Advanced Cryptography — Midterm Exam
Solution

Serge Vaudenay

7.5.2012

- duration: 3h00
- any document is allowed
- a pocket calculator is allowed
- communication devices are not allowed
- the exam invigilators will not answer any technical question during the exam
- the answers to each exercise must be provided on separate sheets
- readability and style of writing will be part of the grade
- do not forget to put your name on every sheet!

The exam grade follows a linear scale in which each question has the same weight.

1 Decryption Attack on Broadcast RC4

This exercise is inspired from Isobe-Ohigashi-Watanabe-Morii, Full Plaintext Recovery Attack on Broadcast RC4*, to be published in the proceedings of FSE 2013, LNCS, Springer.*

The RC4 pseudorandom number generator is defined by a state and an algorithm which update the state and produces an output byte. In RC4, a state is defined by

- two indices i and j in $\mathbf{Z}_{256}$;
- one permutation S of $\mathbf{Z}_{256}$.

By abuse of notation we write $S(x)$ for an arbitrary integer x as for $S(x \bmod 256)$. The state update and output algorithm works as follows:

1: $i \leftarrow i + 1$
2: $j \leftarrow j + S(i)$
3: exchange the values at position $S(i)$ and $S(j)$ in table S
4: output $z_i = S(S(i) + S(j))$

Q.1 Assume that the initial S is a random permutation with uniform distribution and that i and j are set to 0.

Q.1a What is the probability that $[S(1) \neq 2 \text{ and } S(2) = 0]$?

It is $\frac{1}{N} \times \frac{N-2}{N-1}$ *with* $N = 256$.

Q.1b If $S(1) \neq 2$ and $S(2) = 0$ hold, show that the second output z_2 is always 0.

Let $S(1) = x$ *and* $S(x) = y$ *initially. At the first iteration,* i *is set to 1,* j *is set to* x*, and* $S(1)$ *and* $S(x)$ *are exchanged. There values become* y *and* x *respectively. Then,* i *is set to 2,* j *is set to* x *again, and* $S(2)$ *and* $S(x)$ *are exchanged. There values become* x *and* 0 *respectively. The output is* $S(x)$ *which is 0.*

Q.1c In other cases, we assume that $z_2 = 0$ with probability close to $\frac{1}{256}$.
Deduce $p = \Pr[z_2 = 0]$. What do you think of this probability?

Clearly, $p = \frac{1}{N} \times \frac{N-2}{N} + \frac{1}{N} \approx \frac{2}{N}$. This is twice that what we should expect. This is a deviant property.

Q.2 Here, we let $p = \Pr[z_2 = 0]$ and we assume that $\Pr[z_2 = x] = \frac{1-p}{N-1}$ for all $x \neq 0$ and $N = 256$. We consider that a message m is encrypted by XORing to the stream generated by RC4. I.e., the ciphertext c is such that $c_i = m_i \oplus z_i$. We assume that the *same* message m is encrypted n many times and that the adversary collected the ciphertext. Each encryption starts with an independent random permutation. Let n_x be the number of occurrences of the byte x in c_2. I.e., there are n_x collected ciphertexts c such that $c_2 = x$ in total.

Q.2a Compute the expected value of n_x for $x = m_2$ then for any fixed $x \neq m_2$.

Since $c_2 = m_2$ is equivalent to $z_2 = 0$, clearly, $E(n_{m_2}) = np$ and $E(n_x) = n\frac{1-p}{N-1}$ for all $x \neq m_2$.

Q.2b For $x \neq m_2$ fixed, express $n_{m_2} - n_x$ as a sum of n independent identically distributed (iid) random variables X_i which take values in $\{-1, 0, 1\}$ and compute their expected value.

We define $X_i = 1$ if $c_2 = m_2$ in the ith ciphertext, $X_i = -1$ if $c_2 = x$ in the ith ciphertext, and $X_i = 0$ otherwise. Clearly, $n_{m_2} - n_x = \sum_{i=1}^{n} X_i$. The X_i's are iid. Finally,

$$E(X_i) = \frac{1}{n} E(n_{m_2} - n_x) = p - \frac{1-p}{N-1} = \frac{Np-1}{N-1}$$

Q.2c We recall the Hoeffding bound:

Theorem 1 (Hoeffding). *Let $X_1, \ldots, X_n$ be n iid random variables which take values in $[a, b]$ and expected value μ. For any $t > 0$, we have*

$$\Pr\left[\sum_{i=1}^{n} X_i \leq \mu - t\right] \leq e^{-\frac{2nt^2}{(b-a)^2}}$$

Give an upper bound for $\Pr[n_{m_2} \leq n_x]$ for any $x \neq m_2$.
Deduce an upper bound for the event that n_{m_2} is not the largest counter value n_x.

We take $a = -1$, $b = 1$, $\mu = t = \frac{Np-1}{N-1}$ and we obtain

$$\Pr[n_{m_2} - n_x \leq 0] \leq e^{-\frac{n}{2}\mu^2} = e^{-\frac{n}{2}\left(\frac{Np-1}{N-1}\right)^2}$$

So,

$$\Pr\left[\bigvee_{x \neq m_2} (n_{m_2} \leq n_x)\right] \leq (N-1)e^{-\frac{n}{2}\left(\frac{Np-1}{N-1}\right)^2}$$

Q.2d Propose an algorithm to decrypt m_2 and a lower bound on its probability of success.
What is the required number of ciphertexts to decrypt well almost certainly?
Propose a numerical application with the values from this exercise.

The algorithm simply collects n ciphertexts and compute $\arg\max_x n_x$ *which is* m_2 *except with probability bounded by* $(N-1)e^{-\frac{n}{2}\left(\frac{Np-1}{N-1}\right)^2}$. *By taking*

$$n = 2(s\ln 2 + \ln(N-1))\left(\frac{N-1}{Np-1}\right)^2$$

the probability is bounded by 2^{-s}. *Concretely, with* $p \approx \frac{2}{N}$, *we obtain* $n \approx 2N^2\ln(N2^s)$. *For instance, with* $n = 2^{21}$, *we have* $s = 15$. *So, we decrypt* m_2 *correctly almost for sure when collecting two millions of ciphertexts encrypting the same message.*

2 Generic Attacks on Multiple Encryption

This exercise is inspired from Efficient Dissection of Composite Problems... *by Dinur, Dunkelman, Keller, and Shamir. Published in the proceedings of Crypto'12 pp. 719–740, LNCS vol. 7417 Springer 2012.*

We consider a block cipher E with n-bit blocks and n-bit keys. We denote by D the decryption algorithm. A r-time encryption is a process of encrypting a plaintext P into $C = E_{k_r}(\cdots E_{k_1}(P)\cdots)$. We consider the problem of key recovery for a multiple encryption, with a few known plaintext/ciphertext pairs. I.e., we assume that the adversary knows some pairs (P_i, C_i), for $i = 1, \ldots, r$, and want to find all $(k_1, \ldots, k_r)$ which would encrypt each P_i to C_i. In what follows, we consider the worst case complexity.

Q.1 Give an algorithm for $r = 1$. What are its time complexity and memory complexity?

For each k, we compute $E^k(P_1)$. If it matches C_1, then we print k and continue.
This is exhaustive search. The time complexity is 2^n and the memory complexity is constant.

Q.2 Give an algorithm for $r = 2$. What are its time complexity and memory complexity?

For each k_1, we compute $x = E_{k_1}(P_1)$ and store (x, k_1) in a hash table, keyed by the first value (i.e., stored at the address $h(x)$ in memory). Then, for each k_2, we compute $y = D_{k_2}(C_1)$. If there is at address $h(y)$ some record (x, k_1) with $x = y$, then we compute $E_{k_2}(E_{k_1}(P_2))$ and compare it with C_2. If they match, we print (k_1, k_2) and continue.
This is meet-in-the-middle. The time complexity is twice 2^n and the memory complexity is 2^n.

Q.3 We now consider $r = 4$.

Q.3a Given $P_1, P_2, B_1 \in \{0,1\}^n$, how many (B_2, k_1, k_2) triplets are such that $E_{k_2}(E_{k_1}(P_i)) = B_i$ for $i = 1, 2$?
Propose an algorithm with time-complexity $\mathcal{O}(2^n)$ and memory complexity $\mathcal{O}(2^n)$ to list them all.

We have an equation on $2n$ bits (P_i mapped to B_i for $i = 1, 2$) and $3n$ bits of unknowns (for B_2, k_1, k_2). So, we shall expect 2^n triplets on average.
We do a meet-in-the-middle and split with E_{k_1} and E_{k_2}. That is, for all k_1, we compute and store in a table $E_{k_1}(P_1)$. Then, for all k_2, compute $D_{k_2}(B_1)$ and see if there is a match in the table. In the case of a match, we obtain a (k_1, k_2) pair and we can compute $B_2 = E_{k_2}(E_{k_1}(P_1))$ to be listed. Clearly, we store 2^n in the table and the complexity is 2^n.

Q.3b Given P_1, P_2, B_1, C_1, C_2 and a list of (B_2, k_1, k_2) such that $E_{k_2}(E_{k_1}(P_i)) = B_i$ for $i = 1, 2$ from the previous algorithm, propose an algorithm to list all $(k_1, \ldots, k_4)$ such that $E_{k_4}(\cdots E_{k_1}(P_i)\cdots) = C_i$ for $i = 1, 2$ and $E_{k_2}(E_{k_1}(P_1)) = B_1$.

We do a standard meet-in-the-middle attack with E_{k_3} and E_{k_4}, based on B_1 and C_1. So, we list all (k_3, k_4) mapping B_1 to C_1. For each of the 2^n pairs found, we can decrypt C_2 and see if it matched any B_2 in the list. If it does, we have a (k_1, k_2, k_3, k_4) solution to be printed.

Q.3c Propose an algorithm with time-complexity $\mathcal{O}(2^{2n})$ and memory complexity $\mathcal{O}(2^n)$ to solve the key recovery problem.

We iterate the previous algorithm for each B_1, and for each of the (k_1, k_2, k_3, k_4) found, we check on-the-fly the constraints with P_3, P_4, C_3, C_4. The previous attack runs with complexity 2^n and memory complexity 2^n. So, the final attack runs with time complexity 2^{2n} and memory complexity 2^n.

Q.4 We now consider $r = 7$.

Q.4a Given $P_1, P_2, B_1, B_2 \in \{0,1\}^n$, how many (k_1, k_2, k_3) triplets are expected to satisfy the relations $E_{k_3}(E_{k_2}(E_{k_1}(P_i))) = B_i$ for $i = 1, 2$?
Propose an algorithm with time-complexity $\mathcal{O}(2^{2n})$ and memory complexity $\mathcal{O}(2^n)$ to list them all.

We have an equation on $2n$ bits (P_i mapped to B_i for $i = 1, 2$) and $3n$ bits of unknowns (for k_1, k_2, k_3). So, we shall expect 2^n triplets on average.
We do a meet-in-the-middle and split with $E_{k_2} \circ E_{k_1}$ and E_{k_3}. That is, for all k_1, we compute and store in a table $(E_{k_1}(P_1), E_{k_1}(P_2))$. Then, for all (k_2, k_3), compute $D_{k_2}(D_{k_3}(B_i))$ for $i = 1, 2$ and see if there is a match in the table. In the case of a match, we obtain a triplet to be listed. Clearly, we store 2^n in the table and the complexity is 2^{2n}.

Q.4b Given $P_1, \ldots, P_7, B_1, B_2 \in \{0,1\}^n$, propose an algorithm with time-complexity $\mathcal{O}(2^{2n})$ and memory complexity $\mathcal{O}(2^n)$ to list all $(B_3, \ldots, B_7)$ such that there exists a (k_1, k_2, k_3) triplets are such that $E_{k_3}(E_{k_2}(E_{k_1}(P_i))) = B_i$ for $i = 1, \ldots, 7$.

We just change the last algorithm: whenever a new triplet (k_1, k_2, k_3) is found, we just compute and list $(B_3, \ldots, B_7)$. The complexity is the same.

Q.4c By combining the algorithms of Q.4b and Q.3, propose an algorithm to do the key recovery for 7-multiple encryption, with time complexity $\mathcal{O}(2^{4n})$ and memory complexity $\mathcal{O}(2^n)$.

For all B_1 and B_2, we run the following loop. First, we run the algorithm of Q.4b to list all $(B_3, \ldots, B_7)$. The elements of this list are stored in a new hash table of size 2^n. We can even assume storing (k_1, k_2, k_3) at the address of this tuple. Then, we change a bit the algorithm of Q.3: instead of printing $(k_4, \ldots, k_7)$, we decrypt $C_3, \ldots, C_7$ with the newly obtained quadruplet and look for a match in the hash table. If there is a match, we can print $(k_1, \ldots, k_7)$. (This is the trick to avoid having to store 2^{2n} tuples!) Otherwise, we just continue.
The loop is done 2^{2n} and has a complexity of 2^{2n}. The storage does not exceed a complexity of 2^n.

3 Another Attack on Broadcast RSA

This exercise is inspired from Solving Systems of Modular Equations in One Variable... *by May and Ritzenhofen. Published in the proceedings of PKC'08 pp. 37–46, LNCS vol. 4939 Springer 2008.*

Q.1 Let $N_1 = 235$, $N_2 = 451$, $N_3 = 391$ be three RSA moduli, all working with the public exponent $e = 3$. Let $y_1 = 99$, $y_2 = 238$, $y_3 = 278$ be the respective encryption of the same x under the three RSA keys. Compute x without factoring any moduli.
Hint: $(N_2N_3)^{-1} \bmod N_1 = 31$, $(N_1N_3)^{-1} \bmod N_2 = 72$, $(N_1N_2)^{-1} \bmod N_3 = 277$.

By applying the Chinese Remainder Theorem, we obtain that x^3 *is equal to* $31y_1N_2N_3 + 72y_2N_1N_3 + 277y_3N_1N_2$ *modulo* $N_1N_2N_3$. *By doing the computation, we obtain on a normal pocket calculator* $59\,300 + \varepsilon$ *with* $0 \leq \varepsilon < 100$. *(We are loosing the least two significant digits due to the imprecision of floating point arithmetic.) Since* x *must be lower than the moduli, we deduce that* $x^3 = 59\,300 + \varepsilon$. *By extracting the cubic root of* $59\,300$, *we obtain about* 38.996. *So, we deduce (and we can check) that* $x = 39$.

Q.2 Let (N_i, e_i), $i = 1, \ldots, r$ be r different RSA public keys, with pairwise coprime moduli. Let $y_i = x^{e_i} \bmod N_i$, for some positive x which is lower than all moduli. Let $e = \max_i e_i$ and $N = N_1 \cdots N_r$. We assume that an adversary knows all public keys and all y_i but not x.

Q.2a Show that for each i, there is a monic polynomial $P_i(z)$ of degree e which can be computed by the adversary and such that $P_i(x) \equiv 0 \pmod{N_i}$.

Clearly, $P(z) = (z^{e_i} - y_i)z^{e-e_i}$ *is such polynomial.*

Q.2b Deduce that there is a monic polynomial $P(z)$ of degree e which can be computed by the adversary and such that $P(x) \equiv 0 \bmod N$.

We write $P(z) = \sum_{j=0}^{e} a_j z^j$ *and solve the system* $P(z) \equiv P_i(z) \pmod{N_i}$ *for all* i. *That is, we apply the Chinese Remainder Theorem to compute* a_j *such that* a_j *modulo* N_i *is the coefficient of* z^j *in* $P_i(z)$. *We obtain the required polynomial. We observe that* a_e *modulo all* N_i *is 1, so* $a_e = 1$*: we have a monic polynomial.*

Q.2c Deduce an algorithm to solve x, for r large enough. How large?
We recall the Coppersmith result: Let $f(z)$ be a monic polynomial of degree e in one variable modulo N. There is an efficient algorithm to find all roots x such that $0 \leq x \leq N^{\frac{1}{e}}$.

We now have to solve $P(x) \equiv 0 \pmod{N}$ *with degree* e. *We observe that* $P(z)$ *is a monic polynomial of degree* e *in one variable modulo* N. *When* $r \geq e$, *the root we are looking for is such that* $0 \leq x \leq N^{\frac{1}{e}}$. *So, we can find it efficiently. We just need* $r \geq e$.