Conception et analyse de protocoles cryptographiques

Brocoli

Louis Béziaud
louis.beziaud@ens-rennes.fr

Jérémy Thibault
jeremy.thibault@ens-rennes.fr

Master SIF / 2017 – 2018

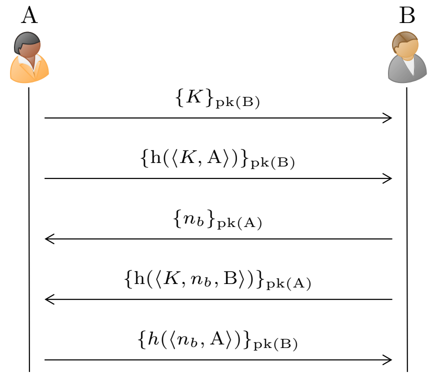
A
B
$\{K\}_{\mathrm{pk(B)}}$
$\{\mathrm{h}(\langle K, \mathrm{A}\rangle)\}_{\mathrm{pk(B)}}$
$\{n_b\}_{\mathrm{pk(A)}}$
$\{\mathrm{h}(\langle K, n_b, \mathrm{B}\rangle)\}_{\mathrm{pk(A)}}$
$\{h(\langle n_b, \mathrm{A}\rangle)\}_{\mathrm{pk(B)}}$

Figure 1: Brocoli Protocol

Initial knowledge We assume that agents A and B know the public key $\mathrm{pk}(C)$ corresponding to agent C for each agent C. Agent A also know the secret data K.

Created values Nonce n_b is generated by B.

Description

- Alice (i.e., A) sends her secret K, encrypted by an asymmetric encryption algorithm using Bob's public key (denoted pk(B)). Note that only Bob (i.e., B) knows the private key corresponding to the public key pk(B). Alice then sends a hash (denoted $\mathrm{h}(\cdot)$) of her secret K and her name, also encrypted with pk(B).
- Bob uses his private key to decrypt the two messages sent by Alice and learns K. Bob generates a random number n_b and sends it to Alice, encrypted with Alice's public key pk(A). Bob also sends to Alice an encrypted hash of K, n_b and his name B.
- Alice receives and decrypts the two messages from Bob using her private key. Alice finally sends a hash of Bob's random number n_b with its name, encrypted with $\mathrm{pk}(B)$.

Security properties At the end of the protocol,

- if B think he has received K from A, then A has sent him K;
- if A has sent K to B, then B has received K;
- secret data K is only known by A and B.

Cost

$$
\begin{aligned}
&(1) & 1+1+1 &= 3\\
&(2) & 1+1+5+1+1 &= 9\\
&(3) & 1+1+1 &= 3\\
&(4) & 1+1+5+1+1+1 &= 10\\
&(5) & 1+1+5+1+1 &= 9\\
\hline
& & &= 34
\end{aligned}
$$

Appendix

Below are formalizations of our protocol provided for completeness. Listing 1 corresponds to the Scyther implementation, and Listing 2 to ProVerif. Comments indicate verification's result and message cost.

Listing 1: Brocoli in Scyther

```
hashfunction hash;

usertype MySecret;

protocol brocoli(I, R) { // 34
  role I {
    var nr: Nonce;
    fresh K: MySecret;

    send_1a (I, R, {K}pk(R)); // 3
    send_1b (I, R, {hash(K, I)}pk(R)); // 9

    recv_2a (R, I, {nr}pk(I));
    recv_2b (R, I, {hash(K, nr, R)}pk(I));

    send_3 (I, R, {hash(nr, I)}pk(R)); // 9

        claim(I, Secret, K); // No attacks
        claim(I, Secret, nr); // No attacks
        claim(I, Alive); // No attacks
        claim(I, Weakagree); // No attacks
        claim(I, Niagree); // No attacks
        claim(I, Nisynch); // No attacks
  }

  role R {
    fresh nr: Nonce;
    var K: MySecret;

    recv_1a (I, R, {K}pk(R));
    recv_1b (I, R, {hash(K, I)}pk(R));

    send_2a (R, I, {nr}pk(I)); // 3
    send_2b (R, I, {hash(K, nr, R)}pk(I)); // 10

    recv_3 (I, R, {hash(nr, I)}pk(R));

        claim(R, Secret, K); // No attacks
        claim(R, Secret, nr); // No attacks
        claim(R, Alive); // No attacks
        claim(R, Weakagree); // No attacks
        claim(R, Niagree); // No attacks
        claim(R, Nisynch); // No attacks
  }
}
```

Listing 2: Brocoli in ProVerif

```
free c: channel.

type skey.
type pkey.
fun pk(skey): pkey.
fun aenc(bitstring, pkey): bitstring.
reduc forall m: bitstring, k: skey; adec(aenc(m, pk(k)), k) = m.

fun p2b(pkey): bitstring [typeConverter].

fun h2(bitstring, bitstring): bitstring.
fun h3(bitstring, bitstring, bitstring): bitstring.

free k: bitstring [private].

event endA(pkey, bitstring).
event endB(pkey, bitstring).

query a: pkey, b: pkey, s: bitstring;
event(endB(a, s)) ==> event(endA(b, s)).
(* RESULT event(endB(a_903,s_905)) ==> event(endA(b_904,s_905)) is true. *)

query a: pkey, b: pkey, s: bitstring;
event(endB(a, s)) ==> s = k.
(* RESULT event(endB(a,s_455)) ==> s_455 = k[] is true. *)

query attacker(k).
(* RESULT not attacker(k[]) is true. *)

let roleA(pkA: pkey, skA: skey, pkB: pkey) =
  out(c, aenc(k, pkB)); (* 3 *)
  out(c, aenc(h2(k, p2b(pkA)), pkB)); (* 9 *)
  in(c, x1: bitstring);
  let nb = adec(x1, skA) in
    in(c, x2: bitstring);
    let (=h3(k, nb, p2b(pkB))) = adec(x2, skA) in
        event endA(pkB, k);
            out(c, aenc(h2(nb, p2b(pkA)), pkB)). (* 9 *)

let roleB(pkB: pkey, skB: skey, pkA: pkey) =
  new nb: bitstring;
  in(c, x1: bitstring);
  let s = adec(x1, skB) in
    in(c, x2: bitstring);
    let (=h2(s, p2b(pkA))) = adec(x2, skB) in
            out(c, aenc(nb, pkA)); (* 3 *)
            out(c, aenc(h3(s, nb, p2b(pkB)), pkA)); (* 10 *)
            in(c, x3: bitstring);
            let (=h2(nb, p2b(pkA))) = adec(x3, skB) in
                event endB(pkA, k).

process
  new skA: skey;
  new skB: skey;
  let pkA = pk(skA) in out(c, pkA);
  let pkB = pk(skB) in out(c, pkB);
  ((!roleA(pkA, skA, pkB)) | (!roleB(pkB, skB, pkA)))
```