Assignment #2 - Blockchain Cryptography

You will perform some basic RSA cryptographic work. You will start with creating public and private keys, and then encrypt, decrypt, sign, and verify messages.

As deliverable, you will have to submit just your R script. Open Rstudio, go to File → New File → R-Script, and save the new script as HW2_LASTNAME_FIRSTNAME.r . Make sure that in the script you add print commands to display your answers.

This is an individual assignment. For many of you, this is your first time using R. Please review the two Intro to R documents on Canvas. If you struggle to complete the assignment, please ask for help to your colleagues, the TA, or me.

Part 0: How to use R with big numbers

To complete the assignment, you will have to install and load the package "openssl". To do so, you need to add the following commands at the beginning of your script :

```
install.packages("openssl")
library("openssl")
```

In R, numbers defined as *int* or *double* types store small numbers with finite precision. This is enough for many applications, but cryptographic methods require arithmetic on much larger numbers and without loss of precision. OpenSSL provides a bignum data type which holds arbitrary sized integers. Each time you define a number, you will have to define it as bignum type. So instead of typing x = 1234567890, you will have to type x=bignum("1234567890"). I will provide more guidance later on.

Part 1: Create Public and Private Keys

You need to create a public key (n and e)and a private key (d) using R.

Public Keys

The first step is to choose large prime numbers p and q. These numbers should stay private. Anyone with these two numbers can easily compute the private key. For the purpose of this exercise, use the following prime numbers:

```
p=bignum("112481050639317229656723018120659623829736571015511322021617837187076258724819")
q=bignum("89185111938335771293328323333111422985697062149139368049232365065924632677343")
```

Your first public key is $n = p * q$. The security of RSA encryption relies on the fact that given n, it is very hard to figure out what p and q are. This is called factorization. The second public key is another prime number e, smaller than n. Use this one: e=bignum("65537")

Now that you have n and e, you can broadcast these public keys n and e to anyone who wants to send you a message.

Private Key

The second step is to compute your private key. You need to find a number d so that $(d * e - 1)$ is a multiple of $(p - 1)(q - 1)$, or in other words, find d so that $d * e \mod (p - 1)(q - 1) = 1$. For small p and q, this can be done with a simple while loop. However, for big numbers like the one in this exercise, the loop would take a very long time to run. Fortunately the extended Euclidean Algorithm helps us to speed things up. For bignum type, the function d=bignum_mod_inv(x,y) finds the smallest number d so that $(d * x) \mod y = 1$. Use the bignum_mod_inv function to find the private key d.

Part 2: Encrypt a Message

You receive your friend's public keys n and e from part 1. You want to encrypt the plaintext message "Running late. Wait for me.", and send the encrypted ciphertext message to her by text message.

First, you have to transform the message into a bignum number m. Use the charToRaw and the bignum functions to transform the text into a bignum number.

m= bignum(charToRaw("Whatever Message you want to send"))

Then you encrypt m using the public keys n and e using the formula $c = m^e \mod n$. Because we are using bignum type numbers, you should use the function bignum_mod_exp function.[1].

Now that you have the ciphertext, you can safely send it through an unsecured line. Only someone with the private key will be able to decrypt the message. The encrypted message is quite long to type in, so you might want to encode it with base64 notation for human readability, using the function base64_encode().

Part 3: Decrypt a Message

You receive the following base64 ciphertext message from your friend:

rGhkBLUmPQStyYGrhIcNxnhZw6GeGoFGswZuUihd+kPx21VtPSMmdBRQOkKw8uLPhsh0NV4qk27G/EFuVT2iAw==

He used your public keys from part 1 to encrypt the plaintext message. You want to decrypt the ciphertext.

First, you decode the base64 into a bignum number using the base64_decode() and the bignum() functions. Second, in order to decrypt the ciphertext you need to apply the formula $m = c^d \mod n$ using the bignum_mod_exp function. Finally, you convert the numeric message into text using the rawToChar() function. What does the message say?

Part 4: Sign a Message

Your friend wants to make sure that you indeed received the message. So he asks you to sign the message you just received as a receipt. Because you don't want to reveal to the word the message he just sent you, you first hash it using the SHA-256 algorithm. Please use the function sha256() to hash the message m into m_hash, and then transform the hash into a number using the charToRaw and bignum functions as in part 2.

Then proceed and sign the hashed message. To sign a message, you need to create a signature by encrypting the hashed message using your private key d. The signature can be computed using the formula $s = m\_hash^d \mod n$. Then you send along the hashed message together with the signature to your friend.

Part 5: Verify a Message

Your friend receives the hashed message m_hash, and the signature s. He wants to verify that the signature matches the hash. He can simply check that $m\_hash = s^e \mod n$. Is the signature valid?

[1]bignum_mod_exp(x,y,z) is equivalent to type $x^y \mod z$.