

c:x & c:y -> c:encrypt(x,y); (\*conosco messaggio, conosco chiave pubblica, posso cifrare\*)

c:sign(x,y) -> c:x; (\*se vedo una firma, vedo il contenuto\*)

c:x & c:y -> c:sign(x,y); (\*Se ho il messaggio e la chiave privata posso firmare\*)

(\* The protocol \*)

(\* A \*)

(\*se sono a conoscenza della chiave pubblica di un attore x, posso andare a cifrare il messaggio per un attore x utilizzando la sua chiave pubblica e firmando con la mia (di A) chiave privata un nonce k associato all'identità dell'attore x\*)

c:pk(x) -> c:encrypt(sign(k[pk(x)], sA[]), pk(x));

(\* B \*)

(\*se ho ricevuto un messaggio da A (la parte sinistra della clausola coincide con quella destra di A), posso inviare un messaggio con il secret ad A utilizzando la sua chiave pubblica\*)

c:encrypt(sign(k, sA[]), pk(sB[])) -> c:encrypt(secret[], pk(k)).

(\* EXPECTPV

RESULT goal reachable: c:secret[]

0.004s (user 0.004s + system 0.000s), max rss 7196K

END \*)

### Considerazioni

Lo scenario nel presente file horn, come si evince anche dai commenti aggiunti evidenziati, è vulnerabile di attacco di tipo replay, in quanto se l’attaccante entra in possesso di una chiave di sessione, conoscendo almeno un cifrato, potrà creare nuovi messaggi cifrati che saranno accettati dalla vittima.

### Output

Initial clauses:

Clause 11: c:c[]

Clause 10: c:pk(sA[])

Clause 9: c:pk(sB[])

Clause 8: c:x & c:encrypt(m,pk(x)) -> c:m

Clause 7: c:x -> c:pk(x)

Clause 6: c:x & c:y -> c:encrypt(x,y)

Clause 5: c:sign(x,y) -> c:x

Clause 4: c:x & c:y -> c:sign(x,y)

Clause 3: c:pk(x) -> c:encrypt(sign(k[pk(x)],sA[]),pk(x))

Clause 2: c:encrypt(sign(k\_1,sA[]),pk(sB[])) -> c:encrypt(secret[],pk(k\_1))

Clause 1: c:new-name[!att = v]

Completing...

goal reachable: c:secret[] Derivation:

Abbreviations:

k\_1 = k[pk(x)]

clause 8 c:secret[]

clause 5 c:k\_1

duplicate c:sign(k\_1,sA[])

clause 2 c:encrypt(secret[],pk(k\_1))

clause 6 c:encrypt(sign(k\_1,sA[]),pk(sB[]))

clause 8 c:sign(k\_1,sA[])

duplicate c:x

clause 3 c:encrypt(sign(k\_1,sA[]),pk(x))

clause 7 c:pk(x)

any c:x

clause 9 c:pk(sB[])

RESULT goal reachable: c:secret[]

## Denning-Sacco-corrected

### File horn

(\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* Cryptographic protocol verifier \*

\* \*

\* Bruno Blanchet, Vincent Cheval, and Marc Sylvestre \*

\* \*

\* Copyright (C) INRIA, CNRS 2000-2020 \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*)

(\*

This program is free software; you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation; either version 2 of the License, or

(at your option) any later version.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details (in file LICENSE).

You should have received a copy of the GNU General Public License

along with this program; if not, write to the Free Software

Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*)

pred c/1 elimVar,decompData.

nounif c:x.

fun pk/1.

fun encrypt/2.

fun sign/2.

query c:secret[].

reduc

(\* Initialization \*)

c:c[];

c:pk(sA[]);

c:pk(sB[]);

(\* The attacker \*)

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

c:x -> c:pk(x);

c:x & c:y -> c:encrypt(x,y);

c:sign(x,y) -> c:x;

c:x & c:y -> c:sign(x,y);

(\* The protocol \*)

(\* A \*)

c:pk(x) -> c:encrypt(sign((pk(sA[]), pk(x), k[pk(x)]), sA[]), pk(x)); (\* aggiunta di un nonce generato da chiave pubblica di origine pk(sA[]) e chiave pubblica di destinazione pk(x) \*)

(\* B \*)

c:encrypt(sign((pk(sA[]), pk(sB[]), k), sA[]), pk(sB[])) -> c:encrypt(secret[], pk(k)).

(\* EXPECTPV

RESULT goal unreachable: c:secret[]

0.004s (user 0.004s + system 0.000s), max rss 7264K

END \*)

### Considerazioni

Rispetto all’istanza del problema non corretta, indicata al punto 1.1, si aggiunge come forma di sicurezza un nonce basato su pk(sA[]) e pk(x), ovvero origine e destinazione del messaggio, che rendono l’attacco replay non più funzionante.

### Output

Initial clauses: Clause 15: c:(v,v\_1,v\_2) -> c:v\_2 Clause 14: c:(v,v\_1,v\_2) -> c:v\_1 Clause 13: c:(v,v\_1,v\_2) -> c:v Clause 12: c:v & c:v\_1 & c:v\_2 -> c:(v,v\_1,v\_2) Clause 11: c:c[] Clause 10: c:pk(sA[]) Clause 9: c:pk(sB[]) Clause 8: c:x & c:encrypt(m,pk(x)) -> c:m Clause 7: c:x -> c:pk(x) Clause 6: c:x & c:y -> c:encrypt(x,y) Clause 5: c:sign(x,y) -> c:x Clause 4: c:x & c:y -> c:sign(x,y) Clause 3: c:pk(x) -> c:encrypt(sign((pk(sA[]),pk(x),k[pk(x)]),sA[]),pk(x)) Clause 2: c:encrypt(sign((pk(sA[]),pk(sB[]),k\_1),sA[]),pk(sB[])) -> c:encrypt(secret[],pk(k\_1)) Clause 1: c:new-name[!att = v] Completing... RESULT goal unreachable: c:secret[]

# Esercizio 2

## Hybrid protocol

### File horn

pred c/1 elimVar,decompData.

nounif c:x.

fun pk/1.

fun aencrypt/2.

fun sencrypt/2.

query c:secret[].

reduc

(\* Initialization \*)

c:c[];

c:k[];

c:pk(sA[]);

c:pk(sB[]); (\*(1) il cancale conosce l'identità di B \*)

(\* The attacker \*)

c:x & c:aencrypt(m,pk(x)) -> c:m;

c:x & c:sencrypt(m,x) -> c:m; (\*(4) l'attaccante riceve secret[] \*)

c:x -> c:pk(x);

c:x & c:y -> c:aencrypt(x,y); (\*(2) l'attaccante usa aencrypt(k, pk(sB[])) per inviare una chiave di cifratura fingendosi A \*)

(\* The protocol \*)

(\* A \*)

c:k[] & c:pk(x) -> c:aencrypt(k[], pk(x));

(\* B \*)

c:aencrypt(k, pk(sB[])) -> c:sencrypt(secret[], k). (\*(3) B riceve la chiave dall'attaccante e risponde con il secret[] cifrato \*)

(\* EXPECTPV

RESULT goal reachable: c:secret[]

\*)

### Considerazioni

Il protoccolo per la condivisione di un

### Output

## Hybrid protocol corretto

### File horn

pred c/1 elimVar,decompData.

nounif c:x.

fun pk/1.

fun aencrypt/2. (\* costruttore funzione di asimmetric encryption \*)

fun sencrypt/2. (\* costruttore funzione di simmetric encryption \*)

fun sign/2.

query c:secret[].

reduc

(\* Initialization \*)

c:c[];

c:k[];

c:pk(sA[]);

c:pk(sB[]);

(\* The attacker \*)

c:x & c:aencrypt(m,pk(x)) -> c:m;

c:x & c:sencrypt(m,x) -> c:m;

c:x -> c:pk(x);

c:x & c:y -> c:aencrypt(x,y);

c:sign(x,y) -> c:x; (\*se vedo una firma, vedo il contenuto\*)

c:x & c:y -> c:sign(x,y); (\*Se ho il messaggio e la chiave privata posso firmare\*)

(\* The protocol \*)

(\* A \*)

c:k[] & c:pk(x) & c:sA[] -> c:aencrypt(sign(k[], sA[]), pk(x)); (\* A invia la chiave cifrata con l’asimmetric encryption e firmandola con la propria identità se e solo se conosce sA[] \*)

(\* B \*)

c:aencrypt(sign(k[], sA[]), pk(sB[])) -> c:sencrypt(secret[], k). (\* Se B riceve il messaggio firmato da A allora invia il secret[] \*)

(\* EXPECTPV

RESULT goal reachable: c:secret[]

\*)

### Considerazioni

Considerando la vulnerabilità nel protocollo standard, ovvero il caso in cui l’attaccante inviasse una chiave scelta da lui immedesimandosi in A, in tal caso, applicando la signature sul messaggio, B è sicuro che sia stato A ad inviare la chiave.

### Output

Initial clauses:

Clause 13: c:c[]

Clause 12: c:k[]

Clause 11: c:pk(sA[])

Clause 10: c:pk(sB[])

Clause 9: c:x & c:pencrypt(m,pk(x)) -> c:m

Clause 8: c:x & c:sencrypt(m,x) -> c:m

Clause 7: c:x -> c:pk(x)

Clause 6: c:x & c:y -> c:pencrypt(x,y)

Clause 5: c:sign(x,y) -> c:x

Clause 4: c:x & c:y -> c:sign(x,y)

Clause 3: c:k[] & c:pk(x) & c:sA[] -> c:pencrypt(sign(k[],sA[]),pk(x))

Clause 2: c:pencrypt(sign(k[],sA[]),pk(sB[])) -> c:sencrypt(secret[],k\_1)

Clause 1: c:new-name[!att = v]

Completing...

RESULT goal unreachable: c:secret[]

# Esercizio 3

## Traccia

Implementare il protocollo formato dai seguenti messaggi

A -> B: {N1}pk(B)

A -> B: {N2}pk(B)

B -> A: {x}pk(A) dove x=N1 oppure x=N2

A -> B: s se A ha ricevuto cifrati attraverso la propria chiave pubblica sia N1 che N2

Per semplicità si consideri il protocollo con soli due partecipanti A e B.

Verificare con Proverif se s risulta segreto. Motivare la risposta

## File horn

pred c/1 elimVar,decompData.

nounif c:x.

fun pk/1.

fun aencrypt/2.

query c:secret[].

reduc

(\* Initialization \*)

c:c[];

c:pk(sA[]);

c:pk(sB[]);

(\* The attacker \*)

c:x & c:aencrypt(m,pk(x)) -> c:m;

c:x -> c:pk(x);

c:x & c:y -> c:aencrypt(x,y);

(\* The protocol \*)

(\* A \*)

c:n1[]; (\* sceglie il nonce 1 \*)

c:n2[]; (\* sceglie il nonce 2 \*)

c:sA[] & c:n1[] & c:pk(sB[]) -> c:aencrypt(n1[], pk(sB[])); (\* A invia il nonce 1 a B cifrato con pk(sB) \*)

c:sA[] & c:n2[] & c:pk(sB[]) -> c:aencrypt(n2[], pk(sB[])); (\* A invia il nonce 2 a B cifrato con pk(sB) \*)

c:sA[] & c:aencrypt(n1[], pk(sA[])) & c:aencrypt(n2[], pk(sA[])) -> c:secret[]; (\* Se A riceve i nonce da lui scelti cifrati con la sua chiave, allora invia il secret nel canale \*)

(\* B \*)

c:aencrypt(x, pk(sB[])) -> c:aencrypt(x, pk(sA[])). (\* Se B riceve un nonce allora lo invia cifrato ad A \*)

(\* EXPECTPV

RESULT goal reachable: c:secret[]

\*)

## Considerazioni

Il protocollo implementato, in seguito all’esecuzione con proverif, reputa la secret[] segreta. Tale segretezza è garantita dalla scelta di A di due nonce che non possono essere scelti da un’attaccante dato che A li riverificherà in seguito. Dato che la comunicazione è cifrata asimmetricamente, un attaccante non è in grado di leggere i nonce. A, infine, invia il secret[] solo se è stato B a rispondere con i nonce verificati da A.

## Output

Initial clauses:

Clause 13: c:c[]

Clause 12: c:pk(sA[])

Clause 11: c:pk(sB[])

Clause 10: c:x & c:aencrypt(m,pk(x)) -> c:m

Clause 9: c:x -> c:pk(x)

Clause 8: c:x & c:y -> c:aencrypt(x,y)

Clause 7: c:n1[]

Clause 6: c:n2[]

Clause 5: c:sA[] & c:n1[] & c:pk(sB[]) -> c:aencrypt(n1[],pk(sB[]))

Clause 4: c:sA[] & c:n2[] & c:pk(sB[]) -> c:aencrypt(n2[],pk(sB[]))

Clause 3: c:sA[] & c:aencrypt(n1[],pk(sA[])) & c:aencrypt(n2[],pk(sA[])) -> c:secret[]

Clause 2: c:aencrypt(x,pk(sB[])) -> c:aencrypt(x,pk(sA[]))

Clause 1: c:new-name[!att = v]

Completing...

RESULT goal unreachable: c:secret[]