Cryptography

Academic Year 2025-2026

Homework 3

Francesco Testa, ID 1179083

December 6, 2025

Exercise 1.
The scheme is modified as follows:

- $\mathsf{Gen}(1^n)$: remains the same;
- $\mathsf{Enc}((N,e),m)$: the amount of randomness used is fixed to $\Theta(\lg n)$ bits.

$\mathrm{Enc}((N,e),m)$

1 : $r \leftarrow \{0,1\}^{\Theta(\lg n)}$
2 : $f \leftarrow 0^{|N|-|r|-\ell(n)-1}$
3 : $c \leftarrow (f\|r\|m)^e \bmod N$
4 : **return** c

- $\mathsf{Dec}((N,d),c)$: remains the same.

We now prove that the modified scheme is not CPA-secure. Like in classic Padded RSA, the goal of the Adversary is to guess the value of r used in the encryption. Thus, in the classical version, the Adversary has to brute-force all the possible values of r. This requires an asyntotic time of:

$$O(2^{|N|-\ell(n)-1})$$

Using the theorem seen in class[1], if we assume $\ell(n) = O(\lg n) \approx c \cdot \lg n$:

$$\begin{aligned} O\left(2^{|N|-\ell(n)-1}\right) &= O\left(2^{|N|-(c\cdot\lg(n))-1}\right) \\ &= O\left(\frac{2^{|N|}}{2^{c\cdot\lg(n)}} \cdot \frac{1}{2}\right) \\ &= O\left(\frac{2^{|N|}}{n^c}\right) \\ &= O\left(2^{|N|}\right) \end{aligned}$$

So this takes exponential time in $|N|$.

[1] If the RSA Assumption holds with respect to GenRSA and if $\ell(n) = O(\lg n)$, then Padded RSA is secure with respect to passive attacks.

Let's consider now the modified version, where the randomness used is $\Theta(\lg n) \approx c \cdot \lg n$ bits. Using the definition above, the brute-force attack to find r requires an asymptotic time of:

$$\begin{aligned} O(2^{|r|}) &= O(2^{c \cdot \lg(n)}) \\ &= O(2^{\lg(n^c)}) \\ &= O(n^c) \end{aligned}$$

This is polynomial in n and therefore efficient. Consequently, the Adversary can easily brute-force all the possible values of r and therefore break the CPA-security of the scheme.

Exercise 2.
Let's build a MSR theory corresponding to the Handshake Protocol, namely:

$$\begin{aligned} &A \rightarrow B : pk_A \\ &B \rightarrow A : aEnc(sign((pk_B, k), sk_B), pk_A) \\ &A \rightarrow B : sEnc(s, k) \end{aligned}$$

To begin, let's define the **sorts**:

`key`		`skey`	`pkey`
`bitstring`		`sskey`	`spkey`

where `key` is the sort for symmetric keys, `skey` and `pkey` are the sorts for private and public keys respectively, `bitstring` is the sort for messages, `sskey` is the sort for signing private keys and `spkey` is the sort for signing public keys.

Let's define the **function symbols**:

$$\begin{aligned} pk &: \texttt{skey} \mapsto \texttt{pkey} \\ spk &: \texttt{sskey} \mapsto \texttt{spkey} \\ sEnc &: \texttt{bitstring} \times \texttt{key} \mapsto \texttt{bitstring} \\ sDec &: \texttt{bitstring} \times \texttt{key} \mapsto \texttt{bitstring} \\ aEnc &: \texttt{bitstring} \times \texttt{pkey} \mapsto \texttt{bitstring} \\ aDec &: \texttt{bitstring} \times \texttt{skey} \mapsto \texttt{bitstring} \\ sign &: \texttt{bitstring} \times \texttt{sskey} \mapsto \texttt{bitstring} \\ getmess &: \texttt{bitstring} \times \texttt{spkey} \mapsto \texttt{bitstring} \\ pair &: \texttt{spkey} \times \texttt{key} \mapsto \texttt{bitstring} \end{aligned}$$

The last function symbol ($pair$) is a way to formalize the input (pk_B, k) used as a parameter in the $sign$ function.

Let's now define the **predicates**:

$$\begin{aligned} A_0 &: \texttt{skey} \times \texttt{pkey} \times \texttt{spkey} \times \texttt{bitstring} \\ A_1 &: \texttt{skey} \times \texttt{pkey} \times \texttt{spkey} \times \texttt{bitstring} \\ A_2 &: \texttt{skey} \times \texttt{pkey} \times \texttt{spkey} \times \texttt{bitstring} \times \texttt{key} \times \texttt{bitstring} \end{aligned}$$

$$\begin{aligned} B_0 &: \texttt{spkey} \times \texttt{sskey} \\ B_1 &: \texttt{spkey} \times \texttt{sskey} \times \texttt{pkey} \times \texttt{key} \\ B_2 &: \texttt{spkey} \times \texttt{sskey} \times \texttt{pkey} \times \texttt{key} \times \texttt{bitstring} \end{aligned}$$

Let's now define the **rules**:

$A_0(pk_A, sk_A, pk_B, s) \to A_1(pk_A, sk_A, pk_B, s), N_1(pk_A)$
$B_0(pk_B, sk_B), N_1(pk_X) \to \exists k.B_1(pk_B, sk_B, pk_X, k), N_2(aEnc(sign(pair(pk_B, k), sk_B), pk_X))$
$A_1(pk_A, sk_A, pk_B, s), N_2(aEnc(sign((pk_B, k), sk_B), pk_A)) \to A_2(pk_A, sk_A, pk_B, k, s), N_3(sEnc(s, k))$
$B_1(pk_B, sk_B, pk_X, k), N_3(sEnc(s, k)) \to B_2(pk_B, sk_B, pk_X, k, s)$

I use pk_X to indicate that the server B accepts any public key.

As we have seen in class, a Man-in-the-Middle attack can easily break this protocol, because the attacker can impersonate A.

Therefore, we use the Dolev-Yao model to model the attacker.

First, I define the specific rules for the attacker to account for the function symbols defined previously.

$$D(\mathsf{sEnc}(m, k)), M(k) \xrightarrow{\mathsf{sDec}} D(m) \tag{1}$$
$$D(\mathsf{aEnc}(m, pk_X)), M(sk_X) \xrightarrow{\mathsf{aDec}} D(m), M(sk_X) \tag{2}$$
$$D(\mathsf{Sign}(m, sk_B)), M(pk_X) \xrightarrow{\mathsf{getmess}} D(m), M(pk_X) \tag{3}$$
$$C(m), M(k) \xrightarrow{\mathsf{sEnc}} C(\mathsf{sEnc}(m, k)) \tag{4}$$
$$C(m), M(pk_X) \xrightarrow{\mathsf{aEnc}} C(\mathsf{aEnc}(m, pk_X)) \tag{5}$$

For the sake of completeness and to enhance readability in the subsequent steps, I also include the standard Dolev-Yao rules derived from the course slides.

$$N_1(x) \longrightarrow D(x) \tag{6}$$
$$N_2(x, y) \longrightarrow D(\langle x, y\rangle) \tag{7}$$
$$D(\langle x, y\rangle) \longrightarrow D(x), D(y) \tag{8}$$
$$D(x) \longrightarrow M(x) \tag{9}$$
$$M(x) \longrightarrow C(x), M(x) \tag{10}$$
$$C(x) \longrightarrow N_1(x) \tag{11}$$
$$C(x), C(y) \longrightarrow C(\langle x, y\rangle) \tag{12}$$
$$C(\langle x, y\rangle) \longrightarrow N_2(x, y) \tag{13}$$
$$\longrightarrow \exists x.M(x) \tag{14}$$

During the trace, it's necessary to do the following two operations: duplicating an item in C and storing both elements of a pair in M. Therefore, I add new rules to describe these operations. This doesn't change the Dolev-Yao model because it is only syntactic sugar:

$$D(x) \longrightarrow D(x), M(x) \tag{15}$$
$$M(\langle x, y\rangle) \longrightarrow M(\langle x, y\rangle), M(x), M(y) \tag{16}$$

In fact, both rules are easily derivable:

$$\begin{aligned}
D(x) &\xrightarrow{(9)} M(x) \xrightarrow{(10)} M(x), C(x) \xrightarrow{(11)} M(x), N(x) \xrightarrow{(6)} M(x), D(x) \\
M(\langle x, y\rangle) &\xrightarrow{(10)} M(\langle x, y\rangle), C(\langle x, y\rangle) \xrightarrow{(13),} M(\langle x, y\rangle), N(\langle x, y\rangle) \\
&\xrightarrow{(7),(6)} M(\langle x, y\rangle), D(\langle x, y\rangle) \xrightarrow{(8)} M(\langle x, y\rangle), C(x), C(y) \\
&\xrightarrow{(9),(9)} M(\langle x, y\rangle), M(x), M(y)
\end{aligned}$$

In the initial state, the intruder (like every client) has his own public and private keys, and the public key of the server.

For to readability, the full trace is divided into steps.

Step 1 In the first step the Intruder takes from the network a key from server B, as if it were a normal client.

$$
\begin{aligned}
& A_0(pk_A, sk_A, pk_B, s), B_0(pk_B, sk_B), M(pk_I), M(sk_I), M(pk_B) \\
\xrightarrow{(10)} & A_0(pk_A, sk_A, pk_B, s), B_0(pk_B, sk_B), M(pk_I), M(sk_I), M(pk_B), C(pk_I) \\
\xrightarrow{(11)} & A_0(pk_A, sk_A, pk_B, s), B_0(pk_B, sk_B), M(pk_I), M(sk_I), M(pk_B), N_1(pk_I) \\
\xrightarrow{B_0 \to B_1} & \exists k. A_0(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), \\
& \quad N_2(\mathsf{aEnc}(\mathsf{sign}(\langle pk_B, k\rangle, sk_B), pk_I))
\end{aligned}
$$

Step 2 In the second step, client A tries to communicate with server B, sending pk_A over the network. This message is stolen by the intruder.

$$
\begin{aligned}
\xrightarrow{A_0 \to A_1} & \exists k. A_1(pk_A, sk_A, pk_B, s), N_1(pk_A), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), \\
& \quad N_2(\mathsf{aEnc}(\mathsf{sign}(\langle pk_B, k\rangle, sk_B), pk_I)) \\
\xrightarrow{(6)} & \exists k. A_1(pk_A, sk_A, pk_B, s), D(pk_A), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), \\
& \quad N_2(\mathsf{aEnc}(\mathsf{sign}(\langle pk_B, k\rangle, sk_B), pk_I)) \\
\xrightarrow{(9)} & \exists k. A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A), \\
& \quad N_2(\mathsf{aEnc}(\mathsf{sign}(\langle pk_B, k\rangle, sk_B), pk_I))
\end{aligned}
$$

Step 3 In the third step, the intruder can use the message $\mathsf{aEnc}(\mathsf{Sign}(pk_B, k, sk_B), pk_I)$ from server B to obtain the k.

$$
\begin{aligned}
\xrightarrow{(6)} & \exists k. A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A), \\
& \quad D(\mathsf{aEnc}(\mathsf{sign}(\langle pk_B, k\rangle, sk_B), pk_I)) \\
\xrightarrow{(2)} & \exists k. A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A), \\
& \quad D(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)) \\
\xrightarrow{(15)} & \exists k. A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A), \\
& \quad M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), D(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)) \\
\xrightarrow{(3)} & \exists k. A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A), \\
& \quad M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), D(\langle pk_B, k\rangle) \\
\xrightarrow{(9)} & \exists k. A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A), \\
& \quad M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), M(\langle pk_B, k\rangle)
\end{aligned}
$$

Step 4 The intruder sends to A the same message obtained from server B, but encrypted with A's public key, instead of pk_I.

$$\xrightarrow{(10)}\exists k.A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A),$$
$$M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), M(\langle pk_B, k\rangle), C(\mathsf{sign}(\langle pk_B, k\rangle, sk_B))$$
$$\xrightarrow{(5)}\exists k.A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A),$$
$$M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), M(\langle pk_B, k\rangle), C(\mathsf{aEnc}(\mathsf{sign}(\langle pk_B, k\rangle, sk_B), pk_A))$$
$$\xrightarrow{(11)}\exists k.A_1(pk_A, sk_A, pk_B, s), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A),$$
$$M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), M(\langle pk_B, k\rangle), N_2(\mathsf{aEnc}(\mathsf{sign}(\langle pk_B, k\rangle, sk_B), pk_A))$$

Step 5 Client A sees the message and sends the secret encrypted with the key obtained:

$$\xrightarrow{A_1 \to A_2}\exists k.A_2(pk_A, sk_A, pk_B, s, k), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A),$$
$$M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), M(\langle pk_B, k\rangle), N_3(\mathsf{sEnc}(s, k))$$

Step 6 The intruder takes the message from the network and obtains the secret s using the key stolen in step 3

$$\xrightarrow{A_1 \to A_2}\exists k.A_2(pk_A, sk_A, pk_B, s, k), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A),$$
$$M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), M(\langle pk_B, k\rangle), D(\mathsf{sEnc}(s, k))$$
$$\xrightarrow{(16)}\exists k.A_2(pk_A, sk_A, pk_B, s, k), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A),$$
$$M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), M(\langle pk_B, k\rangle), M(pk_B), M(k), D(\mathsf{sEnc}(s, k))$$
$$\xrightarrow{(1)}\exists k.A_2(pk_A, sk_A, pk_B, s, k), B_1(pk_B, sk_B, pk_I, k), M(pk_I), M(sk_I), M(pk_B), M(pk_A),$$
$$M(\mathsf{sign}(\langle pk_B, k\rangle, sk_B)), M(\langle pk_B, k\rangle), M(pk_B), M(k), D(s)$$

So the Intruder has the secret and this implies that the Handshake Protocol, as it stands, isn't secure.

Exercise 3.

The exercise requires formalizing the following protocol and determining whether an adversary is capable of finding the value of m or p.

$$A \to C : \{j\}_k$$
$$B \to C : \{i\}_h$$
$$C \to D : f(i, j)$$
$$D \to A : \{\{d(m)\}_i\}_j$$
$$D \to B : g(p)$$

Where m, p are messages, i, j, k, h are private keys, f, g are one-way functions and d allows deriving m from $d(m)$.

The protocol can be formalized as follows:

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

(* we need to cast key to bitstring, i.e. first two operations *)
(* 4.1.2. https://bblanche.gitlabpages.inria.fr/proverif/manual.pdf *)
fun k2b(key):bitstring [data].
```

```
10
11 (* One Way Function(s) *)
12 fun f(key,key): bitstring.
13 fun g(bitstring):bitstring.
14
15 (* d function *)
16 fun d(bitstring):bitstring.
17 reduc forall m: bitstring;
18    getd(d(m))=m.
19
20 free i:key[private].
21 free j:key[private].
22 free k:key[private].
23 free h:key[private].
24
25 free m:bitstring[private].
26 free p:bitstring[private].
27
28 free c:channel.
29
30
31 query attacker(m).
32 query attacker(p).
33
34
35 let processA=
36 (* A -> C : {j}k *)
37 let msg=k2b(j) in
38 out(c,senc(msg,k));
39 in(c, dmij:bitstring).
40
41
42 let processB=
43 (* B -> C : {i}h *)
44 let msg=k2b(i) in
45 out(c,senc(msg,h));
46 in(c, gp:bitstring).
47
48
49 let processC=
50 in(c,jk:bitstring);
51 in(c,ih:bitstring);
52 (* C -> D : f (i, j) *)
53 out(c,f(i,j)).
54
55 let processD=
56 in(c,fij:bitstring);
57 (* D -> A : {{d(m)}i}j *)
58 out(c, senc(senc(d(m),i),j));
59 (* D -> B : g(p) *)
60 out(c, g(p)).
61
62 process
63     (processA | processB | processC | processD)
```

The output is:

```
1 -- Query not attacker(m[]) in process 0
2 Translating the process into Horn clauses...
3 Completing...
4 Starting query not attacker(m[])
5 RESULT not attacker(m[]) is true.
6 -- Query not attacker(p[]) in process 0
7 Translating the process into Horn clauses...
8 Completing...
9 Starting query not attacker(p[])
10 RESULT not attacker(p[]) is true.
11
```

```
12 ------------------------------------------------------------
13 Verification summary:
14
15 Query not attacker(m[]) is true.
16
17 Query not attacker(p[]) is true.
18
19 ------------------------------------------------------------
```

Therefore, the protocol is secure.