University of Padova
Automated Analysis of Security Protocols exercises report

Alberto Lazari - 2089120

June 2023 - A.Y. 2022-2023

Contents

1 Message deduction

1.1 Inference system

$\mathcal{I}_{DY}$ provides the following inference rules:

$$(\text{senc})\ \frac{x \qquad y}{\mathsf{senc}(x,y)} \qquad (\text{aenc})\ \frac{x \qquad y}{\mathsf{enc}(x,y)}$$

$$(\text{pair})\ \frac{x \qquad y}{\langle x,y\rangle} \qquad (pair_1)\ \frac{\langle x,y\rangle}{x} \qquad (pair_2)\ \frac{\langle x,y\rangle}{y}$$

$$(\text{sdec})\ \frac{\mathsf{senc}(x,y) \qquad y}{x} \qquad (\text{adec})\ \frac{\mathsf{aenc}(x,\mathsf{pk}(y)) \qquad y}{x}$$

1.2 Proof tree solving

Let $S = sk_A, sk_B, \mathsf{aenc}(n_A, \mathsf{pk}(sk_B)), \mathsf{senc}(\mathsf{aenc}(n_B, \mathsf{pk}(sk_A)), n_A), \mathsf{senc}(s, \langle n_A, n_B\rangle)$.
$S \vdash_{\mathcal{I}_{DY}} s$ is solved by the following proof tree:

$$\dfrac{\mathsf{senc}(s,\langle n_A,n_B\rangle) \qquad \dfrac{\dfrac{\vdots}{n_A} \qquad \dfrac{\dfrac{\mathsf{senc}(\mathsf{aenc}(n_B,\mathsf{pk}(sk_A)),n_A) \qquad \dfrac{\vdots}{n_A}}{\mathsf{aenc}(n_B,\mathsf{pk}(sk_A))}\,\text{sdec} \qquad sk_A}{n_B}\,\text{adec}}{\langle n_A,n_B\rangle}\,\text{pair}}{s}\,\text{sdec}$$

Given $S \vdash_{\mathcal{I}_{DY}} n_A$, which is solved by the following proof tree:

$$\dfrac{\mathsf{aenc}(n_A,\mathsf{pk}(sk_B)) \qquad sk_B}{n_A}\,\text{adec}$$

1.3 Local theory

$\mathcal{I}_{DY}$ is a *local theory*, because, given S and t, every label in the proof tree that solves $S \vdash_{\mathcal{I}_{DY}} t$ is a subterm of $S \cup \{t\}$, i.e. $\mathcal{I}_{DY}$ rules do not introduce new terms (other than t) that are not already present in S.

2 Deduction under equational theories

$S \vdash_{E_{enc}} s$ is solved by the following tree:

$$\dfrac{\dfrac{\mathrm{senc}(s, \langle n_A, n_B\rangle) \qquad \dfrac{\dfrac{\vdots}{n_A} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\mathsf{senc}(\mathsf{aenc}(n_B, \mathsf{pk}(sk_A)), n_A) \qquad \dfrac{\vdots}{n_A}}{\mathsf{sdec}(\mathsf{senc}(\mathsf{aenc}(n_B, \mathsf{pk}(sk_A)), n_A), n_A)}}{\mathsf{aenc}(n_B, \mathsf{pk}(sk_A))}{=_{E_{enc}}} \qquad sk_A}{\mathsf{adec}(\mathsf{aenc}(n_B, \mathsf{pk}(sk_A)), sk_A)}}{n_B}{=_{E_{enc}}}}{\langle n_A, n_B\rangle}}{\mathsf{sdec}(\mathsf{senc}(s, \langle n_A, n_B\rangle), \langle n_A, n_B\rangle)}}{s}{=_{E_{enc}}}$$

Given $S \vdash_{E_{enc}} n_A$, which is solved by the following proof tree:

$$\dfrac{\dfrac{\mathsf{aenc}(n_A, \mathsf{pk}(sk_B)) \qquad sk_B}{\mathsf{adec}(\mathsf{aenc}(n_A, \mathsf{pk}(sk_B)), sk_B)}}{n_A}{=_{E_{enc}}}$$

Note that the rules without a label are not the ones defined in $\mathcal{I}_{DY}$ (though they are the same). They are derived from the functions, generated from the equivalence steps. The only exception is:

$$\dfrac{a \qquad b}{\langle a, b\rangle}$$

but $\langle a, b\rangle$ could be defined as a 2-ary function $\mathsf{pair}(a, b)$.

3 Static equivalence of frames

1. Let $M = x, N = \mathsf{h}(y)$.
   $(x = \mathsf{h}(\mathsf{senc}(a, k)) \neq_{E_{enc}} \mathsf{h}(y))_{\varphi_1}$ and $(x =_{E_{enc}} \mathsf{h}(y))_{\varphi_2} \implies \varphi_1 \not\sim \varphi_2$

2. Let $M = x, N = \mathsf{aenc}(\langle z, c\rangle, y)$.
   $(x =_{E_{enc}} \mathsf{aenc}(\langle z, c\rangle, y))_{\varphi_1}$ and $(x = \mathsf{aenc}(\langle \mathsf{senc}(b, k), c\rangle, y) \neq_{E_{enc}} \mathsf{aenc}(\langle z, c\rangle, y))\varphi_1)_{\varphi_2}$
   $\implies \varphi_1 \not\sim \varphi_2$

4 Observational equivalence

Let $C[\_]$ be the context

$$C[\_] = \mathsf{out}(c, 0); \mathsf{out}(c, 1); \mathsf{in}(c, pk); \mathsf{in}(c, m); \mathsf{if}\ m = \mathsf{aenc}(\langle 0, 1\rangle, pk)\ \mathsf{then}\ \mathsf{out}(b, 1) \parallel \_$$

$C[B] \Downarrow b$, while $C[A]$ will never emit on channel $b \implies B \not\approx A \implies A \not\approx B$

5 Secrecy and authentication

The following code models the protocol in ProVerif:

```
channel c.

(* A and B identities *)
const idA : bitstring.
const idB : bitstring.

(* Symmetric encryption *)
fun senc(bitstring, bitstring) : bitstring.
reduc forall m : bitstring, k : bitstring;
      sdec(senc(m, k), k) = m.

(* Asymmetric encryption *)
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.

(* Pair *)
fun pair(bitstring, bitstring) : bitstring.
reduc forall a : bitstring, b : bitstring;
      fst(pair(a, b)) = a.
reduc forall a : bitstring, b : bitstring;
      snd(pair(a, b)) = b.

(* Hash function *)
fun h(bitstring) : bitstring.


(* Events *)
event authentication(bitstring, bitstring).
event integrity(bitstring).


(* Queries *)
(* Authentication *)
query m : bitstring, nB : bitstring;
      event(authentication(m, nB)).
```

```
(* Integrity *)
query m : bitstring;
      event(integrity(m)).
(* Confidentiality *)
query attacker(new m).


(* Processes *)
let A(sk : skey, pkB : pkey, kS : bitstring) =
    let req = pair(aenc(idA, pkB), idB) in
    out(c, senc(req, kS));
    in(c, x : bitstring);
    let nB = adec(x, sk) in
    new m : bitstring;
    out(c, senc(m, nB));
    in(c, hash : bitstring);
    if hash = h(m) then
    event integrity(m).

let S(kA : bitstring, kB : bitstring) =
    in(c, x : bitstring);
    let req = sdec(x, kA) in
    if snd(req) = idB then
    out(c, senc(fst(req), kB)).

let B(sk : skey, pkA : pkey, kS : bitstring) =
    in(c, x : bitstring);
    let enc_idA = sdec(x, kS) in
    if adec(enc_idA, sk) = idA then
    new nB : bitstring;
    out(c, aenc(nB, pkA));
    in(c, y : bitstring);
    let m = sdec(y, nB) in
    event authentication(m, nB);
    out(c, h(m)).

process
    new skA : skey; new skB : skey;
    new kAS : bitstring; new kBS : bitstring;
    ( !A(skA, pk(skB), kAS)
    | !S(kAS, kBS)
    | !B(skB, pk(skA), kBS) )
```

5.1 Authentication

Authentication is proved by the first query: the event occurs only when B recieves m from A.
B can be certain that m comes from A, because it decrypts it from a message encrypted with N_b, that could only be decrypted by A, because B encrypted asymmetrically with pk_A, thus requiring sk_A to decrypt.
The event is reached beacuse ProVerif outputs the following statement:

```
Query not event(authentication(m_2,nb_2)) is false.
```

which means that the event is reachable.

5.2 Integrity

Integrity is proved by the second query: the event occurs only if A recieves back from B the hash of m. A checks wether the hash sent from B and the real $\mathsf{h}(m)$ are the same.
The event is reached beacuse ProVerif outputs the following statement:

```
Query not event(integrity(m_2)) is false.
```

which means that the event is reachable.

5.3 Confidentiality

Confidentiality is proved by the last query: it checks wether an external attacker can deduce the message m, but since it cannot only A and B know the message.
The last query is true because ProVerif outputs the following statement:

```
Query not attacker(m[x = v,!1 = v_1]) is true.
```