Supplemental file

This material is not necessary to be able to finish the assignment. It is here only for those interested in how things work and to explain how some smart ideas allow encryption and an efficient method to make it happen.

Encryption part 1. A shortcut to calculate A^{exp} mod n

The formula for raising a number to a large exponent is based on a binary representation of the exponent. Let's use 27 as an example.

$27 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$
$27 = 1 * 16 + 1 * 8 + 0 * 4 + 1 * 2 + 1 * 1$
$27 = 16 + 8 + 2 + 1$

Then $A^{27} = A^{16} * A^8 * A^2 * A \bmod n$
By using A = X * Y mod n = ((X mod n) * (Y mod n)) mod n, we get
$A^{27} = ((A^{16} \bmod n) * (A^8 \bmod n) * (A^2 \bmod n) * (A \bmod n)) \bmod n$

So, for the calculations we start with A (the base), then calculate A^2 then $(A^2)^2 = A^4$, $(A^4)^2 = A^8$ and so on. We actually calculate all these numbers mod n which makes them more manageable. Notice that for the final product we only multiply the ones that actually contribute even though we calculate more. In our example we calculate A^4 but we don't use it.

Try to identify in the pseudocode or in your code where/when this happens.

Encryption part 2. Theory behind RSA encryption

To understand encryption note that the whole system is based on finding n, key1 and key2 so that $(A^{key1})^{key2} = A \bmod n$

key1 is usually trivial: the prime number 2**8 + 1 or something similar.

Encryption uses the fact that for prime numbers p1 and p2, we can get $n = p_1 * p_2$ so that $A^{tot(n) / gcd(p1 - 1, p2 - 1)} = 1 \bmod n$, with tot(n) being the Euler's tot function. This is an old result in number theory.

The totient function for $n = p_1 * p_2$ is simple to calculate and is $tot(p_1 * p_2) = (p_1-1) * (p_2-1)$. However, any multiple M of p_1-1 and p_2-1 will make $A^M = 1 \bmod n$. The minimum is found by dividing $(p_1-1) * (p_2-1)$ by $gcd(p_1-1, p_2-1)$, which eliminates the common factors while yielding a multiple of both.

Taking m as $tot(n) / gcd(p_1-1, p_2-1)$ we can proceed to find two keys with the expected requirement above. Notice that by Euler's result and how we defined m, $A^m = 1 \bmod n$

By using an extended Euclidean algorithm for calculating the gcd, it is possible to find coefficients k_1 and k_2 for
$p_3 * k_1 + m * k_2 = 1$ since $gcd(p_3, m) = 1$
(See the wikipedia page for the extended Euclidean algorithm.)

Moreover taking any k_3,
$exp = p_3 * (k_1 + k_3 * m) + m * k_2$ yields
$exp = p_3 * k_1 + m * k_2 + p_3 * k_3 * m$ or
$exp = 1 + p_3 * k_3 * m$
Defining f as p3 * k3, we get
$exp = 1 + f * m = 1 + m * f$

With this result let's explore what happens when we define private_key as $k_1 + k_3 * m$.
$exp = p_3 * private\_key + m * k_2 = 1 + m * f$

Now, this is where the interesting part happens and it is very simple algebra after this point.
$A^{p3 * private\_key + m * k2} = A^{1 + m * f} \bmod n$ which is
$A^{p3 * private\_key + m * k2} = A \bmod n * A^{m * f} \bmod n$
So by the definition of m on both the left and the right side,
$A^{p3 * private\_key} * A^{m * k2} = A \bmod n * 1 \bmod n$
$A^{p3 * private\_key} = A \bmod n$

Hence we are left with $A^{p3 * private\_key}$ which is A as we desired when we started. So we get
$(A^{p3})^{private\_key} = A \bmod n$

Defining A as the message, p_3 as the public key and the private key as before we have
$C = A^{public\_key}$ which is a cipher that can be transmitted without protection
$M = C^{private\_key} = (A^{public\_key})^{private\_key} \bmod n$ which is the decrypted message since $(A^{p3})^{private\_key} = A \bmod n$.
So we get $M = C^{private\_key} = A$ which is the original message.

It turns out that all the steps for encryption are simple. However, the steps for figuring out the private key is extremely hard and time consuming.