CHALMERS UNIVERSITY OF TECHNOLOGY
Dept of Computing Sciences

Programming assignment, Cryptography course 2021

0. Introduction

The programming assignment consists of five small subtasks; they can (and should) be solved as the corresponding topics are discussed in the lectures. For each of the assignments below, we specify which lecture that gives the necessary background to start the work on that particular subtask.
This assignment is done in groups of two students. There is only one submission deadline for this assignment, on December 14, but try to form groups early, so that you can get to work in time. You must submit the assignment in Fire as a group. One of the group members creates the group and the other joins.
You can choose to program in Java, Haskell or C++. For each subtask we provide you with a code skeleton with a main program that reads and parses the input file and calls a function that you must define.
We recommend that you organize your solutions to the programming assignment in a directory `progass` with five subdirectories for the five subtasks. Note that the skeleton code for four of the subtasks reads a file `input.txt`, which contains the assignment data for the respective subtask. For the submission, make a zip archive of the directory `progass` and submit that.

1. Attacking a weak cipher in CBC mode (After lecture 3)

In this subtask you will attack a home-brewed, very badly designed cipher:

- The cipher is based on a block cipher with block size 96 bits (12 bytes). A longer message is split into 96 bit blocks and encrypted in CBC mode, using a random IV.
- The key size for the block cipher is also 12 bytes. The encryption of a block m with key k is simply $E(k, m) = k \oplus m$.

We see immediately that E is the one time pad, and that it is misused by encrypting several blocks with the same key k.
In addition, you as attacker have access to both an encrypted message $c = c_0||c_1||\ldots||c_n$ (where c_0 is the IV) and to the first block m_1 of the corresponding plaintext, so you can do a known plaintext attack and recover the complete plaintext.
You should now visit the data generation and code skeleton page where you

- give the personnummer of one group member and get data generated for your attack. The data consists of two lines, first the 12 bytes personnummer you gave and then an encrypted message (in hexadecimal notation). The plaintext message that is encrypted has the personnummer as its first block, followed by a secret message which you must retrieve. Note that the skeleton code parses the input file and takes care of necessary conversions from the representation we just mentioned. Copy the data to `input.txt` in the subdirectory.
- download skeleton code for your chosen programming language. Read the code and make sure that you understand the name, parameter types and result type of the function you must define.

The secret message will be a sentence in English.

2. A fragment of a math library for cryptography (After lecture 5)

In this task you will implement some functions from number theory that are commonly used in cryptography. The aim is to make you familiar with the algorithms by actually implementing them. To emphasize that this a learning task, you implement the function for 64-bit integers rather than arbitrary size integers as in real cryptographic applications. You must implement the following functions:

- the extended Euclidean algorithm.
- modular inverse. Your function should return 0 if the number is not invertible.
- modular exponentiation.
- Fermat's primality test. Instead of picking random values a to test the primality of a number n, make a start from 2 and increment it by 1 at each new iteration, until you have tested all the values below $n/3$.

You download skeleton code from the same page as for the previous task. For this task, there is no personalised data. Instead, together with the skeleton code you get a test suite program that your library must pass. Make sure that your solution passes before you submit.

3. Attacking ElGamal with a weak PRG (After lecture 6)

Thanks to its random component, two ElGamal encryptions of the same message can look completely different. However, this also makes the strength of the encryption depend on the random number generation. In this assignment you will attack textbook ElGamal under a weak number generator.
We intercepted an ElGamal encryption of a message together with a partial time stamp with one-second precision on the ciphertext. We also know that the PRG used for encryption was in pseudocode

```
integer createRandomNumber:
   return YEAR*(10^10) + month*(10^8) + days*(10^6)
        + hours*(104) + minute*(10^2) + sec + millisecs;
```

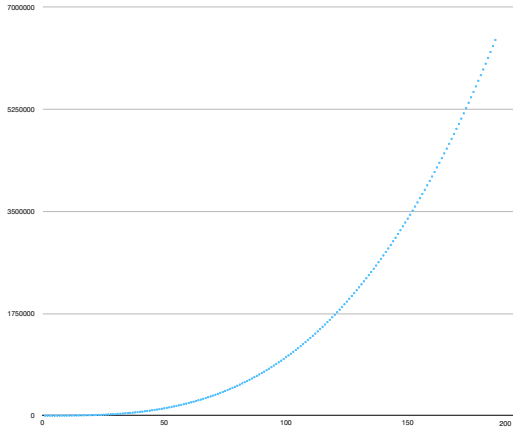
Thus, even though we cannot exactly say which random number was generated during encryption, the search space is limited. Your task is to decrypt the message exploiting this weakness of the pseudorandom generator. Assignment data and code skeleton at the same site as before. The encrypted message is again a sentence in English.
Note that p is a 1024 bit prime, so we are here working with numbers of realistic size.

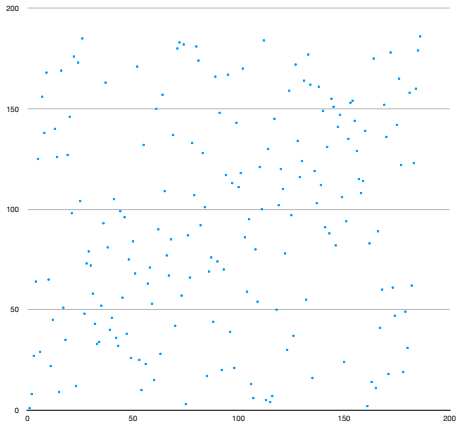
4. Attacking RSA (After lecture 8)

This attack applies to the case in which the same message m is encrypted using textbook RSA to three different recievers. The enablers of the attack are that

- all receivers have the same public key $e = 3$, and
- the receivers have different moduli (N_1, N_2, N_3), that are coprime.

Your task is to use the three eavesdropped ciphertexts and recover the secret message, which is a name.
You may recall that in lecture 6 we saw that it is dangerous to encrypt a too small message. This is illustrated as follows:





The left graph shows $f(x) = x^3$ where x is an integer, while to the right we see $g(x) = x^3 \in \mathbb{Z}_N^*$ for $N = p \cdot q = 187$. It should be clear that for the left function it is not difficult to find x for a given value $f(x)$; simple binary search is good enough. To the right, on the other hand, the graph looks completely random and no useful algorithm is obvious. In fact, no efficient algorithm for cube roots in $\mathbb{Z}_N$ is known.
So, when using $e = 3$, one must make sure that the encrypted value is $> N$. We now consider the related situation where we have eavesdropped on a message m encrypted for *three* receivers, which all use $m = 3$, so we have the ciphertexts

$$c_1 = m^3 \in \mathbb{Z}_{N_1}^*,$$
$$c_2 = m^3 \in \mathbb{Z}_{N_2}^*,$$
$$c_1 = m^3 \in \mathbb{Z}_{N_3}^*.$$

We assume that all three have taken precautions, so that m^3 as an integer is bigger than all N_i. We can now use the Chinese Remainder Theorem, first on the two first ciphertexts to get the value of $m^3 \in \mathbb{Z}_{N_1 \cdot N_2}^*$, and then once more on this value and the third ciphertext. Since $m < N_i$ for $i = 1, 2, 3$ (we can only encrypt elements of $\mathbb{Z}_{N_i}$) we get to the situation discussed in the previous paragraph and can use an integer cuberoot algorithm to recover m!

5. Attacking the Fiat-Shamir protocol (After lecture 10)

In the lecture, we noted that it is essential that the prover's commitment R is different for each protocol run. In this final subtask, you will get a protocol run where this requirement is violated. Exploit this to discover the prover's secret key!