Advanced security - quiz 3

Philip Korsholm

October 2020

Exercise 1 - Message deduction

1.1

$$\dfrac{\dfrac{}{senc(s,\langle n_A,n_B\rangle)} \quad \dfrac{\dfrac{aenc(n_A,pk(sk_B)) \quad sk_B}{n_A} \quad \dfrac{\dfrac{\dfrac{}{senc(aenc(n_B,pk(sk_A)),n_A)} \quad \dfrac{aenc(n_A,pk(sk_B) \quad sk_B}{n_A}}{aenc(n_B,pk(sk_A))} \quad \dfrac{}{sk_A}}{n_B}}{\langle n_A,n_B\rangle}}{s}$$

1.2

An inference system is local if the derived terms is directly written in the terms it's derived from.

Exercise 2 - Equational theories

2.1

The same inference tree from can be extended with method calls to construct and destruct the terms. I have created pairs from terms without functions since none was supplied, thus i assumed it to be implied, otherwise one would have to add another fraction when creating pairs that apply a "pair" function.

$$\dfrac{\dfrac{\dfrac{}{senc(s,\langle n_A,n_B\rangle)} \quad \dfrac{\dfrac{\dfrac{aenc(n_A,pk(sk_B)) \quad sk_B}{adec(aenc(n_A,pk(sk_B)),sk_B)}}{n_A} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{senc(aenc(n_B,pk(sk_A)),n_A)} \quad \dfrac{\dfrac{aenc(n_A,pk(sk_B) \quad sk_B}{adec(aenc(n_A,pk(sk_B)),sk_B)}}{n_A}}{sdec(senc(aenc(n_B,pk(sk_A)),n_A),n_A)}}{aenc(n_B,pk(sk_A))} \quad sk_A}{adec(aenc(n_B,pk(sk_A)),sk_A)}}{n_B}}{\langle n_A,n_B\rangle)}}{sdec(senc(s,\langle n_A,n_B\rangle),\langle n_A,n_B\rangle)}}{s}$$

2.2

Exercise 3 - Static equivalence

Which pairs of frames are statically equivalent

frame 1

The frames are not statically equivalent because if we pick $M = x$ $N = h(y)$ these are equivalent in ϕ_2 but not in ϕ_1 since in phi_1 $x = h(senc(a, k))$ but $y = senc(b, k)$ and when hashing y they are not equal.

Other frames

The other frames are statically equivalent.

There is one caveat, in frame 3, even though it's not restricted we don't see c. If we are allowed to use c they are not statically equivalent due to the pair: $M = aenc((z, c), y)$ and $N = x$

Exercise 4 - Observational equivalence

4.1 - Show that A is not observationally equivalent to B

I will show that A and B are not observationally equivalent by constructing a context that provides an example where $C[A]\mathcal{R}[B]$ is not true.

Construct $C[\_]$ as such:

$$C[\_] = out(c, 1).out(c, 0).in(c, pk(k)).in(c, y).$$

$$if\ y = aenc((1, 0), pk(l))\ then\ out(c, 1)|\_$$

This will output on channel c when encountering B but not A, and thus they are not observationally equivalent.

4.2 - Diff-equivalence

Are A and B comparable using diff-equivalence?

Why Yes, because they only differ in one term and thus we can input it into proverif.

Proverif Model

```
channel c.

type skey.
type pkey.

free s : bitstring.
free k : skey.
```

```
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall x:bitstring, y:skey;
            adec(aenc(x, pk(y)), y) = x.

process
      in(c, x:bitstring);
      in(c, y:bitstring);
      out(c, pk(k));
      if x <> y then
      out(c, aenc(diff[(x,y,s), (x,y)], pk(k)))
```

5 - Proverif

I have described the protocol below in proverif, and written three queries that describe the desired security properties, the queries are labelled with comments.

```
channel c.
free d: channel[private].

type key.
type pkey.
type skey.
type host.

fun enc(bitstring, key) : bitstring.
reduc forall x: bitstring, y:key; dec(enc(x, y), y) = x.

fun h(bitstring): bitstring.

fun pk(skey): pkey.

fun aenc(bitstring, pkey): bitstring.
reduc forall x:bitstring, y:skey; adec(aenc(x, pk(y)), y) = x.

fun pkey2B(pkey): bitstring[typeConverter].
fun B2pkey(bitstring): pkey[typeConverter].

fun nonce2Key(bitstring): key[typeConverter].

fun bits2Host(bitstring): host[typeConverter].
fun host2bits(host): bitstring[typeConverter].

(*
```

```
A -> S {{A}pkB, B }Kas
S -> B {{A}pkB } Kbs
B -> A {Nb}pkA
A -> B {m}Nb
B -> A h(m)
*)

event startA(host, host, bitstring).
event endB(host, host, bitstring).

event acceptM(host, host, bitstring).
event endA(host, host, bitstring).

(*      Authentication      *)
query A: host, B:host, m:bitstring;
        inj-event (endB(A, B, m)) ==>
                inj-event (startA(A, B, m)).

(*      Integrity           *)
query A: host, B:host, m:bitstring;
        inj-event (endA(A, B, m) ) ==>
                inj-event (acceptM(A, B, m)).

(*      Confidentiality     *)
query attacker(new m).

let pA(A: host, B:host, Kas: key) =
        new skA: skey;
        new m: bitstring;
        event startA(A, B, m);
        out(c, (A, pk(skA)));
        in(c, (=B, pkB: pkey));
        out(c, enc((aenc(host2bits(A), pkB), pkB), Kas));
        in(c, x : bitstring);
        let (nB: bitstring) = adec(x, skA) in
        out(c, enc(m, nonce2Key(nB)));
        in(c, y:bitstring);
        if y = h(m) then event endA(A, B, m).

let pB(B: host, Kbs: key) =
        new skB: skey;
        out(c, (B, pk(skB)));
        in(c, x: bitstring);
        let A:host  = bits2Host(adec(dec(x, Kbs), skB)) in
        in(c, (=A, pkA: pkey));
        new nB: bitstring;
```

```
        out(c, aenc(nB, pkA));
        in(c, y: bitstring);
        let m = dec(y, nonce2Key(nB)) in
        event acceptM(A, B, m);
        out(c, h(m));
        event endB(A, B, m).

let pS() =
        in(d, Kas: key);
        in(d, Kbs: key);
        in(c, x :bitstring);
        let (xpkb: bitstring, pkB:pkey) = dec(x, Kas) in
        out(c, enc(xpkb, Kbs)).

process
        (! new X: host; new Kxs: key ; !out(d, (X, Kxs)) |
        (! in(d, (A: host, Kas: key)) ; in(d, (B:host, Kbs:key)) ; pA(A, B, Kas)
        (! in(d, (B: host, Kbs: key)) ; pB(B, Kbs))) |
        (! pS())
```