Tel Aviv University 15/12/2016

Introduction to Modern Cryptography (0368.3049) – Ex. 4

Benny Chor and Orit Moskovich

Submission in singles or pairs to Orr Fischer's Schreiber mailbox (289) until 28/12/2016, 23:59 (IST)

- Appeals/missing grade issues: bdikacs AT gmail.com
- Issues regarding missing/unchecked assignments will be addressed only if a soft copy will be submitted on time to: crypto.f16 AT gmail.com.
Subject of the email: Ex.4, ID

1. **Testing Primality of Charmichael Numbers:**
   This question deals with a test aimed at determining that Charmichael numbers are composites. As discussed in class (lecture 6, slides 39-40), if m is a Charmichael number, then for every $1 < a \leq m-1$, if $\gcd(a,m) = 1$, then $a^{m-1} = 1 \pmod{m}$. Thus, if all prime factors of m are large (such numbers do exist), most candidate witnesses will be relatively prime to m and will also provide no evidence for compositeness in the standard Fermat test. In this question we will go through a concrete example for the compositeness test developed for Charmichael numbers.

   Consider the Charmichael number[1] $m = 90256390764228001$. Its prime factorization is $m = 380251 \cdot 410671 \cdot 577981$, and $m-1$ factorization is $m - 1 = 2^5 \cdot 3^6 \cdot 5^3 \cdot 13^2 \cdot 19^2 \cdot 61 \cdot 8317$. Let $1 < a < m$ be an integer. Since $2^5 = 32$ divides $m-1$, the exponentiations (all modulo m) $a^{(m-1)/32}, a^{(m-1)/16}, \ldots, a^{(m-1)/2}, a^{m-1}$ are all well defined.

   Write a short Sage program that chooses at random 100 a in the range $1 < a < m$. For each of those, compute $\gcd(a,m)$ and the *largest* $i, 1 \leq i \leq 5$ such that $a^{(m-1)/2^i} \neq \pm 1$ (in Z_m, -1 is simply $m-1$), but $a^{(m-1)/2^{i-1}} = 1$.

   Submit your code and the following statistics: How many a's had $\gcd(a,m) \neq 1$ (with high probability you won't see any), for how many a's the largest such i equals $5, 4, 3, 2, 1$, or that no such i exists (though that would surprise us). Briefly explain why any a with $i, 1 \leq i \leq 5$ provides a *proof* that m is composite.

2. **Square Roots and Factorization:** We are given a composite number, m, which is n bits long, and we are told it is a product of two large primes $m = p \cdot q$. Recall that every square $x = z^2 \in Z^*_{pq}$ has *four* square roots in Z^*_{pq}.

   Suppose we are now supplied with a blackbox deterministic algorithm $\mathcal{A}$ (we can feed it with several inputs and observe the outputs, but have no access to its internal working). On input $y \in Z^*_{pq}$, $\mathcal{A}$ produces one of the following: If y is not a quadratic residue, then $\mathcal{A}$ outputs the text ``go catch a Stellagama stellio'' (it sounds better in Hebrew, as you could see in the original, below). If $y = x^2$ is a quadratic residue, $\mathcal{A}$ outputs *one* square root of y.

[1] taken from a paper by G.E. Pinch, titled "the Charmichael numbers up to 10^{17}".

Suppose on input y, $\mathcal{A}$ takes $t(n)$ steps. Show how to use $\mathcal{A}$ in order to factor m with high probability in $O(t(n))$ steps. Explain your analysis, and why randomization is essential in it.

3. **Pollard's ρ Algorithm:**
Write a short Sage or Python code that implements Pollard's ρ factoring algorithm. Let x_0 (the starting point) and c (of the "random function" $F(z) := z^2 + c$) be two parameters in your program.

   1) Choose at random two prime numbers p and q such that $2^{45} < p < 2^{46}$ and $2^{47} < q < 2^{48}$, and let $m = pq$. Print p, q and m. Run your implementation with $c = 1$ and with four additional values of c. For each c, run three different starting points x_0. For each choice print x_0, c, the number of iterations, i, to factor N, and whether the factor found was p or q. Compare the number of iterations to $\sqrt{p}$. (Do not print intermediate results. Also set up some upper bound and abort in case a factor is not found after that many iterations.)

   Can you make *any* recommendation of preferred values for c and x_0 based on this small scale experiment?

   2) Execute the same instructions as in (1), only this time use the "random function" $F(z) := z + c$. Did your program terminate in any of the executions? If not, explain why you think this is the case.

4. **Implementing RSA:**
In this problem we will implement an instance of the RSA cryptosystem using Sage/Python. Start by choosing *at random* two prime numbers p and q. The prime number p should be 82 *digits* long and $p - 1$ should have a prime factor that is at least 72 *digits* long. The prime number q should be 77 *digits* long and $q-1$ should have a prime factor that is at least 70 *digits* long. Let $N = pq$. Pick at random e and d that are appropriate encryption and decryption RSA exponents.

   1) Print (with appropriate headings so we know what these numbers are) the numbers N, p, q, e and d, and also the complete factorizations of $p-1$ and of $q-1$. As a "scale for measuring lengths" print 10^{82} and 10^{72} as well so they are aligned with p and q respectively. Explain (in plain language, not in code) how p and q were found and especially how the random choices were made.

(To make this "visual comparison" meaningful, fixed width fonts should be used for Maple's output. Here is a way to achieve this, courtesy of Meir Marom and Gil Berkovski from the 2002 course: In Maple, go to the "Format" menu, and select "Styles...". A new window will appear, with various items in it, specifically a list. Go down that list and choose "2D Output", and click on the button "Modify". A new window will appear, with a list of available fonts. Choose "Courier New" or any other fixed width font, click "Ok", and then click the "Done" button. All the opened windows should now be closed, and you'll be back to the main Maple window. You're done. The fonts of the output are changed.)

2) Use the simple coding scheme presented in class (space=`00`, `A`=`01`, `B`=`02`,...,`Z`=`26`). Make up a short text, encode it (ascii to numbers), encrypt it under your public key, then decrypt using the private key. Print the plaintext message, its encryption and the decryption.

5. Let p be a prime and let $g \in \mathbb{Z}_p^*$ be a generator. Suppose that there exists a polynomial-time algorithm A that given $p, g, g^x \bmod p$ finds x for $\frac{1}{1000}$ of the possible x's. Show how to use A as a subroutine to construct a probabilistic polynomial time algorithm B that solves the DL problem for all instances (i.e., for every $x \in \mathbb{Z}_p^*$) with probability $\frac{1}{2}$. Analyze the running time of B.

6. Consider the following public-key encryption scheme. The public key is $(G, q, g, h = g^x)$ and the private key is x, generated exactly as in the ElGamal encryption scheme. In order to encrypt a bit b, the sender does the following:

   - If $b = 0$ then choose a random $y \in \mathbb{Z}_q$ and compute $c_1 = g^y$ and $c_2 = h^y$. The ciphertext is (c_1, c_2).
   - If $b = 1$ then choose independent random $y, z \in \mathbb{Z}_q$, compute $c_1 = g^y$ and $c_2 = g^z$, and set the ciphertext equal to (c_1, c_2).

   (a) Show that it is possible to decrypt efficiently (with some negligible error probability) given knowledge of the secret-key x.

   (b) Prove that this encryption scheme is CPA-secure if the Decisional Diffie-Hellman problem is hard.

7. *Theorem:* If an encryption scheme is ε-CPA-secure (for one message), then it is ε_t-CPA-secure for t messages.

   We have proved that if an encryption scheme is ε-CPA-secure, then it is ε_2-CPA-secure for encryption of 2 messages.

   In this question, we will generalize the hybrid argument we have seen in class to t messages:

   Step 1: Define the vectors:

$$C^i = \big(\underbrace{Enc_{pk}(m_0^1), ..., Enc_{pk}(m_0^i)}_{i \text{ terms}}, \underbrace{Enc_{pk}(m_1^{i+1}), ...Enc_{pk}(m_1^t))}_{t-i \text{ terms}} \big)$$

   Step 2: Define the experiment for A_{mult} as follows:

(a) A random key (pk, sk) is generated using Gen
(b) A_{mult} is given pk and outputs a pair of vectors $M_0 = (m_0^1, ..., m_0^t)$ and $M_1 = (m_1^0, ..., m_1^t)$
(c) A random bit $b \leftarrow \{0, 1\}$ is chosen
(d) The vector $C = (Enc_{pk}(m_b^1), ..., Enc_{pk}(m_b^t))$ is given to A_{mult}
(e) A_{mult} outputs a bit b'

Step 3: Define A_1 as follows:

(a) A random key (pk, sk) is generated using Gen
(b) A_1 is given pk and runs A_{mult} to obtain a pair of vectors $M_0 = (m_0^1, ..., m_0^t)$ and $M_1 = (m_1^0, ..., m_1^t)$
(c) A_1 chooses a random index $i \leftarrow \{1, ..., t\}$ and outputs the pair m_0^i, m_1^i
(d) A random bit $b \leftarrow \{0, 1\}$ is chosen
(e) A_1 is given $c^i = Enc_{pk}(m_b^1)$
(f)
- For $j < i$: A_1 computes $c^j = Enc_{pk}(m_0^j)$
- For $j > i$: A_1 computes $c^j = Enc_{pk}(m_1^j)$
- A_1 generates the vector $C = (c_1, ..., c_i, ..., c_t)$ and give the result to A_{mult}

(g) A_1 outputs the bit that is output by A_{mult}

Then, assuming the encryption scheme is ε-CPA secure:

$$Pr[A_1 \text{ wins}] \leq \frac{1}{2} + \varepsilon$$

Step 4: (This is your task!) Use A_1 in order to prove:

$$\begin{aligned} Pr[A_{mult} \text{ wins}] &= \frac{1}{2} \cdot Pr[A_{mult} \text{ outputs 0 on } (Enc_{pk}(m_0^1), ..., Enc_{pk}(m_0^t))] \\ &\quad + \frac{1}{2} \cdot Pr[A_{mult} \text{ outputs 1 on } (Enc_{pk}(m_1^1), ..., Enc_{pk}(m_1^t))] \\ &= \frac{1}{2} \cdot Pr[A_{mult} \text{ outputs 0 on } C^0] \\ &\quad + \frac{1}{2} \cdot Pr[A_{mult} \text{ outputs 1 on } C^t] \\ &\leq \frac{1}{2} + \varepsilon_t \end{aligned}$$

For this, you might want to consider $Pr[A_1 \text{ outputs } 0 | b = 0] =?$ and $Pr[A_1 \text{ outputs } 1 | b = 0] =?$ in terms of i (use the law of total probability).

8. Let p, q be two n-bit primes, chosen at random in the corresponding range. Let $m = pq$, and a be chosen at random in the range $2 < a < m - 2$. Given a positive integer t, how many modular multiplications of $O(n)$ bit numbers does it take to compute $a^{2^t} \pmod m$, as a function of t and n (using good old iterated squaring, which you all saw back in the CS1001.py course):

- When the factorization of m is unknown.
- When the factorization of m is known.