Examination of the module MPRI 2-30
Cryptographic protocols: formal and computational proofs

(Solution)
March 2, 2016

2 CryptoVerif

2.1 Exercise 1

(1) $X \parallel Y = f_{sk}^{-1}(c)$, $r = H(X) \oplus Y$, $m \parallel 0 = X \oplus G(r)$. One can check that the last $k_1$ bits of $X \oplus G(r)$ are 0.

(2) declare the type large

(3) type $Dr$ has size $k_0$, type $Dow$ has size $n - k_0$, type $Dm$ has size $n - k_0 - k_1$.

let hashoracleG(hkg: hashkey) = !iG <= qG in(chG1, x:Dr); out(chG2, G(hkg,x)).

let hashoracleH(hkh: hashkey) = !iH <= qH in(chH1, x:Dow); out(chH2, H(hkh,x)).

let processT(hkg: hashkey, hkh: hashkey, pk: pkey) =
    in(c1, (m1: Dm, m2: Dm));
    new b1: bool;
    (* The next line is equivalent to an "if" that will not be expanded. This is necessary for the system to succeed in proving the protocol. *)
    let menc = test(b1, m1, m2) in
    new r: Dr;
    let s = xorDow(concatm(menc, zero), G(hkg,r)) in
    let t = xorDr(r, H(hkh,s)) in
    out(c2, f(pk, concat(s,t))).

process
    in(start, ());
    new hkh: hashkey;
    new hkg: hashkey;
    new r: seed;
    let pk = pkgen(r) in
    let sk = skgen(r) in (* Not necessary for IND-CPA *)
    out(c0, pk);
    (hashoracleG(hkg) | hashoracleH(hkh) | processT(hkg, hkh, pk))

(4) Random oracle of $H$ and $G$ can be applied directly. The property of $\oplus$ cannot (even after syntactic transformation) because $r$ is used in $G(r)$. One-wayness cannot (even after syntactic transformation) because the argument of $f$ is not random.
Applying the random oracle assumption replaces $G(r)$ with a fresh random value $r'$, which allows applying the assumption of $\oplus$ twice. (Actually, in the hash oracles, we need to introduce events using Shoup lemma to avoid leaking $r$.) After that, the argument of $f$ is random, so one-wayness can be applied (after replacing $pk$ with its value and removing the assignment to $sk$).

(5) We need to add a decryption oracle:

```plaintext
let processD(hkg: hashkey, hkh: hashkey, sk: skey) =
  !qD
  in(c3, c: D);
  find such that defined(cT) && c = cT then yield else
  let concat(s,t) = invf(sk, c) in
  let r = xorDr(t, H(hkh, s)) in
  let mz = xorDow(s, G(hkg, r)) in
  let concatm(m, =zero) = mz in
  out(c4, m).

processD(hkg, hkh, sk) is added to final parallel composition, and the last line of processT is replaced with

let cT: D = f(pk, concat(s,t)) in
out(c2, cT).

so that $cT$ is defined.

2.2 Exercise 2

(1) let processA(pkA: spkey, skA: sskey, pkB: pkey) =
  in(c1, pkX: pkey);
  new k:key;
  (* The signature and encryption are probabilistic, CryptoVerif
     adds the random number generation internally, but you may
     also write it explicitly, e.g.:
     new r: sseed;
     sign(k, skA, r) *)
  let payload = concat(pkA, k, sign(k, skA)) in
  out(c2, penc(payload, pkX));
  (* Test for secrecy *)
  in(c5, ());
  if pkX = pkB then
  let k': key = k in
  yield.

let processB(skB: skey, pkA: spkey) =
  in(c3, m: bitstring);
  let pinjbot(concat(pkY, kB, s)) = pdec(m, skB) in
  if check(kB, pkY, s) then
  (* Test for secrecy *)
  if pkY = pkA then
  let k'': key = kB in
  yield.
```
process
    in(start, ());
    new rkA: skeyseed;
    let pkA = spkgen(rkA) in
    let skA = sskgen(rkA) in
    new rkB: pkeyseed;
    let pkB = pkgen(rkB) in
    let skB = skgen(rkB) in
    out(c7, (pkA, pkB));
    (! NA processA(pkA, skA, pkB)) |
    (! NB processB(skB, pkA))

(2) The key $k$ that $A$ has is secret, but the key that $B$ has is not secret. The attack is
the well-known attack against the Denning-Sacco protocol (similar to the one against
Needham-Schroeder public key):

$$A \rightarrow I : \mathcal{E}_{pk_A}(pk_A, k, \mathcal{S}_{sk_A}(k))$$

$$I(A) \rightarrow B : \mathcal{E}_{pk_B}(pk_A, k, \mathcal{S}_{sk_A}(k))$$

$A$ starts a session with the attacker $I$, which forwards the message to $B$ after reencrypting
it under $pk_B$. The fix consists in adding the public key of $B$ in the signature.