CRYPTOGRAPHY #02

Supportive Algorithms

Jacek Tchórzewski, jacek.tchorzewski@pk.edu.pl

1. Euclidean Algorithm

Developed in Greece in 300 B.C. by ancient mathematician Euclid. This algorithm allows to find GCD of two numbers in an efficient way.

Example: find GCD(22, 59)

1) 59 = 2 * 22 + 15
2) 22 = 1 * 15 + 7
3) 15 = 2 * 7 + 1 ---------- the result is **1**, because the next rest is equal to 0
4) 7 = 7 * 1 + 0

As You can see above, the final result is the last non-zero rest, thus GCD(22, 59) = 1. However, we should perform algorithm until the rest is equal to 0, because we need to know which rest is the last non-zero rest.

We can assume that each iteration can be represented as:

$$b = \left\lfloor \frac{b}{a} \right\rfloor * a + r$$

And algorithm looks as follows:

0) r = b mod a
1) if *r* = 0 break. The result is *r* from the previous iteration.
2) b = a
3) a = r
4) r = b mod a
5) go to the step 1)

2. Extended Euclidean Algorithm

Extended Euclidean Algorithm may be used to represent GCD of two numbers as a combination of this numbers:

$$GCD(a, b) = x * a + y * b$$

Where x and y are integers. From cryptographical point of view, it is important, because it allows to find an inverse modulo value. Inverse modulo can be represented as:

$$x \equiv a^{-1} \, mod \, \mathrm{b}$$

Equation above can be represented also as: $x * a \equiv 1 \, mod \, b$ and further as $x * a + y * b = 1$. As You can see, inverse modulo value can be found only when GCD(a, b) = 1. Before I will show You pseudocode, let's look at the example from mathematical point of view. So, let's assume that we want to find $22^{-1} mod \, 59$. We know from the previous example that GCD(22, 59) = 1, thus inverse modulo exist. We can write: $x * 22 + y * 59 = 1$ and start to solve it in the following way:

Classic Algorithm	Extended Algorithm
(solving for the rests)	(backwards subsitutions)
1) 15 = 59 – 2 * 22	1 = 15 – 2 * 7
2) 7 = 22 – 1*15	1 = 15 – 2 * (22 – 15) = 15 – 2*22 + 2 * 15 = 3 * 15 – 2 * 22
3) 1 = 15 – 2 * 7	1 = 3 * (59 – 2 * 22) – 2 * 22 = 3 * 59 – 8 * 22

Thus we know that 1 = 3 * 59 – 8 * 22 = (-8) * 22 + 3 * 59. *x* = –8 and *y* = 3. The value which we are looking for is *x*. *x* also should be greater than 0. Thus we can write: *x = -8 + 59 = 51*. The inverse modulo of our example is equal to 51. As you can see, it may be hard to implement this algorithm in such way, beacuse to perform first step of the Extended Algorithm we need values from the last step of the Classic Algorithm. If we assume that the only value which we need is *x*, we can rewrite algorithm in a simpler, interative way.

Input parameters: $x_0 = 0, x_1 = 1, q_i = \left\lfloor \frac{b_i}{a_i} \right\rfloor$

1) 59 = 2 * 22 + 15	$x_0 = 0$
2) 22 = 1 * 15 + 7	$x_1 = 1$

Next x_i values are calculated accordingly to the formula:

$x_i = (x_{i-2} - [q_{i-2} * x_{i-1}]) \, mod \, b$

3) 15 = 2 * 7 + 1	$x_2 = (0 – [2 * 1]) \, mod \, 59 = –2 \, mod \, 59 = 57$
4) 7 = 7 * 1 + 0	$x_3 = (1 – [1 * 57]) \, mod \, 59 = –56 \, mod \, 59 = 3$
5) ----------	$x_4 = (57 – [2 * 3]) \, mod \, 59 = 51$

Result is equal to 51. Note, that in comparison with a classic algorithm, we need to calcuate *x* one more time after the rest is equal to 0. The algorithm above is simpler implementation of Extended Euclidean Algorithm and is dedicated only for finding modular inverse.

3. Fermat's Little Theorem

Theorem state that if *b* is a prime number, than for any value *a*, $a^b - a$ is a multiple of *b*:

$$a^b \equiv a \; mod \; b$$

Thus we can write:

$$a^b \equiv a \; mod \; b \quad | :a$$
$$a^{b-1} \equiv 1 \; mod \; b \quad | :a$$
$$a^{b-2} \equiv a^{-1} \; mod \; b$$

59 is a prime number. So we may write: $22^{59-2} \equiv 22^{-1} \; mod \; 59$, and further: $22^{59-2} = 22^{57}$ = 32968015588397012383304605752905171719650002960101406309072097007825019994112 mod 59 = 51.

4. Binary Powering Method

Powering big numbers is common in cryptography. Naive methods appears to be useless (too time consuming) when it comes to practice usage. Binary powering allows to spped this proces up drastically and works in quite simple way. Example:

Naive method: $5^{13} = 5*5*5*5*5*5*5*5*5*5*5*5*5$ (12 multiplications)

Binary method: $5^{13} = 5^{(1+4+8)} = 5 * 5^4 * 5^8$ (2 multiplications)

Note that $(13)_{10} = (1101)_2$. Thus: $5^{13} = 5^{(1101)_2} = 5^{2^0+2^2+2^3}$

Algorithm of binary multiplication of a^b can be presented as follows:

Notation and input params:

base = a,

result = 1,

LSB(*b*) – Least Significant Bit of *b*,

1) If LSB(b) = 1 do step 2), otherwise skip it

2) result = result * base
3) base = base * base
4) b = b >> 1
5) if b = 0 return result, otherwise go to step 1)

Project 2

Exercise 1:

Implement Euclidean Algorithm and calculate: GCD(59, 22), GCD(15, 7), GCD(101, 49) and GCD(123, 66).

Exercise 2:

Implement Extended Euclidean Algorithm and try to calculate: $22^{-1} mod\ 59$, $7^{-1} mod\ 15$, $49^{-1} mod\ 101$ and $66^{-1} mod\ 123$. Is it possible to calculate all values?

Exercise 3:

Implment Fermat Little Theorem and try to calculate the same modular inverses as in Exercise 2. Is it possible to calcualte all values?

Exercise 4:

Implment Binary Multiplication Algorithm and solve: 5^{10}, 3^{13}, 16^{5}, and 2^{17}.