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

(Solution)

March 2, 2016

2 CryptoVerif

2.1 Exercise 1

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

(2) declare the type \texttt{large}

(3) type \texttt{Dr} has size \( k_0 \), type \texttt{Dow} has size \( n - k_0 \), type \texttt{Dm} has size \( n - k_0 - k_1 \).

\[ \begin{align*}
\text{let} & \text{ hashoracleG(hkg: hashkey)} = \text{foreach } i \leq qG \text{ do } \text{OG(x:Dr)} := \text{return}(G(hkg,x)). \\
\text{let} & \text{ hashoracleH(hkh: hashkey)} = \text{foreach } i \leq qH \text{ do } \text{OH(x:Dow)} := \text{return}(H(hkh,x)). \\
\text{let} & \text{ processT(hkg: hashkey, hkh: hashkey, pk: pkey)} = \\
& \quad \text{OT(m1: Dm, m2: Dm)} := \\
& \quad b_1 \leftarrow \text{R bool}; \\
& \quad (* \text{ The next line is equivalent to an "if" that will not be expanded. This is necessary for the system to succeed in proving the protocol. } *) \\
& \quad \text{let} \text{ menc = test(b1, m1, m2) in } \\
& \quad r \leftarrow \text{R Dr}; \\
& \quad \text{let} \text{ s = xorDow(concatm(menc, zero), G(hkg,r)) in } \\
& \quad \text{let} \text{ t = xorDr(r, H(hkh,s)) in } \\
& \quad \text{return}(f(pk, concat(s,t))). \\
\end{align*} \]

process

\[ \begin{align*}
\text{Ostart()} := \\
\text{hkh \leftarrow \text{R hashkey};} \\
\text{hkg \leftarrow \text{R hashkey};} \\
\text{r \leftarrow \text{R seed};} \\
\text{let} \text{ pk = pkgen(r) in } \\
\text{let} \text{ sk = skgen(r) in } \\
\text{return}(pk); \\
\text{(run hashoracleG(hkg) \mid run hashoracleH(hkh) \mid run processT(hkg, hkh, pk))}
\end{align*} \]

(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) =
  foreach iD <= qD do
    OD(c: D) :=
      find suchthat 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
      return(m).
run processD(hkg, hkh, sk) is added to the final parallel composition, and the last line of processT is replaced with

  let cT: D = f(pk, concat(s,t)) in
  return(cT).
```

so that $cT$ is defined.

### 2.2 Exercise 2

(1) let processA(pkA: spkey, skA: sskey, pkB: pkey) =

```plaintext
OA1(pkX: pkey) :=
  k <-R key;
  (* The signature and encryption are probabilistic, CryptoVerif
     adds the random number generation internally, but you may
     also write it explicitly, e.g.:
     r <-R sseed;
     sign(k, skA, r) *)
  let payload = concat(pkA, k, sign(k, skA)) in
  return(penc(payload, pkX));
(* Test for secrecy *)
OA2() :=
  if pkX = pkB then
    let k': key = k in
    yield.
```

let processB(skB: skey, pkA: spkey) =

```plaintext
OB(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

Ostart() :=
    rkA <- R skeyseed;
    let pkA = spkgen(rkA) in
    let skA = sshgen(rkA) in
    rkB <- R pkeyseed;
    let pkB = pkgen(rkB) in
    let skB = skgen(rkB) in
    return(pkA, pkB);

    ((foreach iA <= NA do run processA(pkA, skA, pkB)) |
     (foreach iB <= NB do run 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):

\[
A \to I : E_{pkI}(pk_A, k, S_{skA}(k))
\]
\[
I(A) \to B : E_{pkB}(pk_A, k, S_{skA}(k))
\]

$A$ 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.