Readme

Albanese Daniele, Marco Russodivito e Federico Zappone

2021-05-12 Wed

Contents

1 Esercizio 1

1.1 Denning-sacco

1.1.1 File Horn

```
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(k[pk(x)], sA[]), pk(x));

(* B *)

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

1.1.2 Proverif Output

```
$ proverif denning-sacco.horn
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[]
```

1.2 Denning-sacco corretto

1.1.1 File Horn

```
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));

(* B *)

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

1.2.2 Proverif Output

```
$ proverif denning-sacco-corrected.horn
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[]
```

1.3 Risultati

Dall'output delle due istanze si nota sin da subito che solo nel primo caso l'attaccante riesce ad ottenere *secret[]* e quindi *proverif* rileva un errore. Ciò è possibile dato che nella prima istanza si assume che l'attaccante non arrivi mai a conoscenza di una chiave di sessione. Ciò rende infatti il protocollo vulnerabile al *replay attack*, in quanto, se l'attaccante entra in possesso di una chiave di sessione potrà cifrare un nuovo messaggio che sarà accettato dalla vittima. Nella seconda istanza ciò non accade dato che è presente come ulteriore forma di sicurezza una *signature* del messaggio scambiato durante la cifratura:

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

2 Esercizio 2

2.1 Hybrid Protocol

2.1.1 File Horn

```
pred c/1 elimVar,decompData.
nounif c:x.

fun pk/1.
fun sencrypt/2.
```

```
fun pencrypt/2.

query c:secret[].

reduc
(* Initialization *)
c:c[];
c:pk(sA[]);
c:pk(sB[]);


(* The attacker *)
c:x -> c:pk(x);

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

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


(* The protocol *)
(* A *)
c:pk(x) -> c:pencrypt(secret[], pk(x));

(* B *)
c:pencrypt(secret[], pk(sA[])) -> c:sencrypt(secret[], m[]).
```

2.1.2 Proverif Output

```
$ proverif hybrid-protocol.horn
Initial clauses:
Clause 11: c:c[]
Clause 10: c:pk(sA[])
Clause 9: c:pk(sB[])
Clause 8: c:x -> c:pk(x)
Clause 7: c:x & c:pencrypt(m_1,pk(x)) -> c:m_1
Clause 6: c:x & c:y -> c:pencrypt(x,y)
Clause 5: c:k & c:m_1 -> c:sencrypt(m_1,k)
Clause 4: c:k & c:sencrypt(m_1,k) -> c:m_1
```

```
Clause 3: c:pk(x) -> c:pencrypt(secret[],pk(x))
Clause 2: c:pencrypt(secret[],pk(sA[])) -> c:sencrypt(secret[],m[])
Clause 1: c:new-name[!att = v]
Completing...
goal reachable: c:secret[]

Derivation:
clause 7 c:secret[]
    duplicate c:x
    clause 3 c:pencrypt(secret[],pk(x))
        clause 8 c:pk(x)
            any c:x

RESULT goal reachable: c:secret[]
```

2.2 Hybrid Protocol Corrected

2.2.1 File horn

```
pred c/1 elimVar,decompData.
nounif c:x.

fun pk/1.
fun sencrypt/2.
fun pencrypt/2.
fun sign/2.

query c:secret[]. (* shared key not found *)

reduc
(* Initialization *)
c:c[];
c:pk(sA[]);
c:pk(sB[]);


(* The attacker *)
c:x -> c:pk(x);

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

```
c:x & c:y -> c:pencrypt(x,y);

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

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

(* The protocol *)
(* A *)
c:pk(x) -> c:pencrypt((k[pk(x)], sign(k[pk(x)], sA[])), pk(x));

(* B *)
c:pencrypt((k[pk(sB[])], sign(k[pk(sB[])], sA[])), pk(sB[]))
    -> c:sencrypt(secret[], k[pk(sB[])]).
```

2.2.2 Proverif Output

```
$ proverif hybrid-protocol-corrected.horn
Initial clauses:
Clause 16: c:(v,v_1) -> c:v_1
Clause 15: c:(v,v_1) -> c:v
Clause 14: c:v & c:v_1 -> c:(v,v_1)
Clause 13: c:c[]
Clause 12: c:pk(sA[])
Clause 11: c:pk(sB[])
Clause 10: c:x -> c:pk(x)
Clause 9: c:x & c:pencrypt(m,pk(x)) -> c:m
Clause 8: c:x & c:y -> c:pencrypt(x,y)
Clause 7: c:k_1 & c:m -> c:sencrypt(m,k_1)
Clause 6: c:k_1 & c:sencrypt(m,k_1) -> c:m
Clause 5: c:sign(x,y) -> c:x
Clause 4: c:x & c:y -> c:sign(x,y)
Clause 3: c:pk(x) -> c:pencrypt((k[pk(x)],sign(k[pk(x)],sA[])),pk(x))
Clause 2: c:pencrypt((k[pk(sB[])],sign(k[pk(sB[])],sA[])),pk(sB[]))
              -> c:sencrypt(secret[],k[pk(sB[])])
Clause 1: c:new-name[!att = v]
Completing...
RESULT goal unreachable: c:secret[]
```

2.2.3 Risultati

Il protocollo ibrido proposto nel secondo esercizio non risulta essere sicuro in quanto non vi è alcuna forma di integrità durante lo scambio dei messaggi, in particolare della chiave condivisa. Ciò porta l'attaccante a poter eseguire un attacco al protocollo. Per ovviare a questo problema si è inserita una *signature* della chiave scambiata che ne certifica l'integrità.

2.3 False alarm protocol

2.3.1 File horn

```
pred c/1 elimVar,decompData.
nounif c:x.

fun pk/1.
fun pencrypt/2.

query c:secret[].

reduc
(* Initialization *)

c:c[];
c:pk(sA[]);
c:pk(sB[]);

(* The attacker *)
c:x -> c:pk(x);

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

(* The protocol *)
(* A *)
c:pk(x) -> c:pencrypt(n1[pk(x)], pk(x));
c:pk(x) -> c:pencrypt(n2[pk(x)], pk(x));

c:pencrypt(n1[pk(sA[])], pk(sA[])) &
    c:pencrypt(n2[pk(sA[])], pk(sA[])) -> c:secret[];
```

```
(* B *)
c:pencrypt(n1[pk(sA[])], pk(sB[])) -> c:pencrypt(n1[pk(sA[])], pk(sA[]));
c:pencrypt(n2[pk(sA[])], pk(sB[])) -> c:pencrypt(n2[pk(sA[])], pk(sA[])).
```

2.3.2 Proverif Output

```
$ proverif false-error-protocol.horn
Initial clauses:
Clause 12: c:c[]
Clause 11: c:pk(sA[])
Clause 10: c:pk(sB[])
Clause 9: c:x -> c:pk(x)
Clause 8: c:x & c:pencrypt(m,pk(x)) -> c:m
Clause 7: c:x & c:m -> c:pencrypt(x,m)
Clause 6: c:pk(x) -> c:pencrypt(n1[pk(x)],pk(x))
Clause 5: c:pk(x) -> c:pencrypt(n2[pk(x)],pk(x))
Clause 4: c:pencrypt(n1[pk(sA[])],pk(sA[])) &
              c:pencrypt(n2[pk(sA[])],pk(sA[])) -> c:secret[]
Clause 3: c:pencrypt(n1[pk(sA[])],pk(sB[])) -> c:pencrypt(n1[pk(sA[])],pk(sA[]))
Clause 2: c:pencrypt(n2[pk(sA[])],pk(sB[])) -> c:pencrypt(n2[pk(sA[])],pk(sA[]))
Clause 1: c:new-name[!att = v]
Completing...
goal reachable: c:secret[]

Derivation:
Abbreviations:
n1_1 = n1[pk(sA[])]
n2_1 = n2[pk(sA[])]
clause 4 c:secret[]
    clause 6 c:pencrypt(n1_1,pk(sA[]))
        duplicate c:pk(sA[])
    clause 5 c:pencrypt(n2_1,pk(sA[]))
        clause 11 c:pk(sA[])

RESULT goal reachable: c:secret[]
```

2.3.3 Risultati

Il protocollo proposto mostra un semplice ma efficace esempio di come un sistema di *over-approximation* come *proverif* può portare a dei *false alarms*. In particolare il protocollo non è in alcun modo utilizzabile in quanto esso arriva a conclusione solo nel caso in cui *Alice* abbia ricevuto da *Bob* entrambi i *Nonces* cifrati. Ciò però non è possibile in quanto il protocollo definisce l'invio di un unico *Nonce* da parte di *Bob*, pertanto *Alice* non potrà mai riceverli entrambi. Il *false-alarm* è dato dal fatto che in *proverif* non può essere definito l'invio di un messaggio una sola volta, infatti, il *secret[]*, ovvero il caso in cui *Alice* abbia ricevuto entrambi i *Nonces* è raggiungibile in quanto la *over-approximation* aggiunge path inesistenti nella definizione reale del problema come appunto l'invio ripetuto di messaggi unici.