

## Solution of Exercise Sheet 2

Out: Thu, Nov 5, 2009

Due: Tue, Nov 17, 2009, 10am

A request: If possible, please use white paper (without lines or grid) for writing your solution. I find that the grid distracts from the text and makes the solution harder to read. Also, please leave a large margin for corrections.

### Problem 1: Automating the protocol analysis

In exercise sheet 1, problem 2, we have analyzed a simple example protocol by hand. This time, your task is to make an automated analysis:

- Write an input file for the SPASS theorem prover to automatically prove the security of the protocol from exercise sheet 1, problem 2.
- To check that your SPASS input makes sense, try out what happens when the hash function is insecure, i.e., when we additionally have the deduction rule  $\frac{S \vdash \text{hash}(x)}{S \vdash x}$ .

The necessary changes to the SPASS input should be minimal.

**Hint:** You find a very short tutorial for SPASS on <http://www.spass-prover.org/tutorial.html>. You do not need to install SPASS on your computer, there is a web-interface that allows to execute SPASS input files: <http://www.spass-prover.org/WebSPASS>.

**Hint:** Do not try to model the deduction relation in its full generality (i.e., as a relation between sets of messages and messages). This will be unnecessarily difficult because you would have to model sets. Instead, define a predicate `knows1` such that `knows1(x)` holds iff  $S_1 \vdash x$ , and similarly `knows2`. Furthermore, it is OK to do a slight overapproximation if necessary.

**Hint:** If you do not manage to write a SPASS input file for the whole protocol analysis, you might try to break it down and first let SPASS show that  $S_2 \not\vdash N_{secret}$  assuming that  $K' = K_2$ . This would not be the full solution, but it will give you part of the points.

### Solution.

```
begin_problem(fmc09_homework2).
list_of_descriptions.
name({*FMC 2009, homework 2, solution*}).
author({*Dominique Unruh*}).
```

```

status(unsatisfiable).
description({* *}).
end_of_list.

list_of_symbols.
% We define function symbols for all constructors, and for the
% protocol nonces that appear in the protocol. We use a generic
% function advnonce to model adversary-nonces. That is,
% adversary-nonces are of the form advnonce(whatever).
functions[(enc,2),(hash,1),(advnonce,1),(Nsecret,0),(K1,0),(K2,0)].

% We model two predicates for the adversary knowledge.
% knows1(x) means that S1|-x
% knows2(x) means that S2|-x
predicates[ (knows1,1), (knows2,1) ].
end_of_list.

list_of_formulae(axioms).
% Deduction rules for S1|-...
formula(forall([x],knows1(advnonce(x)))).
formula(forall([x,y],implies(and(knows1(x),knows1(y)),knows1(enc(x,y))))).
formula(forall([x],knows1(hash(x)))).
formula(forall([x,y],implies(and(knows1(enc(x,y)),knows1(x)),knows1(y)))).

% Deduction rules for S2|-...
% These are the same as for S1|-...
formula(forall([x],knows2(advnonce(x)))).
formula(forall([x,y],implies(and(knows2(x),knows2(y)),knows2(enc(x,y))))).
formula(forall([x],knows2(hash(x)))).
formula(forall([x,y],implies(and(knows2(enc(x,y)),knows2(x)),knows2(y)))).

% Definition of the set S1
% Instead of defining S1, we give axioms S1|-x for all x\in S1
formula(knows1(hash(K1))).
formula(knows1(enc(K1,K2))).

% Definition of the set S2
% Instead of defining S2, we give axioms S2|-x for all x\in S2
formula(knows2(hash(K1))).
formula(knows2(enc(K1,K2))).
% It is difficult in SPASS to say "let K' be a term such that
% knows1(enc(K1,K'))". Instead, we just say "for all K' with
% knows1(enc(K1,K')), let enc(K',Nsecret)\in S2".

```

```

% (This constitutes an overapproximation.)
formula(forall([k],implies(knows1(enc(K1,k)),knows2(enc(k,Nsecret))))).

% Assuming a bad hash function, we need to add the rules
% S_i|-hash(x) ==> S_i|-x for i=1,2.
% Thus, for the second part of the problem, uncomment the following
% two lines:

%formula(forall([x],implies(knows1(hash(x)),knows1(x)))).
%formula(forall([x],implies(knows2(hash(x)),knows2(x)))).
end_of_list.

list_of_formulae(conjectures).
  formula(knows2(Nsecret)).
end_of_list.

end_problem.

```

When running SPASS on this input, it outputs (besides a lot of other text): SPASS beiseite: Completion found. This means that the conjecture `knows2(Nsecret)` cannot be shown from the axioms. I.e.,  $S_2 \not\vdash N_{secret}$ . .noituloc

## Problem 2: Computationally unsound protocols

When proving computational soundness of passive secrecy properties, various conditions on the passive secrecy properties have to hold. In the lecture, we have already seen that we need to exclude key-cycles (Definition 24 in the lecture notes). During the proof, we will meet additional conditions. For example, freshness of randomness requires that any two different ciphertexts use different randomness. In the following two problems, you are required to design a computational implementation such that certain protocols are not computationally sound.

In both problems, you will need the definition of IND-CPA security which we give here for reference:

**Definition 1 (IND-CPA)** *An encryption scheme  $(K, E, D)$  consisting of probabilistic polynomial-time algorithms  $K$  (key-generation),  $E$  (encryption), and  $D$  (decryption) is IND-CPA secure if for any polynomial-time oracle machine  $\text{Adv}$ , there is a negligible function  $\mu$  such that for all  $k$  we have  $|\text{Pr}_0(k) - \text{Pr}_1(k)| \leq \mu(k)$ . Here*

$$\text{Pr}_i(k) := \Pr[b = 1 : \text{key} \leftarrow K(1^k), b \leftarrow \text{Adv}^{\mathcal{E}_i(\text{key}, \cdot)}(1^k)].$$

and  $\mathcal{E}_0(\text{key}, \cdot)$  is an oracle that returns  $E(1^k, \text{key}, m)$  when queried with a message  $m$ , and  $\mathcal{E}_1(\text{key}, \cdot)$  is an oracle that returns  $E(1^k, \text{key}, 0^{|m|})$  when queried with a message  $m$ .

(The relevant difference to IND-OT-CPA as defined in Definition 1 in the lecture notes is that in the present definition, the adversary may request the encryptions of many messages, while in the IND-OT-CPA definition, he may request only one.)

We say a computational implementation  $A$  is IND-CPA secure if  $(K, E, D)$  is IND-CPA secure where (i) all  $A_N$  for  $N \in \mathbf{N}$  are identical, (ii)  $K(1^k) := A_N(1^k)$  for some  $N \in \mathbf{N}$ , (iii)  $E(1^k, key, m) := A_{enc}(1^k, key, m, A_N(1^k))$  for some  $N \in \mathbf{N}$ . (There is no need to specify  $D$  because it does not occur in Definition 1.)

- (a) **Key-cycles.** Let  $\wp := (N_{secret}, (enc(N_{secret}, N_{secret}, R)))$  with  $N_{secret}, R \in \mathbf{N}^P$ . (The most trivial key-cycle.) Design a computational implementation  $A$  that is IND-CPA secure such that  $\wp$  does not hold computationally. (You do not need to prove that  $A$  is IND-CPA secure.)

**Hint:** Assume an encryption scheme  $(K', E', D')$  that is IND-CPA secure and modify it to get another, still IND-CPA secure scheme  $(K, E, D)$  which you use for your computational implementation.

**Solution.** Let  $(K', E', D')$  be an IND-CPA secure encryption scheme. Assume that  $D'$  is deterministic, and that  $K'(1^k)$  outputs a uniformly chosen  $k$ -bit string. Assume that  $E'$  only takes  $k$  bit of randomness (where  $k$  is the security parameter). Write  $E'(1^k, key, m; r)$  for the result of running  $E'(1^k, key, m)$  with randomness  $r$ . Let  $\parallel$  denote concatenation of bitstrings.

We define the following computational implementation  $A$ :

$$A_N(1^k) := \text{uniformly chosen } k\text{-bit string (for all nonces } N)$$

$$A_{enc}(1^k, key, m, r) := \begin{cases} 1 \parallel m & \text{if } m = key \\ 0 \parallel E'(1^k, key, m; r) & \text{otherwise} \end{cases}$$

$$A_{dec}(1^k, key, c) := \begin{cases} m & \text{if } c = 1 \parallel m \text{ for some } m \\ D'(1^k, key, c') & \text{if } c = 0 \parallel c' \text{ for some } c' \end{cases}$$

It is easy to see that this computational implementation is IND-CPA secure if  $(K', E', D')$  is.

With this computational implementation, and  $\wp$  as defined in the problem statement, the adversary  $\text{Adv}$  in the definition of  $\text{Succ}(\wp, \text{Adv}, k)$  (Definition 21 in the lecture notes) is invoked as  $\text{Adv}(1^k, A_{enc}(1^k, r_{N_{secret}}, r_{N_{secret}}, r_R))$  with  $r_{N_{secret}}, r_R$  begin uniformly chosen  $k$ -bit strings. By definition of  $A_{enc}$  we have  $A_{enc}(1^k, r_{N_{secret}}, r_{N_{secret}}, r_R) = 1 \parallel r_{N_{secret}}$ . Thus, if we define  $\text{Adv}(1^k, 1 \parallel x) := x$ , we have  $\text{Succ}(\wp, \text{Adv}, k) = 1$ . Note that  $\text{Adv}$  is polynomial-time. Thus  $\wp$  does not hold computationally. **.noituflo2**

- (b) **Fresh randomness.** Let  $\wp_1 := (N_{secret}, (enc(K_1, N_{secret}, R), enc(K_2, N_{secret}, R)))$  with  $N_{secret}, K_1, K_2, R \in \mathbf{N}^P$ . Design a computational implementation  $A$  that is

IND-CPA secure such that  $\wp_1$  does not hold computationally. (You do not need to prove that  $A$  is IND-CPA secure.)

Alternatively, if you prefer, you can do the same for  $\wp_2 := (N_{secret}, (enc(K_1, N_{secret}, R), enc(K_1, N, R), N))$  with  $N_{secret}, K_1, N, R \in \mathbf{N}^P$  instead of  $\wp_1$ .

**Hint:** Same hint as in part (a). Think of one-time-pads (which are, however, not IND-CPA secure). Furthermore, if you happen to need randomness that is longer than a nonce, use a pseudorandom generator.

**Solution.** First, we solve the problem for  $\wp_1$ . Let  $(K', E', D')$  and  $A_N$  be as in the preceding solution.

Let  $G : \{0, 1\}^k \rightarrow \{0, 1\}^{2k}$  be a pseudorandom generator. Let  $G_0(r)$  and  $G_1(r)$  denote the first and last  $k$  bits of  $G(r)$ . Let  $start(x)$  denote the first  $k$  bits of  $x$  (or  $x$  padded with 0's if  $|x| < k$ ). Let  $end(x)$  denote all but the first  $k$  bits of  $x$ . Let  $bit(x)$  denote the first bit of  $x$ . Let  $\parallel$  denote concatenation of bitstrings. Let  $\oplus$  denote bitwise XOR.

Let

$$A_{enc}(1^k, key, m, r) := \begin{cases} G_1(r) \parallel E'(1^k, key, m; G_0(r)) & \text{if } bit(key) = 0 \\ (start(m) \oplus G_1(r)) \parallel E'(1^k, key, m; G_0(r)) & \text{if } bit(key) = 1 \end{cases}$$

$$A_{dec}(1^k, key, c) := D'(1^k, key, end(c))$$

We omit the proof that  $A$  is IND-CPA secure. The intuitive reason is that  $G_1(r)$  is either used as a one-time-pad when encrypting  $start(m)$ , or included in the plaintext, but never both. Furthermore,  $G_1(r)$  is never used twice for the same  $r$ .

Consider uniformly chosen  $k$ -bit values  $r_{N_{secret}}, r_{K_1}, r_{K_2}, r_R$ . With probability  $\frac{1}{4}$ ,  $bit(K_1) = 0$  and  $bit(K_2) = 1$ . Let

$$(b_1, b_2) := (A_{enc}(r_{K_1}, r_{N_{secret}}, r_R), A_{enc}(r_{K_2}, r_{N_{secret}}, r_R)).$$

Then

$$\begin{aligned} start(b_1) \oplus start(b_2) &= start(G_1(r_R) \parallel \dots) \oplus start((start(r_{N_{secret}}) \oplus G_1(r_R)) \parallel \dots) \\ &= G_1(r_R) \oplus (r_{N_{secret}} \oplus G_1(r_R)) = r_{N_{secret}}. \end{aligned}$$

Thus, for the adversary  $\text{Adv}(1^k, b_1, b_2) := start(b_1) \oplus start(b_2)$ , we get  $\text{Succ}(\wp_1, \text{Adv}, k) = \frac{1}{4}$ . Thus  $\wp_1$  does not hold computationally.

Now, we solve the problem for  $\wp_2$ . Let  $(K', E', D')$  and  $A_N$  be as in the preceding solution. Let  $start, end, \parallel, \oplus$  be defined as in the solution for  $\wp_2$ .

Let

$$A_{enc}(1^k, key, m, r) := (start(m) \oplus G_1(r)) \parallel E'(1^k, key, m, G_0(r))$$

$$A_{dec}(1^k, key, c) := D'(1^k, key, end(c))$$

We omit the proof that the computational implementation  $A$  is IND-CPA secure.

Consider  $k$ -bit values  $r_{N_{secret}}, r_{K_1}, r_N, r_R$ . Let

$$(b_1, b_2, b_3) := (A_{enc}(r_{K_1}, r_{N_{secret}}, r_R), A_{enc}(r_{K_1}, r_N, r_R), r_N).$$

Then

$$\begin{aligned} & start(b_1) \oplus start(b_2) \oplus b_3 \\ &= start((start(r_{N_{secret}}) \oplus G_1(r_R)) \parallel \dots) \oplus start((start(r_N) \oplus G_1(r_R)) \parallel \dots) \oplus r_N \\ &= (r_{N_{secret}} \oplus G_1(r_R)) \oplus (r_N \oplus G_1(r_R)) \oplus r_N = N_{secret}. \end{aligned}$$

Thus, for the adversary  $\text{Adv}(1^k, b_1, b_2, b_3) := start(b_1) \oplus start(b_2) \oplus b_3$ , we have  $\text{Succ}(\varphi_2, \text{Adv}, k) = 1$ . Thus  $\varphi_2$  does not hold computationally. **.noitulos**