Cryptology I (spring 2015) Dominique Unruh

Exercise Sheet 5

Out: 2015-05-06 Due: 2015-05-20

Problem 1: ElGamal FDH

Bob studied the RSA-FDH construction. He notices that RSA-FDH essentially does the following: To sign a message m, it decrypts $H(m)$ using textbook RSA, and to check a signature σ, it encrypts σ and compares the result with $H(m)$.

This lead him to the following idea: Instead of textbook RSA, he uses ElGamal in the construction of FDH, because ElGamal is more secure (it is EF-CMA secure).

Why is the resulting scheme "ElGamal-FDH" bad?

Problem 2: Random oracle model

Write down the definition of IND-CPA security in the random oracle model (for symmetric encryption schemes).

Problem 3: Security proof in the ROM [Bonus problem]

This is a bonus problem.

Fix a hash function $H : \{0,1\}^* \to \{0,1\}^\eta$. We define the following block cipher with message and key space $\{0,1\}^\eta$.:

- **Encryption:** To encrypt $m \in \{0,1\}^\eta$ under key k, choose a random $r \in \{0,1\}^\eta$ and return the ciphertext $c := (r, m \oplus H(k\|r))$.
- **Decryption:** To decrypt $c = (r, c')$ with key k, compute and return $m := H(k\|r) \oplus c'$.

Below is a proof that this encryption scheme is IND-CPA secure in the random oracle model. Fill in the gaps. (The length of the gaps is unrelated to the length of the text to be inserted.)

Proof. In the first game, we just restate the game from the IND-CPA security definition (in the random oracle model).

Game 1. [1] ◇

To show that the encryption scheme is IND-CPA secure, we need to show that

$$|\Pr[b = b' : \mathit{Game}\ 1] - \tfrac{1}{2}| \text{ is negligible} \tag{1}$$

As a first step, we replace the random oracle.

Game 2. Like Game 1, except that we define the random oracle H differently: [2] $\diamond$

We have $\Pr[b = b' : \mathit{Game}\ 1] = \Pr[b = b' : \mathit{Game}\ 2]$.

One can see that the adversary cannot guess the key k (where k is the key used for encryption in Game 2), more precisely, the following happens with negligible probability: "The adversary invokes $H(x)$ with $x = k\|r'$ for some r'." (We omit the proof of this fact.)

Let r_0 denote the value r that is chosen during the execution of $c \leftarrow E^H(k, m_b)$ in Game 2. Consider the following event: "Besides the query $H(k\|r_0)$ performed by $c \leftarrow E^H(k, m_b)$, there is another query $H(x)$ with $x = k\|r_0$ (performed by the adversary or by the oracle $E^H(k, \cdot)$." This event occurs with negligible probability. Namely, the adversary make such $H(x)$ queries with negligible probability because [3], and the oracle $E^H(k, m_b)$ makes such $H(x)$ queries with negligible probability because [4].

Thus, the response of the $H(k\|r_0)$-query performed by $c \leftarrow E^H(k, m_b)$ is a random value that is used nowhere else (except with negligible probability). Thus, we can replace that value by some fresh random value.

Game 3. Like Game 2, except that we replace $c \leftarrow E^H(k, m_b)$ by $r_0 \xleftarrow{\$} \{0,1\}^\eta$, $h^* \xleftarrow{\$} \{0,1\}^\eta$, $c \leftarrow (r_0, m_b \oplus h^*)$. $\diamond$

We have that $|\Pr[b = b' : \mathit{Game}\ 2] - \Pr[b = b' : \mathit{Game}\ 3]|$ is negligible.

To get rid of m_b in Game 3, we use the fact that h^* is chosen uniformly at random and XORed on m_b. That is, we can replace $m_b \oplus h^*$ by [5].

Game 4. Like Game 3, except that we replace $c \leftarrow (r_0, m_b \oplus h^*)$ by [6]. $\diamond$

Notice that b is not used in Game 4, thus we have that $\Pr[b = b' : \mathit{Game}\ 4] =$ [7].
Combining the equations we have gathered, (1) follows. $\square$

Problem 4: Symbolic cryptography

In the following, we will investigate protocols that use both symmetric and asymmetric encryption (in the symbolic model). We model messages as follows: A, B are names of participants (known to everyone). pk_A, pk_B are the public keys of A, B (known to everyone). A ciphertext encrypted with public key pk_X is written $penc(pk_X, m)$. We use symbols K_i for symmetric keys, and we write $senc(K, m)$ for the symmetric encryption of m using K. We use R_i for random values (nonces). We write $\hat{R}_i$ and $\hat{K}_i$ for additional random values/keys belonging to the adversary. We write (x, y) for pairs.

(a) Write down the deduction rules that model the capabilities of the adversary (i.e., those that do not depend on the protocol). You may assume that no party is corrupted (that is, all public keys belong to honest participants and the adversary cannot decrypt messages encrypted with one of these public keys).

(b) Consider the following protocol:

- Alice and Bob share a random value R_1. Alice has a secret value R_0 that he wishes to send to Bob.

- Alice picks a symmetric key K_1 and sends $penc(pk_B, (K_1, R_1))$ to Bob.
- After checking that the message from Alice contains the right value R_1, Bob picks a key K_2 and sends $senc(K_1, K_2)$ to Alice.
- Alice sends $senc(K_2, R_0)$ to Bob.

Write down the deduction rules corresponding to this protocol. (Assume that only a single instance each of Alice and of Bob runs, and that the network between Alice and Bob is insecure in the sense that the adversary can intercept and replace messages.)

(c) Write down a grammar M that describes the terms the adversary can deduce. That is, if $\vdash t$, the t should match M. (Here $\vdash$ is the deduction relation corresponding to the deduction rules you wrote down in (a) and (b).)

For each rule, explain (shortly) why all terms in the conclusion of the rule match M if all terms in the preconditions of the rule match M.

(d) Show that the protocol is secure, i.e., that $\nvdash R_0$.

(e) **[Bonus question]** Consider the following variation of the protocol from (b):

- Alice has a secret value R_0 that he wishes to send to Bob.
- Alice picks a symmetric key K_1 and sends $penc(pk_B, K_1)$ to Bob.
- Bob picks a key K_2 and sends $senc(K_1, K_2)$ to Alice.
- Alice sends $senc(K_2, R_0)$ to Bob.

(The only difference is that now the shared random value R_1 is not included in the first message from Alice.)

Write down the deduction rules corresponding to this protocol (analogous to (b)).

(f) **[Bonus question]** Show that for the deduction rules from (a) and (e), we have $\vdash R_0$ (i.e., the protocol is insecure). Write down the derivation tree of $\vdash R_0$.[1]

Hint: If you do not see the solution, you can try to produce a grammar of terms as in (c). This will help you to see why R_0 can be deduced.

(g) **[Bonus question]** Explain why the attack found in (f) does not work if Bob answers to each message only once. (I.e., when getting several messages of the form $penc(pk_B, K)$ for some K, he responds only to the first one.) Explain why it is difficult to prove the security in this case using the methods we discussed.

[1] The package `mathpartir` is useful for writing deduction rules and derivation trees in LaTeX. `http://cristal.inria.fr/~remy/latex/`

Problem 5: Implementing a protocol verifier [Bonus problem]

In this exercise, we will implement a very simple protocol analyzer.

Consider the deduction rules we formulate for the Needham-SchrÃ¶der protocol (not NSL). We simplify them slightly by saying that the adversary can only produce three different randomnesses. (I.e., the RAND rule holds only for $i = 1, 2, 3$.) That is, we have the following rules:

$$\frac{\vdash enc_C(x)}{\vdash x}\ \textsc{Dec} \qquad \frac{\vdash x \qquad P \in \{A, B, C\}}{\vdash enc_P(x)}\ \textsc{Enc}$$

$$\frac{}{\vdash A \qquad \vdash B \qquad \vdash C}\ \textsc{Name} \qquad \frac{\vdash x \qquad \vdash y}{\vdash (x, y)}\ \textsc{Pair}$$

$$\frac{\vdash (x, y)}{\vdash x \qquad \vdash y}\ \textsc{Split} \qquad \frac{i = 1, 2, 3}{\vdash \hat{R}_i}\ \textsc{Rand}$$

$$\frac{}{\vdash enc_C((A, R_1))}\ \textsc{Msg1} \qquad \frac{\vdash enc_B((A, y))}{\vdash enc_A((y, R_2))}\ \textsc{Msg2-NS}$$

$$\frac{\vdash enc_A((R_1, x))}{\vdash enc_C(x)}\ \textsc{Msg3-NS}$$

Your task is to write a program that computes the set of all t with $\vdash t$, and then to check whether $enc_B(R_2)$ is in that set.

Of course, that set is infinite. For simplicity, we want to avoid having to represent an infinite set (as we did in the lecture). For this reason, we introduce another (arbitrary) restriction on the adversary. Namely, the adversary cannot derive any term of size greater than 4.[2] Formally, that means that all the rules above implicitly have the additional condition $size(t) \leq 4$ where t is the term in the conclusion. Practically, this means that if a term t can be derived from terms already in the set, you only have to add t to the set if $size(t) \leq 4$.

Since there are only finitely many possible terms of size ≤ 4 (in fact 41280), eventually the set will be saturated.

You are free to chose a programming language of your choice. In fact, a functional programming language is probably best suited for dealing with terms, but I will give the solution in Python anyway.

Below are some fragments of the final code in Python. Note that I fixed a particular encoding of terms as nested tuples, where each tuple starts with a string that indicates what kind of term we have (`enc`, `pair`, `rand`, etc.)

```
#!/usr/bin/python
```

[2]Here we define the size as follows: $size(P) = size(R_i) = size(\hat{R}_i) = 1$ for $P = A, B, C$, $i \in \mathbb{N}$, and $size((t, u)) = size(t) + size(u)$, and $size(enc_P(t)) = size(t) + 1$.

```
# Will contain all terms that are deducible by the adversary
known_terms = set()

# Contains all terms that have been added to known_terms but not yet combined
# with other terms
fresh_terms = set()

# The term about which we want to know whether it is deducible
target_term = ("enc","B",("rand",2))

# Maximum size. Terms above this size will not be considered
max_size = 4



# Results of my experiments (I also analyzed the NSL protocol):

# nsl | max_size | #known_terms | derivable
# -------------------------------------
# no  |    2     |      60      | no
# yes |    2     |      60      | no
# no  |    3     |    1768      | yes
# yes |    3     |    1267      | no
# no  |    4     |   41280      | yes
# yes |    4     |   26404      | no
# -------------------------------------

# Encoding of messages:
# enc(pk_A,t) --> ("enc",A,t)  (where A is just a string)
# (t,u) --> ("pair",t,u)
# A --> ("name",i) (i is an integer)
# R_i --> ("rand",i) (i is an integer)
# \Hat R_i --> ("rand_adv",i) (i is an integer)

# Computes the size of t
# (Lots of asserts to catch encoding bugs early on)
def size(t):
    assert isinstance(t,tuple)
    if t[0]=="enc":
        assert len(t)==3
        assert isinstance(t[1],str)
        return size(t[2])+1
```

```
    if t[0]=="pair":
        assert len(t)==3
        return size(t[1])+size(t[2])
    if t[0]=="rand":
        assert len(t)==2
        assert isinstance(t[1],int)
        return 1
    if t[0]=="name":
        assert len(t)==2
        assert isinstance(t[1],str)
        return 1
    if t[0]=="rand_adv":
        assert len(t)==2
        assert isinstance(t[1],int)
        return 1
    raise RuntimeError("unexpected term",t)

# Useful for debugging
def term_to_string(t):
    if t[0]=="enc":
        return "enc({},{})".format(t[1],term_to_string(t[2]))
    if t[0]=="pair":
        return "({},{})".format(term_to_string(t[1]),term_to_string(t[2]))
    if t[0]=="rand":
        return "R{}".format(t[1])
    if t[0]=="name":
        return t[1]
    if t[0]=="rand_adv":
        return "R^{}".format(t[1])
    raise RuntimeError("unexpected term",t)



[...lots more...]


print "Number of derivable terms:",len(known_terms)

# Uncomment to see all terms (useful for debugging)
#for t in known_terms:
#    print ">",term_to_string(t)

derivable = target_term in known_terms
```

```
print "Checking if {} can be derived: {}".format(term_to_string(target_term),derivable)
```