COMP 4958: Assignment 1

The purpose of this assignment is to experiment with the RSA algorithm.

The RSA algorithm requires the generation of a pair of keys – a public key and a private key, each in turn is a pair of integers.

The keys are generated by first choosing a pair of large prime numbers p and q and calculating their product n. n is then one component of the public key and of the private key.

Let l be the LCM (least common multiple) of $p - 1$ and $q - 1$. Choose a number e (with $1 < e < l$) that is relatively prime to l, i.e., the GCD (greatest common divisor) of e and l is 1. The pair (e, n) is then the public key. (According to Wikipedia, a very common value for e is 65537.)

To generate the private key, we need to find d, the multiplicative inverse of e modulo l. This means we need to solve $d \times e \equiv 1 \pmod{l}$ for d. (We'll say more about how to calculate d below.) The pair (d, n) is then the private key.

If a message (the plaintext) is represented by an integer M. The encrypted message (the ciphertext) C is given by $M^e \bmod n$. Given the ciphertext C, we can recover the plaintext by calculating $C^d \bmod n$.

See

`https://en.wikipedia.org/wiki/RSA_(cryptosystem)#Key_generation`

to learn more about key generation.

For the mathematical calculations, you'll need to implement the following three functions (where a, b, m and n are all positive integers)

- `powm(a, n, m)` that returns the remainder of a^n when divided by m.

  For this function, the problem with first finding a^n and then calculating its remainder is that a^n can be quite large. To keep the numbers relatively small, we can use the fact that, for a, b and m all positive integers, `rem(a * b, m)` and `rem(rem(a, m) * rem(b, m), m)` are equal.

  For example, `rem(53 * 24, 7)` is 5, `rem(53, 7) * rem(24, 7)` is 4 * 3 = 12 which also gives a remainder of 5 when divided by 7.

  Secondly, the brute force way of multiplying n copies of a together to obtain a^n involves a lot of multiplications when n is large. To reduce the number of multiplications, we can do something similar to this example: 2^{100} is $(2^{50})^2$, 2^{50} is $(2^{25})^2$, 2^{25} is $2 \times (2^{12})^2$, ...

- `lcm(a, b)` that returns the LCM of a and b.

  Note that the LCM of a and b can be calculated as the quotient of $a \times b$ by the GCD of a and b and the GCD can easily be calculated using the Euclidean algorithm. See

  `https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations`

- `inverse(a, n)` that returns the multiplicative inverse of a modulo n.

  The algorithm in "pseudo-code" to find `inverse(a, n)` is as follows:

```
/* https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm#Modular_integers */
function inverse(a, n)
    t := 0;     newt := 1
    r := n;     newr := a

    while newr != 0 do
        quotient := r div newr
        (t, newt) := (newt, t  quotient  newt)
        (r, newr) := (newr, r  quotient  newr)

    if r > 1 then
        return "a is not invertible"
    if t < 0 then
        t := t + n

    return t
```

Use a tail-recursive helper function in your implementation.

In order to encypt plaintext and decrypt ciphertext, we need to convert between text and integers. The PKCS #1 specification calls for 2 primitives:

- I2OSP (Integer-to-Octet-String Primitive): converts a non-negative integer into an octet string (a sequence of bytes) of a specified length.
- OS2IP (Octet-String-to-Integer Primitive): converts an octet string to a nonnegative integer.

I2OSP performs the conversion by basically writing the integer in base-256 & uses the digits as the octet string. For example,

$$310400273487 = 72 \times 256^4 + 69 \times 256^3 + 76 \times 256^2 + 76 \times 256 + 79$$

Hence the octet string, when written in Elixir bitstring syntax, is `<<72, 69, 76, 76, 79>>`. This corresponds to the string `"HELLO"`. The OS2IP function takes the octet string & converts it back to an integer.

Note that I2OSP also makes sure the octet string has a certain length either by padding or rejecting an octet string that is too long. (It takes 2 arguments; See `https://tools.ietf.org/html/rfc8017#section-4` if you want to know the details.) However, for this assignment, we will omit this length requirement, i.e., the resulting octet string can be any length. Implement two Elixir functions `integer_to_binary` & `binary_to_integer` corresponding to I2OSP & OS2IP. For example

- `integer_to_binary(310400273487)` returns `<<72, 69, 76, 76, 79>>`
- `binary_to_integer(<<72,69,76,76,79>>)` (or `binary_to_integer("HELLO")`) returns 310400273487

You will also need to implement functions to encrypt & decrypt "text". However, the logic for the two operations are essentially the same except that we use different input text & keys. We can implement just one function: `crypt(text, {k, n})` where {k, n} is the key. If k is the e mentioned above, we are using the public key to encrypt `text` (which is a binary, i.e., an octet string). If k is d, then we are using the private key to decrypt `text`. Note that `crypt` returns a binary.

Put your functions in a module named `A1`. As a test, you will be provided values for p, q, e and a ciphertext. You are asked to decrypt that ciphertext.

Due date & submission details will be announced in class.