Homework 2 - Sage Basics and Prehistoric Crypto

Cryptography and Security 2016

- You are free to use any programming language you want, although SAGE is recommended.
- Put all your answers **and only your answers** in the provided SCIPER-answers.txt file. This means you need to provide us with Q_{1a}, Q_{1b}, Q_{2a}, Q_{2b}, Q_3, Q_4 and Q_5. You can download your **personal** files on `http://lasec.epfl.ch/courses/cs16/hw2/index.php`
- The answers should all be ASCII sentences except for Exercise 1 and Exercise 2a where we expect you to give us an array of integers or an integer. **Please provide nothing else. This means, we don't want any comment and any strange character or any new line** in the .txt file.
- We also ask you to submit your **source code**. This file can of course be of any readable format (.sws is fine) and we encourage you to comment your code.
- The plaintexts of most of the exercises contain some random words. Don't be offended by them and Google them at your own risk. Note that they might be really strange.
- If you worked with some other people, please list all the names in your answer file. We remind you that you have to submit your own source code and solution.
- We might announce some typos of this homework on Moodle in the "news" forum. Everybody is subscribed to it and does receive an email as well. If you decided to ignore Moodle emails we recommend that you check the forum regularly.
- The homework is due on Moodle on **Tuesday the 18th of October** at 22h00.

Exercise 1 Order of Element

In your parameter file, you will find the prime numbers p, p_{11}, p_{12}, p_{13}, where $p = 2p_{11}p_{12}p_{13} + 1$ and the list of numbers $X = [x_{11}, ..., x_{19}]$. Find the generator(s) in X in the multiplicative group $\mathbf{Z}_p^*$. Then, write the total number of generators in Q_{1a} and each of the generators in Q_{1b}, sorted in increasing order (use `.sort()` method of Sage's lists!).

Exercise 2 Character Mappings

In this exercise **and in two other exercises of this homework**, we are going to use the following mapping to encode the standard 26 lowercase letter alphabet plus the space symbol and the . symbol to the integers. The letter 'a' is mapped to 0, 'b' to 1, . . . , 'z' to 25 and ' ' to 26 and '.' to 27. Write a function that maps an element from the alphabet to the corresponding integer and one function that given an integer returns the corresponding element from the alphabet. Using the first function, encode *word*$_2$ which you can find in your personal parameter file to obtain an array of integers that you should write under Q_{2a} in your answer file. Using the second function, decode the array *encoded*$_2$ and write the solution under Q_{2b}.

Hint: Have a look at the python functions "ord()", "chr()", and "String.join()".

Exercise 3 ElGamal Decryption

ElGamal encryption is a public-key encryption scheme which is widely used in cryptography. The encryption scheme is as follows:

- *Key Generation:* First, we generate public parameters (g_3, n_3) where $\langle g_3 \rangle$ is a group of order n_3 generated by g_3. Second, we pick the secret key $x_3 \in \mathbf{Z}_{n_3}$. Lastly, we have the public key $y_3 = g_3^{x_3}$.
- *Encryption:* Given a message $m_3 \in \langle g_3 \rangle$, we pick a random $r_3 \in \mathbf{Z}_{n_3}$ and compute the ciphertext (u_3, v_3) where $u_3 = g_3^{r_3}$ and $v_3 = m_3 y_3^{r_3}$.
- *Decryption:* We compute $m_3 = v_3 u_3^{-x_3}$.

In this exercise, we encrypted a secret message Q_3. In the encryption process, in order to be able to encrypt ASCII strings, we use the following encoding scheme. First, the message Q_3 is converted into ASCII where each character is written with three decimal digits and all the digits are concatenated. A "1" is then put in front of the string to obtain an integer. For instance, the string "Red fox" is mapped to "1 082 101 100 032 102 111 120". So, "Red fox" gives the integer "1082101100032102111120" (which can then be used as an ElGamal plaintext). By applying this encoding to Q_3, we obtain m_3. We choose the parameters n_3 and g_3 such that $n_3 + 1$ is a prime number and g_3 is the generator of the (multiplicative) group $\mathbf{Z}^*_{n_3+1}$. In this way the mapping for an integer x (obtained after the encoding) is straight forward: x will map to the element $x \in \mathbf{Z}^*_{n_3+1}$.

You have an ElGamal ciphertext (u_3, v_3) encrypted by the public-key $y_3 = g_3^{x_3}$, the secret key x_3 and public parameters g_3 and n_3 in your parameter file. Decrypt the ciphertext (u_3, v_3) to obtain m_3, decode it to obtain the plaintext Q_3 and put it in your answer file.
Remark: this is not a secure instantiation of the ElGamal cryptosystem! The main goal of the exercise is for you to practice Sage.

Exercise 4 The Adventures of the Crypto-Apprentice: Reduced Enigma

A young apprentice cryptographer just enrolled in the course Cryptography and Security. Eager to get some hands-on experience, he refused to wait until the end of the course and wanted to apply some cryptography straight away. The young apprentice was deeply impressed by the Enigma cryptosystem. He decided that he would try it out, and use it to encrypt a message Q_5 containing a secret password required to enter the lounge of the Crypt-Nerd Club (a secret organization he started when he started the course).

In this exercise, we will be working with the same alphabet as in Exercise 2. We recall how Enigma works. All components of Enigma can be mathematically described as permutations over the set $\{0, 1, \ldots, 28\}$. To initialize Enigma, we have to pick three rotor permutations α, β, γ, a plug board permutation σ and an initial rotation a. These three things together comprise the secret key of enigma (so we have $K = ((\alpha, \beta, \gamma), \sigma, a))$. To encrypt a message $m = m_1 m_2 \ldots m_r$, where $m_i \in \{a, \ldots, z, \ , .\}$, we first encode it with the encoding from Exercise 2 to get the sequence of integers $x_1, x_2, \ldots, x_r = \mathsf{Encode}(m) = x$. We then compute the ciphertext $y = y_1, y_2, \ldots, y_r$ with Enigma, one integer at a time:

$$y_j = \sigma^{-1} \circ \alpha_{i_1}^{-1} \circ \beta_{i_2}^{-1} \circ \gamma_{i_3}^{-1} \circ \pi \circ \gamma_{i_3} \circ \beta_{i_2} \circ \alpha_{i_1} \circ \sigma(x_j),$$

where we compute $i = a + j$ and $i_1 = i \bmod 28$, $i_2 = \lfloor i/28 \rfloor \bmod 28$, $i_3 = \lfloor i/28^2 \rfloor \bmod 28$, and $\alpha_\ell = \rho^\ell \circ \alpha \circ \rho^{-\ell}$ for an integer ℓ and with ρ being the rotation permutation, i.e. mapping $\rho(0) = 1, \rho(1) = 2$ etc. Similar applies to β_ℓ and γ_ℓ. The permutation π is the reflector of Enigma, i.e. a *fixed* involution without stationary points.

What happens is that whenever we encrypt an integer x_j from x, we first rotate the rightmost rotor (α) by a single position (whenever rotor α completes a full revolution, β is rotated by a single position, and similar applies for β and γ). Thus the total rotation will be $a + j$. We apply all the components of Enigma *in a chain*: the initial signal passes through the plug board, then through the three rotors (that are in the position given by $a + j$), it is then returned by the reflector and sent back through the original path (rotors in reversed order, plug board) and we obtain y_j.

The apprentice cryptographer felt that the sheer complexity of Enigma's design will discourage anyone from attacking his encrypted communication, so he decided to make remembering the key easier for him.

First of all, he decided to get rid of the plug board cables (they always got entangled with his headphones), so for all encryptions, he effectively used the identity permutation in place of σ.

Secondly, he was too lazy to remember a large number for the initial rotation a, so he decided, that remembering a smaller number $a' \in \{1, 2, 3, 4, 5, 6\}$ would suffice, as long as he spices his design up with something "random". So he decided to compute $a = c_4 + a'$, where c_4 is a (randomly picked) constant.

Finally, the apprentice was confident that having four rotors to choose from would be more than enough, so every time he encrypted messages with Enigma, he chose α, β and γ from the set of four permutations $p_{41}, p_{42}, p_{43}, p_{44}$.[1]

In your personal file, you will find the constant c_4 and the four permutations $p_{41}, p_{42}, p_{43}, p_{44}$ (given in the form of a list, containing four permutations, each in form of a list of integers), the reflector permutation pi_4, and an intercepted ciphertext y_4 that is an encryption of Q_4 under the reduced Enigma described above. Recover the message Q_4 and place it in your answer file. Note that we expect you to recover the original string of characters!

Exercise 5 The Adventures of the Crypto-Apprentice: Vernam ++

After playing with his own version of Enigma, the crypto-apprentice discovered the Vernam chipher (while reading the end of the first chapter of the Cryptography & Security course)! What a simple cipher that provides perfect secrecy! Our apprentice thought: "Why not trying to improve this cipher while maintaining (hopefully) its security?" Observing that

[1]Note that we can never have $\alpha = \beta$ or similar, as the rotors are physical objects - once we have placed one in one position, it cannot be placed in another position.

the most annoying thing about the Vernam is its key length, the apprentice decided to think on how to improve this.

At the end of a long session of thinking and researching on possible solutions, he found an improved version of the Vernam cipher and decided to call it Vernam++. He thought he can improve a lot the way we generate the key. In Vernam, we require to have a uniformly distributed key, that has the same length as the message. The apprentice thought he could improve this by just selecting the first element of the key and generating the rest with a congruence pseudo-random number generator.[2]

This works as follows: after selecting the first key element k_{51}, we compute the rest of the key using the relation $k_{5i} = a_5 \cdot k_{5(i-1)} + b_5 \bmod 28$ for $i \geq 2$ (our alphabet has 28 elements). Given that a_5 and b_5 are randomly chosen constants, the new component of the key looks also random. In this way, the key can be actually shorter than the message and everyone could start using Vernam++ to have perfect security. As in the Vernam cipher for a plaintext $p_5 = p_{51} \ldots p_{5j}$ we compute the ciphertext $c_5 = c_{51} \ldots c_{5j}$ as $c_{5i} = p_{5i} + k_{5i} \bmod 28$ for $1 \leq i \leq j$.

To test his Vernam++, our crypto-apprentice encrypted a plaintext, using the encoding from Exercise 2, and obtained the ciphertext c_5. In your parameter file you will find the ciphertext c_5 and the constants a_5 and b_5. Recover the plaintext and provide your answer in Q_5. Note that we expect you to recover the original string of characters!

[2]He read about this on Wikipedia, but completely missed that this is by far no *cryptographic* pseudo-random number generator.