RSA Cryptosystem

The RSA cryptosystem is a example of a "public key" system. This means that everyone can know the encryption key, but it is computationally infeasible for an unauthorized person to deduce the corresponding decryption key.

In the case of RSA, here is how it works. Alice makes known two numbers, N and e which she has selected carefully. Then Bob can use these numbers to encode a message and send it to Alice. A third party Oscar has free access to N, e, and the encoded message. It should be essentially impossible for Oscar to decode the message, but Alice can decode the message easily because she knows a secret.

At the center of the RSA cryptosystem is the RSA modulus N. It is a positive integer which equals the product of two distinct prime numbers p and q:

$$\boxed{\text{RSA modulus:} \qquad N = pq.}$$

So $55 = 5 \cdot 11$, $119 = 7 \cdot 17$, and $10403 = 101 \cdot 103$ could each be used as an RSA modulus, although in practice, one would use much larger numbers for better security, to be explained below.

Also needed is an encoding exponent e. The only requirement on e is that

$$\gcd(e, (p-1)(q-1)) = 1 \tag{1}$$

Typically, e is choosen first, and then Alice picks p and q so that equation (1) holds.

Most cryptosystems used in the ASU CryptoRally use a standard method for converting the initial message to numbers, and then the real encoding happens with numbers. The same is true of RSA, but the transition between text and numbers is more complicated. Here we will focus on encoding/decoding numbers in the set $\{0, \ldots, N-1\}$.

One bit of notation before we begin: if a and b are non-negative integers, we will write $a \bmod b$ for the remainder when a is divided by b. It is the unique integer r such that $0 \le r < b$ and $b - r$ is a multiple of a.

To encode a message m, Bob computes the remainder when m^e is divided by N:

$$\boxed{\text{Encoding:} \qquad M = m^e \bmod N.}$$

This can be done efficiently by the square-and-multiply method described below.

To decode the message M, Alice uses the values p and q. After picking N and e, she computes d by:

$$\boxed{\text{Decoding exponent: } d = e^{-1} \bmod (p-1)(q-1).}$$

This inverse is the same as is used in the Affine and Hill ciphers, and it can be computed efficiently by the extended Euclidean Algorithm. Alice then decodes the message by computing

$$\boxed{\text{Decoding:} \qquad m = M^d \bmod N.}$$

We now see Alice's secret: she knows p and q. In other words, she knows how to factor N. In practice, she starts with two large primes p and q and multiplies them together to get N. For RSA to be secure, N has to be hard to factor or else Oscar can determine p and q, in which case he can also compute d and decode messages.

Here is an example with small numbers. Suppose the encoding exponent is $e = 17$, and Alice chooses $p = 5$ and $q = 11$ so $N = 5 \cdot 11 = 55$. Note, $\gcd(e, (p-1)(q-1)) = \gcd(17, 4 \cdot 10) = 1$. Now suppose Bob wants to encode the "message"

$m = 37$. He computes

$$M = 37^{17} \bmod 55 = 27$$

Alice also computes $d = e^{-1} \bmod (5-1)(11-1) = 17^{-1} \bmod 40 = 33$. This can be checked by computing

$$ed \bmod (p-1)(q-1) = 17 \cdot 33 \bmod 4 \cdot 10 = 561 \bmod 40 = 1.$$

Then she can decode the message:

$$M^d \bmod N = 27^{33} \bmod 55 = 37.$$

Square and Multiply

Suppose a, e, and N are positive integers and we want to compute $a^e \bmod N$, i.e., the remainder when a^e is computed modulo N. The simplest approach would be to compute a^e by repeated multiplications, and then take remainders. This can be made more efficient.

The first issue is that a^e might be too large for the memory of a calculator or computer, but the final answer could be small enough to fit in the calculator's memory. The second is that we want to do the computation with a minimal amount of work.

The first issue can be resolved by using congruences. Here we will do the exponentiation naively, by repeated multiplications. Suppose we want to compute $12345^5 \bmod 54321$. If we do the computation directly, we would get

$$12345^5 = 28671833852463546 5625$$

which is quite large. On the other hand, we can replace values with their remainders for division by 54321 at every intermediate step:

$$\begin{aligned}
12345^2 &= 152399025 \equiv 28620 \pmod{54321} \\
12345^3 &= 12345 \cdot 12345^2 \equiv 12345 \cdot 28620 \equiv 10116 \pmod{54321} \\
12345^4 &= 12345 \cdot 12345^3 \equiv 12345 \cdot 10116 \equiv 52362 \pmod{54321} \\
12345^5 &= 12345 \cdot 12345^4 \equiv 12345 \cdot 52362 \equiv 43311 \pmod{54321}
\end{aligned}$$

Since 43311 is congruent to our desired number modulo 54321 and is in the allowed range for remainders on division by 54321 (namely, in $\{0, 1, \ldots, 54320\}$), it is the remainder.

When working modulo N, in each multiplcation along the way we only have to deal with inputs from $\{0, 1, \ldots, N-1\}$. So, the biggest product we have to compute (before taking the next remainder) is less than N^2. In the computation above, that means our calculator needs to be able to hold integers up to $54321^2 = 2950771041$, a 10 digit number whereas 12345^5 is a 21 digit number. The difference is even more dramatic with bigger exponents.

The second improvement makes computations much faster for large exponents. Suppose we want to compute $12345^{21} \bmod 54321$. If we used repeated multiplications, that means 20 multiplications. Instead, we start by doing repeated squarings:

$$\begin{aligned}
12345^2 &\equiv 28620 \pmod{54321} \\
12345^4 \equiv (12345^2)^2 &\equiv 28620^2 \equiv 52362 \pmod{54321} \\
12345^8 \equiv (12345^4)^2 &\equiv 52362^2 \equiv 35211 \pmod{54321} \\
12345^{16} \equiv (12345^8)^2 &\equiv 35211^2 \equiv 46338 \pmod{54321}
\end{aligned}$$

Now we use these values to get the final result:

$$12345^{21} = 12345^1 \cdot 12345^4 \cdot 12345^{16} = 12345 \cdot 52362 \cdot 46338 \equiv 1452 \pmod{54321}$$

We used 4 multiplications in the squaring phase, and 2 more multiplications in the last line, so a total of 6 multiplications, instead of 20.

The key to this method is in the last step where we used that $21 = 1 + 4 + 16 = 2^0 + 2^2 + 2^4$. We wrote our exponent as a sum of distinct powers of 2. You can do this for any positive integer, it is equivalent to finding the binary (or base 2) expansion of the number.