

## Introduction to Modern Cryptography

Benny Chor and Rani Hod

## Assignment #3 (version 1.3)

Published Monday, March 10, 2008. Due Tues., March 25, in Rani Hod's mailbox (second floor, Schreiber building).

Please limit your answers to **one page per question**. Queries will be answered only if made till **March 22**. Please check the course web site periodically for possible corrections or updates.

**Problem A: Coin Flipping Over the Phone**

In class, we will discuss the application of one way functions to enable two parties that do not trust each other to flip coins over the phone. We assume that the parties do not deviate from the *syntax* of the protocol. Still, they may try to cheat in different ways, provided the messages they send have the right format. For example, Alice could send a composite number where the protocol calls for a prime number. (This specific attempt is not very smart since Bob will easily detect it.)

In class, we will discuss a specific implementation using RSA, but more generally the protocol can be cast as follows: Let the two parties be good old Alice & Bob. Alice chooses  $F : D \rightarrow D$  – a one-way, one-to-one function. Let  $B : D \rightarrow \{0, 1\}$  be an efficiently computable predicate on elements of  $D$ . Alice starts by sending a description of  $F$  and of  $B$  to Bob. The general protocol then proceeds as follows:

- (a) Alice picks an  $x \in D$ , computes  $y = F(x)$  and  $b = B(x)$ , and sends  $y$  to Bob. (This is supposed to create a *commitment* to the value  $x$ .)
- (b) Bob send to Alice his guess for  $B(x)$ , namely a bit  $c \in \{0, 1\}$ . The bit  $c$  could either be the result of a coin flip or the outcome of some efficient algorithm applied by Bob in an attempt to guess  $B(x)$ .
- (c) After receiving  $c$ , Alice sends  $x$  to Bob, who can now compute  $b = B(x)$  on his own.
- (d) If  $c = b$  then Bob wins the coin toss. Else ( $c \neq b$ ) Alice wins.

The coin flip described in class had  $D = Z_N^*$ ,  $F(x) = x^e \bmod N$ , and  $B(x) =$  the least significant bit of  $x$ .

1. Explain what goes wrong if the order of steps (a) and (b) is reversed.
2. Let  $D = Z_p^*$ ,  $F(x) = g^x \bmod p$  where  $p$  is a prime number and  $g$  is a primitive element in  $Z_p^*$ . Let  $B(x) =$  be the least significant bit of  $x$ . Show that Bob can win this game with probability 1.
3. Is the assumption that  $F$  is one-to-one necessary?
  - (a) Give an *concrete example* where  $F(x) = x^e \bmod N$  is not one-to-one and Alice can cheat and win the game with probability 1.
  - (b) Give an *concrete example* where  $F(x) = g^x \bmod p$  is not one-to-one and Alice can cheat and win the game with probability 1.

- (c) Give an example where  $F$  is not one-to-one but neither Alice nor Bob can cheat. You can make reasonable assumptions if they do indeed make sense, but make sure to state them carefully.

**Problem B: Hard Core Predicates**

Let  $F : D \rightarrow D$  be a one-way, one-to-one function. Let  $B : D \rightarrow \{0, 1\}$  be an efficiently computable predicate on elements of  $D$ . Let  $A$  be a blackbox (or “magic box”) that on input  $y = F(x)$  produces as output the bit  $B(x)$ . We say that  $B$  is a *hard core bit* for  $F$  if there is an efficient algorithm that inverts  $F$ , using  $A$  as a subroutine (each call to  $A$  is counted as one step). In class we stated (without proof) that  $B(x) =$  “least significant bit of  $x$ ” is a hard core bit for the RSA function  $F(x) = x^e \bmod N$  on  $D = Z_N^*$ .

- 1) Let  $D = Z_p^*$ ,  $F(x) = g^x \bmod p$  where  $p$  is a prime number and  $g$  is a primitive element in  $Z_p^*$ . Define

$$\text{Half}_p(x) = \begin{cases} 0 & \text{if } 1 \leq x < (p-1)/2 \\ 1 & \text{if } (p-1)/2 \leq x \leq p-1 \end{cases}$$

Show that  $\text{Half}_p(\cdot)$  is a hard core predicate for  $F(x) = g^x \bmod p$ .

- 2) Let  $D = Z_N^*$ ,  $F(x) = x^e \bmod N$  where  $N = pq$ , both  $p$  and  $q$  are prime numbers, and  $e$  is relatively prime to  $(p-1)(q-1)$ . The numbers  $e$  and  $N$  are known, but  $N$ 's factorization is not given. Define

$$\text{Half}_N(x) = \begin{cases} 0 & \text{if } 1 \leq x < N/2 \\ 1 & \text{if } N/2 \leq x \leq N-1 \end{cases}$$

Show that  $\text{Half}_N(\cdot)$  is a hard core predicate for  $F(x) = x^e \bmod N$ .

- 3) Suppose  $B : D \rightarrow \{0, 1\}$  is a hard core predicate for  $F$  as defined above, and  $F : D \rightarrow D$  is a one-way, one-to-one function. Is it true that the coin flipping protocol from question 1 is indeed fair (*i.e.* no side has any non-negligible advantage compared to a “regular” unbiased coin). Provide a short proof if the answer is positive, and short, convincing evidence if your answer is negative.

**Problem C: Low Exponent RSA**

In class, we discussed a problem that occurs for low exponent RSA (specifically we had  $e = 3$ ) when the same plaintext  $x$  is encrypted with three different moduli ( $x^3 \bmod N_i$ ,  $i = 1, 2, 3$ ). We stated, however, that there is no known problem when encrypting modulo a *single*  $N$ .

Suppose  $N = pq$  is  $2n$  bits long,  $p$  is of length  $n + 1$ ,  $q$  is of length  $n - 1$ , and  $n \geq 150$ . Let  $E(x) = x^3 \bmod N$ , and let  $d$  denote the private decryption exponent. In this problem we will demonstrate that almost half the bits of  $d$  are easy to recover, without access to any secret information. Since in general  $ed = 1 \bmod \varphi(N)$ , we have in our case  $3d = 1 \bmod \varphi(N)$ . Therefore there is an integer  $A$  such that  $3d - A\varphi(N) = 1$ .

- 1) Prove that  $A = 2$ .  
 2) Show how to efficiently find an integer  $\hat{d}$  satisfying  $|d - \hat{d}| < \sqrt{N}$ .  
 3) Give a convincing argument (*not* a formal proof) why with high probability  $d$  and  $\hat{d}$  have the same  $(n/2) - 5$  most significant bits. The probability here is over the choices of the random primes  $p$  and  $q$ .

**Problem D: 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  $y \in \mathbb{Z}_{pq}^*$ ,  $\mathcal{A}$  produces one of the following: If  $y$  is not a quadratic residue, then  $\mathcal{A}$  outputs the text ‘‘go fetch an Agama stellio for yourself’’ (see below). If  $y = x^2$  is a quadratic residue,  $\mathcal{A}$  outputs *one* square root of  $y$ .

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.



### Problem E: Pollard’s $\rho$ Algorithm

Write a short MAPLE code that implements Pollard’s  $\rho$  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 – use colon,  $:$ , to suppress printing in commands. 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.

### Problem F: Implementing RSA

In this problem we will implement an instance of the RSA cryptosystem using MAPLE. 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.

**3)** Team up with another working group. Get hold of their  $N$  and  $e$ . Encrypt your message from part (2) under these  $N$  and  $e$ . Send the result to that group and receive a similarly encrypted message to you. Print the message you received, its decryption and its decoding as text. Print the name or names of the people you cooperated with, their public key, and the encrypted message you sent them.