Programming Assignement Report

Prelude

The personnummer used to generate our assignment is : **199809077237**.

All the programs are written in Java, and we provide Makefile(s) in order to run the code. Read all the README(s) for each exercices to get additionnal information on how to use the Makefile, but it should be pretty straightforward.

0 A Math Library for Cryptography

1 Special Soundness of Fiat-Shamir sigma-protocol

2 Decrypting CBC with simple XOR

3 Attacking RSA

The same message has been encrypted using RSA to three different recipients. All recipients have the same public key ($e = 3$) but different modulus (N_1, N_2, N_3). We have intercepted the three ciphertexts. The goal is to find the initial message.

Knowing how the RSA protocol works, we have the following encryptions :

$$\begin{cases} c_1 \equiv m^e (\text{mod } N_1) \\ c_2 \equiv m^e (\text{mod } N_2) \\ c_3 \equiv m^e (\text{mod } N_3) \end{cases}$$

With the Chinese Remainder Theoremn we can find c such that the solutions to the system above are all number congruent to c modulo $N_1 N_2 N_3$.

Having found c, we know that :

$$c \equiv m^e (\text{mod } N_1 N_2 N_3) \Longleftrightarrow m \equiv \sqrt[3]{c \ (\text{mod } N_1 N_2 N_3)}$$

The last step is then to compute m with the helper function provided in the `CubeRoot.java` file.

Decoded text: Taher ElGamal

4 Attacking ElGamal

In this exercice, we are given :

- **p** : the group size
- **g** : the generator of the group
- **y** : the public key of the receiver
- **time** : the time at which the message was encrypted (and the weak technique used to choose the "randomness")
- **(c1, c2)** : the ElGamal encryption of the message

The first goal is to find the random number r used for the encryption.

We know that $c_1 = g^r$, and we are given g, c_1 and the technique to find r.

The pseudo code for the 'pseudo random number generator' is the following :

```
integer createRandomNumber:
    return YEAR*(10^10)+month*(10^8)+days*(10^6)+hours*(10^4)+minute*(10^2)+sec+millisecs;
```

We only miss the *millisecs* value.

We can then loop over all the possible value of *milliseconds* and check at each iteration if $c_1 = g^r$.

When the equality above is true, it means that we have found the right r.

Now the last goal is to find the message m.

From ElGamal encryption protocol, we know that $c_2 = my^r$.

We are given c_2, y and we just found r.

$$c_2 \equiv my^r \pmod{p} \iff m \equiv c_2(y^{-1})^r \pmod{p}$$

We can get the value of m by solving the above equation.

After decryption, we get the following message :

Decoded text: Crytanalysis doesn't break cryptosystems. Bruce Schneier breaks cryptosystems.