Notes on Modular Arithmetic

Leonardo Tamiano

December 16, 2022

This document was prepared for the *Computer Network Security* (CNS) course offered during the year 2022-2023 by the Tor Vergata university of Rome. The document contains some notes regarding certain algorithms which are related to modular arithmetic and which are needed to understand the computational processes used by the RSA cryptographic system.

In particular, two algorithms will be presented and discussed.

1. The first is the **square and multiply** algorithm, used to compute efficiently numbers of the form $x^y \mod N$, that is, powers of numbers working in modular arithmetic.
2. The second algorithm is the **extended euclidean algorithm**, which is used to compute identities of the form $a \cdot x + b \cdot y = \text{GCD}(a, b)$, which are also known as *Bézout's identity*. These identities are needed to compute the inverse of a number when working with modular arithmetic.

Contents

1 Efficient Modular Exponentiation with Square and Multiply

Consider the problem of computing powers of numbers when working with a certain modulo N. That is, we want to compute the value

$$x^y \mod N$$

Given that when working modulo N we have to reduce our numbers by taking the remainders when divided by N, and that exponentiation is simply repeated multiplication, we could approach our problem by simply multiplying x by itself y items, and, after each multiplication, before the next one is performed we reduce everything modulo N.

$$\underbrace{x \mod N \to x^2 \mod N \to ... \to x^y \mod N}_{y \text{ multiplications}}$$

For example, suppose $x = 5$, $y = 6$ and $N = 3$. To compute $5^6 \mod 3$ using the previous method we'd need to compute the following numbers

$$\begin{aligned}
5^1 &\mod 3 = 2 \\
5^2 &\mod 3 = 2 \cdot 5 \mod 3 = 1 \\
5^3 &\mod 3 = 1 \cdot 5 \mod 3 = 2 \\
5^4 &\mod 3 = 2 \cdot 5 \mod 3 = 1 \\
5^5 &\mod 3 = 1 \cdot 5 \mod 3 = 2 \\
5^6 &\mod 3 = 2 \cdot 5 \mod 3 = 1
\end{aligned}$$

At the end, after 6 multiplications we have found our answer, that $5^6 \mod 3 = 1$. It is not hard to see why this approach works, as it is a pretty intuitive approach to take. Yet, we're not only interested in correctness, we're also interested in performance. We can then ask ourselves: Is this approach also an efficient one?

A quick analysis reveals that no, this approach is not really efficient. Given that each multiplication requires roughly $O(n^2)$ operations, where n is the maximum bit-length of the numbers we're trying to multiply, and that in this approach we have to do y multiplications, where $y \leq 2^n$, this naive method has a computation complexity of $O(2^n \cdot n^2)$, which, as we can see, is exponential with respect to the input size. This is a huge problem, especially when we consider that in RSA cryptography we must do these sorts of computations with numbers that are at least 1024 bit long.

Can we do it better? Can we compute $x^y \mod N$ without doing y multiplications?

Yes, we can! A better approach to the problem of modular exponentiation is given to us by the algorithmic *Divide et Impera* strategy. The basic idea is, instead of computing all the powers of x, going from x^1 to x^y, we can compute squares of x, that is, powers of x of the form x^{2^i}, such as

$$x^{2^0} = x^1 \ , \ x^{2^1} = x^2 \ , \ x^{2^2} = x^4 \ , \ 2^{2^3} = x^8 \ \ \ldots$$

Notice that we can have at most $\lfloor \log_2 y \rfloor$ of such powers, given that if $i > \lfloor \log_2 y \rfloor$, then $x^{2^i} > x^y$ and therefore we do not need to compute it. This is why this second approach is, as we will shortly see, much faster than the previous one: the old approach required y multiplications, while the new one only requires $\log_2 y$ multiplications. Once we have computed all those powers, we can combine them together to obtain our final answer. To combine these powers we consider the binary representation of y and only use the powers that correspond to a 1 digit.

To make this last point more clear, let us compute once again the value $5^6 \mod 3$ using this new and more efficient algorithm. The first thing we do is we compute the binary representation of 6

$$6 = (\texttt{110})_2 = 0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 = 2 + 4 = 6$$

Then we compute the following powers of 5

$$\begin{aligned} 5^1 &\mod 3 = 5 \mod 3 = 2 \\ 5^2 &\mod 3 = 2 \cdot 2 \mod 3 = 1 \\ 5^4 &\mod 3 = 1 \cdot 1 \mod 3 = 1 \end{aligned}$$

Finally, we combine such powers of 5 to obtain our final answer

$$\begin{aligned} 5^6 \mod 3 &= 5^{(2^1+2^2)} \mod 3 \\ &= (5^2 \mod 3) \cdot (5^4 \mod 3) \\ &= 1 \cdot 1 \mod 3 \\ &= 1 \end{aligned}$$

We obtained the same answer as before, but instead of doing 6 multiplications we only needed 3 multiplications! And this small difference between the two approaches is only a consequence of the fact that these numbers are small. If we needed to compute $5^{1024} \mod 3$, then the first approach would've required 1024 multiplications, while the second approach only requires around 10 multiplication!

We can now understand why this new method works and why it is more efficient than the previous one. What we're doing, essentially, is writing the exponent in a different way. Instead of writing it in base 10, we're writing it in base 2, and we're using various properties of exponents to transform one notation into the other.

$$5^6 = 5^{(0\cdot 2^0 + 1\cdot 2^1 + 1\cdot 2^2)} = 5^0 \cdot 5^2 \cdot 5^4 = 1 \cdot 5^2 \cdot 5^4 = 5^2 \cdot 5^4$$

So far we have seen an example in which we had to write out the entire binary representation of the number before going to the next step. If we're in a rush there is also another similar way of computing $x^y \mod N$ efficiently that does not require to write out explicitly the binary representation of the exponent. The method is based on the following identity

$$x^y = \begin{cases} x^{\lfloor \frac{y}{2} \rfloor} \cdot x^{\lfloor \frac{y}{2} \rfloor} & , \quad y \text{ is even} \\ x^{\lfloor \frac{y}{2} \rfloor} \cdot x^{\lfloor \frac{y}{2} \rfloor} \cdot x & , \quad y \text{ is odd} \end{cases}$$

As we can see, the idea is to start from knowing the value of $x^{\lfloor \frac{y}{2} \rfloor}$ and using this knowledge to compute x^y. This same identity also works in the modular contest. Here follows follows a python implementation of the previous idea, which can be used to compute $x^y \mod n$. In the code we check the base case $x^0 \mod n = 1$. Then, if we're not in the base case, we compute $x^{\lfloor \frac{y}{2} \rfloor} \mod n$ and then we combine this knowledge to finally compute $x^y \mod n$.

```
def fast_exp(x, y, n):
    if y == 0:
        return 1
    r = fast_exp(x, int(y/2), n)
    if y % 2 == 0:
        return r*r % n
    else:
        return r*r*x % n
```

While in code this second approach is more efficient, when writing it by hand I prefer the first one, the one in which the binary represention of the exponent is written explicitly. Still, it's important to point out that these two slightly different ways of computing powers are based on the same idea: computing only powers of x of the form x^{2^i}. The only difference is that the second method is more efficient, while the first one, at least in my opinion, is more clear.

1.1 Examples

Let us now see some examples. In all these examples I use the first described method.

1.1.1 Example 1

Let $x = 5$, $y = 37$ and $N = 13$.

1. First we compute the binary representation of 37

$$(100101)_2 = 2^0 + 2^2 + 2^5 = 1 + 4 + 32 = 37$$

2. Then we compute $5^i \mod N$ where i is a power of 2 that is less than 37.

$$\begin{aligned}
5^1 &\mod 13 = 5 \mod 13 = 5 \\
5^2 &\mod 13 = 5 \cdot 5 \mod 13 = 12 \\
5^4 &\mod 13 = 12 \cdot 12 \mod 13 = 1 \\
5^8 &\mod 13 = 1 \cdot 1 \mod 13 = 1 \\
5^{16} &\mod 13 = 1 \cdot 1 \mod 13 = 1 \\
5^{32} &\mod 13 = 1 \cdot 1 \mod 13 = 1
\end{aligned}$$

3. Finally, we combine the values by multiplying only the powers corresponding to a 1 digit in the binary representation of 37.

$$\begin{aligned}
5^{37} \mod 13 &= 5^{(2^0+2^2+2^5)} \mod 13 \\
&= (5^1 \mod 13) \cdot (5^4 \mod 13) \cdot (5^{32} \mod 13) \\
&= 5 \cdot 1 \cdot 1 \mod 13 \\
&= 5
\end{aligned}$$

1.1.2 Example 2

Let $x = 4$, $y = 28$ and $N = 17$.

1. First we compute the binary representation of 28

$$(11100)_2 = 2^2 + 2^3 + 2^4 = 28$$

2. Then we compute $4^i \mod 17$ where i is a power of 2 that is less than 28.

$$\begin{aligned}
4^1 &\mod 17 = 4 \mod 17 = 4 \\
4^2 &\mod 17 = 4 \cdot 4 \mod 17 = 16 \\
4^4 &\mod 17 = 16 \cdot 16 \mod 17 = 1 \\
4^8 &\mod 17 = 1 \cdot 1 \mod 17 = 1 \\
4^{16} &\mod 17 = 1 \cdot 1 \mod 17 = 1
\end{aligned}$$

3. Finally, we combine the values by multiplying only the powers corresponding to a 1 digit in the binary representation of 28.

$$\begin{aligned}
4^{28} \mod 17 &= 4^{(2^2+2^3+2^4)} \mod 17 \\
&= (4^4 \mod 17) \cdot (4^8 \mod 17) \cdot (4^{16} \mod 17) \\
&= 1 \cdot 1 \cdot 1 \mod 17 \\
&= 1
\end{aligned}$$

1.2 Exercises

Solve the following exercises using the method discussed in the previous examples. You can check if the answer you got is a valid answer by using the `square_and_multiply.py` script available in the code resources of the lecture [1]. The script must be executed in the following way

```
python3 square_and_multiply.py 6553 89999 20013
 Computed value 6553^89999 MOD 20013 = 6322
```

1.2.1 Exercise 1

Let $x = 11$, $y = 103$ and $N = 143$. Compute the value $x^y \mod N$ using the square and multiply algorithm.

1.2.2 Exercise 2

Let $x = 15$, $y = 150$ and $N = 203$. Compute the value $x^y \mod N$ using the square and multiply algorithm.

[1] `https://teaching.leonardotamiano.xyz/class-material/cns/modular_arithmetic/code.zip`

1.2.3 Exercise 3

Let $x = 4$, $y = 1024$ and $N = 37$. Compute the value $x^y \mod N$ using the square and multiply algorithm.

2 Modular Inversion with Extended Euclidean Algorithm

Let us now consider another problem. Let $a, b \in \mathbb{Z}$ be two arbitrary integers, and let $d = \text{GCD}(a, b)$. What we want to find are two other integers, $x, y \in \mathbb{Z}$, such that

$$a \cdot x + b \cdot y = d = \text{GCD}(a, b)$$

Before describing the solution, let us first understand why exactly we are interested in such problem. Being able to solve the problem we just posed is crucial if we want to compute inverses in modular arithmetic. Indeed, consider the slightly different problem of wanting to compute the inverse of a certain $a \in \mathbb{Z}_n$. That is, we want to solve the following equation

$$x \equiv a^{-1} \mod n$$

It can be proved that the previous equation has a solution if and only if $\text{GCD}(a, n) = 1$. We will not prove this fact, but, assuming this is the case, notice how we can rewrite the equation we started with as follows

$$\begin{aligned}
x \equiv a^{-1} \mod n &\iff a \cdot x \equiv 1 \mod n \\
&\iff \exists x \in \mathbb{Z} : \ a \cdot x - 1 \text{ divides } n \\
&\iff \exists x, k \in \mathbb{Z} : \ a \cdot x - 1 = n \cdot k \\
&\iff \exists x, k \in \mathbb{Z} : \ a \cdot x + n \cdot (-k) = 1 \\
&\iff \exists x, k \in \mathbb{Z} : \ a \cdot x + n \cdot (-k) = 1 = \text{GCD}(a, n)
\end{aligned}$$

All of this to say that the inverse of a modulus n is the coefficient x that multiplies the a in the identity

$$a \cdot x + n \cdot (-k) = 1 = \text{GCD}(a, n)$$

If we're able to compute such expressions, also known as *Bézout's identities*, then we will also be able to compute inverses when working with modular arithmetic.

So, with respect to the actual algorithm, we will start with the traditional euclidean algorithm, which can be used anytime we need to compute the *GCD*, also known as the *Greatest Common Divisor*, between two numbers a and b, and then we will extend it in order to compute *Bézout's identities*.

2.1 Euclidean Algorithm

GCD stands for *Greatest Common Divisor*, and, as the name suggests, it is the highest number that divides both a and b.

For example, if $a = 60$ and $b = 10$, then $\text{GCD}(a, b) = \text{GCD}(60, 10) = 10$, because there are no numbers greater than 10 that divide both 10 and 60. One can easily see this by computing the factorization of both numbers and picking only the factors that appear in both numbers.

$$\begin{cases} 60 = 2^2 \cdot 3^1 \cdot 5^1 \\ 10 = 2^1 \cdot 5^1 \end{cases} \implies \text{GCD}(60, 10) = 2^1 \cdot 5^1 = 10$$

This factorization can be done for any numbers, the problem is that factorizing a number is an expensive process, computationally speaking. The euclidean algorithm offers us a smart way to compute such GCD without having to factorize the two numbers. The algorithm is based on the following fact, taken from classical *number theory*.

Given a, b, the GCD of a and b is equal to the GCD of b and $a \mod b$.

We will not prove mathematically this result, but consider the previous example, with $a = 60$ and $b = 10$. It is indeed true that

$$\begin{aligned} \text{GCD}(60, 10) &= \text{GCD}(10, 60 \mod 10) \\ &= \text{GCD}(10, 0) \\ &= 10 \end{aligned}$$

The cool thing about this fact is that it allows us to compute the GCD of two numbers by computing the GCD of two other numbers which are either equal to or smaller than the previous numbers. This is because $a \mod b < b$.

We now present a python implementation of the standard euclidean algorithm

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

The algorithm works because at each step the GCD of the numbers remains unchanged, while the numbers keep decreasing and decreasing until one of them reaches 0, at which point the GCD is the other remaining number. This idea is implemented in the script `standard_gcd.py`, which you can use as follows

```
python3 standard_gcd.py 50 5
 Computed value GCD(50, 5) = 5
```

2.1.1 Example 1

Let $a = 1337$ and $b = 175$. Let us compute the $\text{GCD}(a, b)$ using the traditional Euclidean algorithm.

$$\begin{aligned} \text{GCD}(1337, 175) &= \text{GCD}(175, 1337 \mod 175) = \text{GCD}(175, 112) \\ &= \text{GCD}(112, 175 \mod 112) = \text{GCD}(112, 63) \\ &= \text{GCD}(63, 112 \mod 63) = \text{GCD}(63, 49) \\ &= \text{GCD}(49, 63 \mod 49) = \text{GCD}(49, 14) \\ &= \text{GCD}(14, 49 \mod 14) = \text{GCD}(14, 7) \\ &= \text{GCD}(7, 14 \mod 7) = \text{GCD}(7, 0) \\ &= 7 \end{aligned}$$

2.1.2 Example 2

Let $a = 42$ and $b = 3150$. Let us compute the $\text{GCD}(a, b)$ using the traditional Euclidean algorithm.

$$\begin{aligned}\text{GCD}(42, 3150) &= \text{GCD}(3150, 42 \mod 3150) = \text{GCD}(3150, 42) \\ &= \text{GCD}(42, 3150 \mod 42) = \text{GCD}(42, 0) \\ &= 42\end{aligned}$$

2.2 Extended Euclidean Algorithm

The algorithm we just showed is not enough to solve our particular problem. Remember that we are not particular interested in knowing $\text{GCD}(a, b)$. We are interested in knowing the two coefficients (x, y) such that

$$a \cdot x + b \cdot y = \text{GCD}(a, b)$$

The idea is to start from the basic euclidean algorithm and extend it to construct such x and y. We will develop a **recursive algorithm** following the structure of the traditional euclidean algorithm. As with any recursive algorithm, we will need to define what to do in our base case as well as in the general recursive case.

- (**base case**): As a base case, consider when $b = 0$. This is the final step of the traditional euclidean algorithm, and it allows us to directly infer that $\text{GCD}(a, b) = \text{GCD}(a, 0) = a$. With respect to our new problem, we can easily find our identity by writing

  $$a \cdot 1 + b \cdot 0 = a = \text{GCD}(a, b)$$

  As we can see, the (x, y) coefficients for the *Bézout identity* between a and 0 are given by $x = 1$ and $y = 0$.

- (**recursive case**): Consider now the general recursive case, and assume we have found two coefficients, namely (x_1, y_1) for the *Bézout identity* between b and $a \mod b$. That is, we know

  $$b \cdot x_1 + (a \mod b) \cdot y_1 = \text{GCD}(b, a \mod b) = \text{GCD}(a, b)$$

  can we also find two other coefficients, namely (x, y) for the *Bézout identity* between a and b? By knowing that

  $$a \mod b = a - b \cdot q \quad , \text{ where } q = \lfloor \frac{a}{b} \rfloor$$

  we can rewrite the previous identity in terms of b and a as follows

  $$\begin{aligned}b \cdot x_1 + (a \mod b) \cdot y_1 &= b \cdot x_1 + (a - b \cdot q) \cdot y_1 \\ &= b \cdot x_1 + a \cdot y_1 - b \cdot q \cdot y_1 \\ &= b \cdot (x_1 - q \cdot y_1) + a \cdot y_1\end{aligned}$$

  Putting everything together we have that

$$a \cdot y_1 + b \cdot \left(x_1 - \lfloor \frac{a}{b} \rfloor \cdot y_1 \right) = \text{GCD}(b, a \mod b) = \text{GCD}(a, b)$$

We are set! Indeed, we have found our coefficients, which are

$$\begin{cases} x & := x_y \\ y & := x_1 - \lfloor \frac{a}{b} \rfloor \cdot y_1 \end{cases}$$

By defining (x, y) as shown above, the following identity becomes true

$$a \cdot x + b \cdot y = \text{GCD}(a, b)$$

Here it follows a python implementation of the extended euclidean algorithm

```
def extended_gcd(a, b):
    if b == 0:
        return 1, 0, a
    else:
        x1, y1, d = extended_gcd(b, a % b)
        return y1, x1 - y1*int((a/b)), d
```

Let us now see some numerical example to clear the ideas regarding this algorithm.

2.2.1 Example 1

Let $a = 1337$ and $b = 175$. In the previous section we showed that $\text{GCD}(1337, 175) = 7$. Let us now find x and y such that

$$1337 \cdot x + 175 \cdot y = 7 = \text{GCD}(1337, 175)$$

To start off we will divide 1337 by 175 and we will write both the quotient q_1 as well as the remainder r_1.

$$1337 = 175 \cdot 7 + 112$$

As we can see, $q_1 = 7$ and $r_1 = 112$. Then we keep going, but this time we will divide 175 by 112, which was the previous remainder, to obtain the second quotient q_2 and the second remainder r_2.

$$175 = 112 \cdot 1 + 63$$

This process keeps going until we reach the final step where we have a remainder of 0. Let's write all of these equations one after the other.

$$\begin{aligned} 1337 = 175 \cdot 7 + 112 &\quad , \quad q_1 = 7, r_1 = 112 \\ 175 = 112 \cdot 1 + 63 &\quad , \quad q_2 = 1, r_2 = 63 \\ 112 = 63 \cdot 1 + 49 &\quad , \quad q_3 = 1, r_3 = 49 \\ 63 = 49 \cdot 1 + 14 &\quad , \quad q_4 = 1, r_4 = 14 \\ 49 = 14 \cdot 3 + 7 &\quad , \quad q_5 = 3, r_5 = 7 \\ 14 = 7 \cdot 2 + 0 &\quad , \quad q_6 = 2, r_6 = 0 \end{aligned}$$

Notice how we stop then $r_6 = 0$. We will now begin to rewrite every equation written before starting from the one in which $r_5 = 7$. Notice how we skip the last equation. We will rewrite every equation in terms of the remainder as is shown below

$$\begin{aligned}
49 = 14 \cdot 3 + 7 &\iff 7 = 49 + 14 \cdot (-3) \\
63 = 49 \cdot 1 + 14 &\iff 14 = 64 + 49 \cdot (-1) \\
112 = 63 \cdot 1 + 49 &\iff 49 = 112 + 63 \cdot (-1) \\
175 = 112 \cdot 1 + 63 &\iff 63 = 175 + 112 \cdot (-1) \\
1337 = 175 \cdot 7 + 112 &\iff 112 = 1337 + 175 \cdot (-7)
\end{aligned}$$

Keep in mind the particular way in which we have re-written the equations. This will now be useful, because now we start from the top equation, which is $7 = 49 + 14 \cdot (-3)$, and we substitute the value for 14 to obtain

$$\begin{aligned}
7 = 49 + 14 \cdot (-3) &= 49 + 63 \cdot (-3) + 49 \cdot (3) \\
&= 49 \cdot (4) + 63 \cdot (-3)
\end{aligned}$$

Notice how have rewritten 7 but this time using 49 and 63 instead of 49 and 14. The idea is to keep going until we rewrite 7 using the initial $a = 1337$ and $b = 175$.

$$\begin{aligned}
7 = 49 + 14 \cdot (-3) &= 49 + 63 \cdot (-3) + 49 \cdot (3) \quad \text{(substitute 14)} \\
&= 49 \cdot (4) + 63 \cdot (-3) \\
&= 112 \cdot (4) + 63 \cdot (-4) + 63 \cdot (-3) \quad \text{(substitute 49)} \\
&= 112 \cdot (4) + 63 \cdot (-7) \\
&= 112 \cdot (4) + 175 \cdot (-7) + 112 \cdot (7) \quad \text{(substitute 63)} \\
&= 112 \cdot (11) + 175 \cdot (-7) \\
&= 1337 \cdot (11) + 175 \cdot (-77) + 175 \cdot (-7) \quad \text{(substitute 112)} \\
&= 1337 \cdot (11) + 175 \cdot (-84)
\end{aligned}$$

In the last equation we were finally able to express 7 using 1337 and 175. In particular, if $x = 11$ and $y = -84$ the we have what we wanted from the start, which is

$$1337 \cdot x + 175 \cdot y = 7 = \text{GCD}(1337, 175)$$

2.2.2 Example 2

Let $a = 42$ and $b = 3150$. In the previous section we showed that $\text{GCD}(42, 3150) = 42$. Let us now find x and y such that

$$42 \cdot x + 3150 \cdot y = 42 = \text{GCD}(42, 3150)$$

First, we keep dividing the numbers following the basic euclidean algorithm as we did in the previous example until we reach a remainder of 0.

$$\begin{aligned}
42 &= 3150 \cdot 0 + 42 \\
3150 &= 42 \cdot 75 + 0
\end{aligned}$$

In this case we don't have to do much work, because we can simply invert the first equation to get

$$42 = 3150 \cdot 0 + 42 \iff 42 \cdot 1 + 3150 \cdot 0 = 42 = \text{GCD}(42, 3150)$$

Therefore, our identity is given by $x = 1$ and $y = 0$.

2.3 Exercises

2.3.1 Exercise 1

Let $a = 1362$ and $b = 251$. Compute $\text{GCD}(a, b)$ and find the two coefficients $x, y \in \mathbb{Z}$ such that

$$1362 \cdot x + 251 \cdot y = \text{GCD}(1362, 251)$$

2.3.2 Exercise 2

Let $a = 1444$ and $b = 762$. Compute $\text{GCD}(a, b)$ and find the two coefficients $x, y \in \mathbb{Z}$ such thatq

$$1444 \cdot x + 762 \cdot y = \text{GCD}(1444, 762)$$

2.3.3 Exercise 3

Let $a = 256$ and $b = 900$. Compute $\text{GCD}(a, b)$ and find the two coefficients $x, y \in \mathbb{Z}$ such thatq

$$256 \cdot x + 900 \cdot y = \text{GCD}(256, 900)$$

3 RSA Exercises

We are now ready to see how these two algorithms are used in the context of RSA. As a small remainder, we remember that in RSA keys are generated through the following process:

- Two big (> 1024 bit) and distant primes p and q are choosen.
- The values N and $\Phi(N)$ are computed as

$$N = p \cdot q$$
$$\Phi(N) = (p - 1) \cdot (q - 1)$$

- A value $e < \Phi(N)$ is choosen such that $\text{GCD}(e, \Phi(N)) = 1$
- We compute the d as the inverse of e modulus $\phi(N)$

$$d \equiv e^{-1} \mod \Phi(N)$$

After the keys have been generated we have that (e, N) is the public key, and can therefore be shared with everyone, while d is the private key and must be kept secret. When we want to encrypt something we first transform that something into a number $m \in [0, N)$. For example if we want to encrypt a text we could use the underlying bytes of the text to get the number m. After we

have the number m, which is our plaintext, to actually encrypt it we use modular exponentiation to compute the ciphertext

$$c = m^e \mod N$$

Everyone can encrypt messages, as (e, N) is the public key. To decrypt an encrypted message $c \in [0, N)$ we proceed once again with modular exponentiation

$$m = c^d \mod N$$

Only the owner of the private key d can decrypt messages.

3.1 Exercise 1

Consider the following RSA parameters

$$\begin{cases} P &= 31 \\ Q &= 47 \\ N &= P \cdot Q = 1457 \\ e &= 13 \\ d &= 637 \end{cases}$$

Let $m = 1337$ be a plaintext message. First, compute the encryption of m as $c = m^e \mod N$ using the square and multiply algorithm. Then, after having computed c, compute the value $c^d \mod N$, and make sure that

$$c^d \mod N = m \mod N$$

Solution: First we compute $c = m^e \mod N$, with $M = 1337$, $e = 13$ and $N = 1457$. Following the square and multiply intuition, we write out the binary representation of the exponent

$$13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0 = (\texttt{1101})_2$$

Then we compute all the powers $1337^i \mod 1457$ where i is a power of 2 that is less than 8. In particular we get

$$\begin{aligned} 1337^1 &\mod 1457 = 1337 \\ 1337^2 &\mod 1457 = 1337 \cdot 1337 \mod 1457 = 1287 \\ 1337^4 &\mod 1457 = 1287 \cdot 1287 \mod 1457 = 1217 \\ 1337^8 &\mod 1457 = 1217 \cdot 1217 \mod 1457 = 777 \end{aligned}$$

Finally, we combine the computed values according to the binary representation of the exponent

$$\begin{aligned} 1337^{13} \mod 1457 &= 1337^{(2^0+2^2+2^3)} \mod 1457 \\ &= (1337^1 \mod 1457) \cdot (1337^4 \mod 1457) \cdot (1337^8 \mod 1457) \\ &= 1337 \cdot 1217 \cdot 777 \mod 1457 \\ &= 994 \end{aligned}$$

Thus we can say that $c = m^e \mod N = 994$. Now we have to compute $c^d \mod N$. We proceed as we did previously, just that now the base is $c = 994$ and the exponent is $d = 637$ while the modulus is the same $N = 1457$. First we write out the binary representation of the exponent

$$637 = 512 + 64 + 32 + 16 + 8 + 4 + 1 = 2^9 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0 = (\texttt{1001111101})_2$$

Then we compute all the powers $994^i \mod 1457$ where i is a power of 2 that is less than 637. In particular we get

$$\begin{aligned}
994^1 &\mod 1457 = 994 \\
994^2 &\mod 1457 = 994 \cdot 994 \mod 1457 = 190 \\
994^4 &\mod 1457 = 190 \cdot 190 \mod 1457 = 1132 \\
994^8 &\mod 1457 = 1132 \cdot 1132 \mod 1457 = 721 \\
994^{16} &\mod 1457 = 721 \cdot 721 \mod 1457 = 1149 \\
994^{32} &\mod 1457 = 1149 \cdot 1449 \mod 1457 = 159 \\
994^{64} &\mod 1457 = 159 \cdot 159 \mod 1457 = 512 \\
994^{128} &\mod 1457 = 512 \cdot 512 \mod 1457 = 1341 \\
994^{256} &\mod 1457 = 1341 \cdot 1341 \mod 1457 = 343 \\
994^{512} &\mod 1457 = 343 \cdot 343 \mod 1457 = 1089
\end{aligned}$$

Finally, we combine the computed values according to the binary representation of the exponent

$$\begin{aligned}
994^{13} \mod 1457 &= 994^{(2^9+2^6+2^5+2^4+2^3+2^2+2^0)} \mod 1457 \\
&= (994^1 \mod 1457) \cdot (994^4 \mod 1457) \cdot (994^8 \mod 1457) \cdot (994^{16} \mod 1457) \\
&\quad \cdot (994^{32} \mod 1457) \cdot (994^{64} \mod 1457) \cdot (994^{512} \mod 1457) \\
&= 994 \cdot 1132 \cdot 721 \cdot 1149 \cdot 159 \cdot 512 \cdot 1089 \mod 1457 \\
&= 1337
\end{aligned}$$

As we can see, we have that $c^d \mod N = 1337$, which is also the value of m. This means that with the given parameters (e, d, N) we have that

$$c^d \mod N = (m^e)^d \mod N = m^{e \cdot d} \mod N = m$$

■

3.2 Exercise 2

Consider the following RSA parameters

$$\begin{cases} P &= 233 \\ Q &= 257 \\ N &= P \cdot Q = 59881 \\ e &= 3 \end{cases}$$

Find the values $d, k \in \mathbb{Z}$ such that

$$e \cdot d + N \cdot k = 1$$

What is the private key of this RSA configuration? Why is that?

Solution: To obtain the private key we need to execute the extended euclidean algorithm on the inputs $(e, \Phi(N))$. Given that $\Phi(N) = 256 \cdot 232 = 59392$ we find the following execution.

First we keep dividing until we compute the GCD between e and $\Phi(N)$.

$$\begin{aligned} 59392 &= 7 \cdot 8484 + 4 \\ 7 &= 4 \cdot 1 + 3 \\ 4 &= 3 \cdot 1 + 1 \\ 3 &= 1 \cdot 3 + 0 \end{aligned}$$

Then, we start from the second to last equation, and we write the various remainders in terms of the other numbers.

$$\begin{aligned} 1 &= 4 \cdot (1) + 3 \cdot (-1) \\ 3 &= 7 \cdot (1) + 4 \cdot (-1) \\ 4 &= 59392 \cdot (1) + 7 \cdot (-8484) \end{aligned}$$

And then we substitute, in order to express 1 in terms of $e = 7$ and $\Phi(N) = 59392$.

$$\begin{aligned} 1 &= 4 \cdot (1) + 3 \cdot (-1) \\ &= 4 \cdot (1) + 7 \cdot (-1) + 4 \cdot (1) \quad \text{(substitute 3)} \\ &= 4 \cdot (2) + 7 \cdot (-1) \\ &= 59392 \cdot (2) + 7 \cdot (-16968) + 7 \cdot (-1) \\ &= 59392 \cdot (2) + 7 \cdot (-16969) \end{aligned}$$

Having finished, we can see that the coefficient x that multiplies the $e = 7$ is $x = -16969$. If we want to express it as a positive number modulo $\Phi(N)$ we simply do $d = \Phi(N) - x = 42423$. Our private key is then d. This can also be checked by verifying that, for example

$$1337^{7 \cdot 42423} \mod 59881 = 1337$$

■

3.3 Exercise 3

Consider the following RSA public key $N = 143, e = 7$ and let $c = 115$ be a ciphertext that you managed to capture. You know that c was encrypted using the public key (N, e). That is, you know there exists an m such that

$$c = m^e \mod N$$

find such m.

Solution: To decrypt the ciphertext first we need to recover the key. In this exercise the key can be easily recovered because N is the product of really small primes.

So, first we factorize N by trying out small common factors

$$\begin{aligned}
&\frac{N}{2} = 71.5 \ , \ \frac{N}{3} = 47.666666666666664 \\
&\vdots \\
&\frac{N}{9} = 15.88888888888889 \ , \ \frac{N}{10} = 14.3 \\
&\frac{N}{11} = 13
\end{aligned}$$

As we can see, the value 11 perfectly divides N. This means that $N = 11 \cdot 13$. Now that we have found the factors of N we can compute the private key d, first by computing the value of $\Phi(N) = (P-1) \cdot (Q-1) = 10 \cdot 12 = 120$, and then we find the coefficient x such that

$$e \cdot x + \Phi(N) \cdot k = 1 \iff 7 \cdot x + 120 \cdot k = 1$$

That is, we want to execute the Extended Euclidean Algorithm on the starting values $(7, 120)$. As we remember, we first set up a series of euclidean division to extract the GCD of those two numbers. This time however we don't have many equations, since

$$\begin{aligned}
120 &= 7 \cdot 17 + 1 \\
7 &= 1 \cdot 7 + 0
\end{aligned}$$

Then we start from the second to last equation and we express the GCD computed, which is 1, in terms of $e = 7$ and $\Phi(N) = 120$

$$120 = 7 \cdot 17 + 1 \iff 120 \cdot (1) + 7 \cdot (-17) = 1$$

This means that the coefficient that multiplies $e = 7$ is $x = -17$. If we want the positive version modulo $\Phi(N)$ we can simply do $\Phi(N) - 17 = 103$. Therefore we can say $d = 103$. That is, the private key is $d = 103$. Notice indeed that

$$e \cdot d \mod \Phi(N) = 7 \cdot 103 \mod \Phi(N) = 1$$

Now, having recovered the private key, we can use it to recover the plaintext by computing $c^d \mod N$, wich $c = 115$, $d = 103$ and $N = 143$. We can use the square and multiply algorithm to do so. First we write out the binary representation of the exponent

$$103 = 2^6 + 2^5 + 2^2 + 2^1 + 2^0 = 64 + 32 + 4 + 2 + 1 = (\texttt{1100111})_2$$

Then we compute all the powers of the form $c^i \mod N$, where i is a power of 2 less than 103.

$$
\begin{aligned}
115^{1} \mod 143 &= 115 \\
115^{2} \mod 143 &= 115 \cdot 115 \mod 143 = 69 \\
115^{4} \mod 143 &= 69 \cdot 69 \mod 143 = 42 \\
115^{8} \mod 143 &= 42 \cdot 42 \mod 143 = 48 \\
115^{16} \mod 143 &= 48 \cdot 48 \mod 143 = 16 \\
115^{32} \mod 143 &= 16 \cdot 16 \mod 143 = 113 \\
115^{64} \mod 143 &= 113 \cdot 113 \mod 143 = 42
\end{aligned}
$$

And finally we put multiply together the values that we need

$$
\begin{aligned}
115^{103} \mod 143 &= 115^{(2^6+2^5+2^2+2^1+2^0)} \mod 143 \\
&= (115^{64} \mod 143) \cdot (115^{32} \mod 143) \cdot (115^{4} \mod 143) \cdot \\
&\quad (115^{2} \mod 143) \cdot (115^{1} \mod 143) \\
&= 42 \cdot 113 \cdot 42 \cdot 69 \cdot 115 \mod 143 \\
&= 15
\end{aligned}
$$

This means that the decrypted message, that is, the plaintext, is $m = 15$!

■