

# DM49 - 2. Obligatoriske Opgave

Jacob Christiansen, moffe42, 130282  
Thomas Nordahl Pedersen, nordahl, 270282

1/4-05

## Indhold

<b>1 Opgave 1 - Public Exponent</b>	<b>2</b>
1.1 a . . . . .	2
1.2 b . . . . .	2
1.3 c . . . . .	3
<b>2 Opgave 2 - Block size</b>	<b>4</b>
2.1 a . . . . .	4
2.2 b . . . . .	4
<b>3 Opgave 3 - Primitive Element</b>	<b>5</b>
<b>4 Opgave 4 - Common Factor</b>	<b>6</b>
<b>5 Opgave 5 - Common modulus Protocol failure</b>	<b>7</b>
5.1 a . . . . .	7
5.2 b . . . . .	8
<b>A Source Code</b>	<b>9</b>
A.1 inverse.java . . . . .	9

# 1 Opgave 1 - Public Exponent

With RSA, there are often recommendations to use a public exponent  $e = 3$ .

## 1.1 a

What would the advantage to this be?

If the public exponent is chosen to be 3, then it is quite easy and fast to calculate the private exponent. Page 170 states that the running time for calculating the inverse, is  $k^3$  where  $k$  is the number of bits. Since 3 only uses 2 bits the calculation is quite fast.

## 1.2 b

If  $e = 3$ , the two prime factors dividing the modulus,  $p$  and  $q$ , must be such that  $p \equiv q \equiv 2 \pmod{3}$ . Why is it impossible to have one or both of  $p$  and  $q$  congruent to 0 or 1 modulo 3?

$\mathbf{p}$	$\mathbf{q}$	$\phi(n) = (p-1)(q-1)$	$\mathit{gcd}(\phi(n), 3)$
0	0	$(3r-1)(3s-1) = 3(3rs+r+s)+1$	1
0	1	$(3r-1)(3s) = 3(3rs+s)$	3
0	2	$(3r-1)(3s+1) = 3(3rs+r-s-1)+2$	1
1	1	$(3r)(3s) = 3(3rs)$	3
1	2	$(3r)(3s+1) = 3(3rs+r)$	3
2	2	$(3r+1)(3s+1) = 3(3rs+r+s)+1$	1

It is a requirement that the  $\mathit{gcd}(\phi(n), 3) = 1$ , because otherwise the decryption exponent can not be found.

If  $p \equiv 0 \pmod{3}$  and/or  $q \equiv 0 \pmod{3}$  then the one or both primes are divisible by 3, and the only way this can be true is if the two primes are both 3, which is not good, because the two primes has to be large.

In the case where  $p$  and  $q$  are both congruent to 1 (zero) modulo 3, the gcd is 3, which means that the inverse modulo  $\phi(n)$  can not be found.

If one of the factors is congruent to 1 and the other is congruent to 2 (mod 3), the gcd is also 3. And again the inverse can not be found.

If both the factors are congruent to 2 (mod 3), the gcd is 1 and the inverse can be found.

So a decryption exponent can only be found if the primes are congruent to 2 (mod 3) or one or both are equal to 3, which would make the system insecure.

### 1.3 c

Suppose that  $e = 3$ ,  $p = 3r + 2$  and  $q = 3s + 2$ . What would the decryption exponent  $d$  be?

The easiest way to calculate the exponent is to use algorithm 5.3 in the textbook, which calculate the multiplicative inverse.

We used the algorithm by hand to calculate  $3^{-1} \pmod{\phi(n)}$  and got the following:

$\phi(n) = 3(3rs + r + s) + 1$ . We use  $x = 3rs + r + s$  for convenience.

$$a_0 = 3x + 1, b_0 = 3, q = x, r = 1, t_0 = 0, t = 1$$

$$temp = -x \pmod{3x + 1} = 2x + 1, t_0 = 1, t = 2x + 1, a_0 = 3, b_0 = 1, q = 3, r = 0$$

Then the algorithm returns the inverse (the decryption exponent):

$$t = 2x + 1 = 2(3rs + r + s) + 1$$

## 2 Opgave 2 - Block size

Suppose that user A wants to send a message  $s \in \{s_1, s_2, \dots, s_k\}$  to user B, where  $s_i < 1024$  for  $1 \leq i \leq k$ . Assume that RSA is secure (when the modulus is large enough and is the product of two equal length prime factors).

### 2.1 a

Why would you still advise user A not to use RSA directly?

Say that someone (Oscar) intercepts the encrypted message  $y_i = s_i^e$ . If O finds out that  $s_i$  is less than 1024, he can find  $s_i$  by encrypting all values between 0 and 1023 and comparing them to the value  $y_i$ . This can be done in  $O(1024 \cdot \log(e) \cdot 10^2) = O(\log(e))$  time.

### 2.2 b

What would you recommend instead, if you still wanted to use RSA?

The blocks encrypted and send to user B should be as large as possible. The system would be more secure if the blocks  $\{s_1, s_2, \dots, s_k\}$  were concatenated into as large blocks as possible before encrypted.

If user A wishes to use RSA and only send the small blocks one at a time, this can be done by choosing random bits put in front or behind the block when encrypted, assuming that user B knows which bits are to be used.

### 3 Opgave 3 - Primitive Element

Find a primitive element (generator) in the multiplicative group modulo  $103(\mathbb{Z}_{103}^*)$  and show that it is a primitive element.

We use theorem 5.8 to find a primitive element.

$103-1$  is easily divided into prime factors.

$$103 - 1 = 102 = 2 \cdot 3 \cdot 17$$

We first try 2 as the primitive element, but in the first try it is seen that 2 is not a primitive element. 3 is also ruled out after the second try. We then skip forward to 5 and see that 5 holds for all the prime factors of 102. So 5 is a primitive element in the multiplicative group modulo 103.

$$2^{\frac{102}{2}} = 2^{51} \equiv 1 \pmod{103}$$

$$3^{\frac{102}{2}} = 3^{51} \equiv 102 \pmod{103}$$

$$3^{\frac{102}{3}} = 3^{34} \equiv 1 \pmod{103}$$

$$5^{\frac{102}{2}} = 5^{51} \equiv 102 \pmod{103}$$

$$5^{\frac{102}{3}} = 5^{34} \equiv 56 \pmod{103}$$

$$5^{\frac{102}{17}} = 5^6 \equiv 72 \pmod{103}$$

## 4 Opgave 4 - Common Factor

Suppose we have a set of blocks encoded with the RSA algorithm and we do not have the private key. Assume  $(n = pq, e)$  is the public key. Suppose also that someone tells us they know one of the plaintext blocks has a common factor with  $n$ . Does this help us find the plaintext used to produce these blocks? If so, how? If not, why not?

The fact that one of the blocks have a common factor with  $n$  gives us that  $n$  and the block has a common divisor. Then we can assume that if the gcd of  $n$  and a block is larger than 1, then the gcd is also a factor of  $n$ . This means that we can factor  $n$  in the time it takes to find the gcd of  $n$  and each of the blocks. If  $n$  is a number represented by 1024 bits, then the gcd of  $n$  and a block can be found in  $O(1024^2)$  (page 169). So the decryption exponent can easily be computed and the plaintext found.

## 5 Opgave 5 - Common modulus Protocol failure

In this exercise we are to illustrate the "common modulus protocol failure". If the same plaintext are to be send to two different recipients who both uses RSA encryption with the same  $n$ , but different  $e$  and  $\gcd(e_1, e_2) = 1$ , then it is possible to decode the message. The opponent has to receive both ciphertexts. If he is able to do that, then he can use algorithm 5.16 p. 223 i the textbook to decipher and thereby read the plaintext.

### 5.1 a

Prove that the value  $x_1$  computed in Algorithm 5.16 is in fact Alice's plaintext,  $x_2$ .

We have to prove that:

$$x_2 = y_1^{C_1} (y_2^{C_2})^{-1} \pmod{n}$$

Prof:

$$\begin{aligned} y_1^{C_1} &\equiv (x_1^{b_1})^{C_1} \equiv x_1^{b_1 C_1} \pmod{n} \\ y_2^{C_2} &\equiv (x_1^{b_2})^{C_2} \equiv x_1^{b_2 C_2} \pmod{n} \\ (y_2^{C_2})^{-1} &\equiv ((x_1^{b_2})^{C_2})^{-1} \equiv x_1^{-b_2 C_2} \pmod{n} \end{aligned}$$

$$\begin{aligned} x_2 &= y_1^{C_1} (y_2^{C_2})^{-1} \equiv x_1^{b_1 C_1} \cdot x_1^{-b_2 C_2} \equiv x_1^{b_1 C_1 - b_2 C_2} \pmod{n} \\ b_1 C_1 &\equiv 1 \pmod{b_2} \\ b_2 C_2 &= C_1 b_1 - 1 = 0 \\ b_1 C_1 - b_2 C_2 &= 1 \\ x_2 &= x_1^{b_1 C_1 - b_2 C_2} \equiv x_1^1 \pmod{n} = x_1 \square \end{aligned}$$

## 5.2 b

Illustrate the attack by computing  $x$  by method if:

$$n = 18721, b_1 = 43, b_2 = 7717, y_1 = 12677, y_2 = 14702$$

$b_1^{-1} \pmod{b_2}$  is calculated by the program Invers, listed in appendix A.

$$C_1 = b_1^{-1} \pmod{b_2} = 2692$$

$$C_2 = (2692 \cdot 43 - 1) / 7717 = 15$$

$$x_1 = 12677^{2692} \cdot (14702^{15})^{-1} \pmod{18721} = 15001$$

## A Source Code

### A.1 inverse.java

The program is implemented using Algorithm 5.3

```
public class inverse {

    public static void main( String[] args ) {
        int a = Integer.parseInt(args[0]);
        int a0 = Integer.parseInt(args[0]);
        int b0 = Integer.parseInt(args[1]);
        int t0 = 0;
        int t = 1;
        int q = a0/b0;
        int r = a0 - q*b0;
        int temp;

        while( r>0 ) {
            temp = (t0 - q*t) % a;
            t0 = t;
            t = temp;
            a0 = b0;
            b0 = r;
            q = a0/b0;
            r = a0 - q*b0;
        }
        if( r!=0 )
            System.out.println("No inverse.");
        else
            System.out.println("Invers is: " + t);
    }
}
```