Homework 3 - ElGamal and RSA Cryptography

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_1, Q_2, Q_3, Q_4 and Q_5. You can download your **personal** files on `http://lasec.epfl.ch/courses/cs16/hw3/index.php`
- The answers should all be ASCII sentences except for Exercise 1 and Exercise 5 where we expect you to give us an integer and an array of $\{-1, 1\}$ values. **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 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 **Wednesday the 2nd of November** at 22h00.

Exercise 1 Square Roots

Given a non-zero integer n and an integer $y \in \mathbb{Z}_n$, we call an integer $x \in \mathbb{Z}_n$ a square root of y modulo n, if $x^2 \equiv y \pmod{n}$.

In your parameter file you will find the integers $n_1 = p_1 q_1 r_1$ with p_1, q_1, r_1 all primes, p_1, q_1, r_1 and y_1.

Find the biggest square root of y_1 modulo n_1 and write it under Q_1.

Exercise 2 Weak RSA

Our crypto-apprentice started to read Chapter 3 of the course and found out about RSA. He decided to test it a bit and encrypt an English phrase Q_2. Feeling original, he decided to always take 2 consecutive characters at once and encrypt the pair with the public key (e_2, N_2). Feeling that the design was still too simple to be secure, he also decided to shuffle the resulting list of ciphertexts, hoping that this will increase the security of the scheme. More precisely, he applied a secret permutation on the order of every three consecutive ciphertexts.

Thus, for a message $m = m_1 m_2 \dots m_{2\ell-1} m_{2\ell}$ (where $\ell = 3k$) the apprentice computes $c_1 \dots c_\ell = Enc(\mathsf{encode}(m_1 m_2)) \dots Enc(\mathsf{encode}(m_{2\ell-1} m_{2\ell}))$ and then applies the permutation π to obtain $C_2 = c_{\pi(1)}, c_{\pi(2)}, c_{\pi(3)}, \dots c_{3(k-1)+\pi(1)}, c_{3(k-1)+\pi(2)}, c_{3(k-1)+\pi(3)}$.

The encoding $\mathsf{encode}(\cdot)$ is used to encode ASCII strings as integers. We encode a string $s = s_1 s_2 \dots s_r$ of r ASCII characters as $\mathsf{encode}(s) = \sum_{i=0}^{r-1} \mathtt{ASCII}(s_{i+1}) \cdot 2^{8 \cdot i}$, where $\mathtt{ASCII}(c)$ is the ASCII value of a character c.[1] For example, the encoding of the string “Red Fox!” would be 2411799947438744914.

You will find in your parameter file the ciphertext C_2 and the public key (e_2, N_2). The ciphertext will be given as a list of values that represent the encryption of chunks of two characters. Don’t be scared by the size of the list! Recover the original phrase Q_2.

Exercise 3 RSA with bad modulus

In your parameter file, you will find an RSA public key (e_3, N_3) and a ciphertext c_3 that is an encryption of an encoded secret message $\mathsf{encode}(Q_3)$ under (e_3, N_3). The secret message Q_3 is an ASCII string in English and the encoding is the same as in previous exercise. You also know, that the algorithm that generated the RSA modulus N_3 (for some reason) always outputs moduli of the form $N = p \cdot q$ such that the difference between q^2 and p is not bigger than 2^8 in absolute value.

Break this weak instance of RSA cryptosystem and recover the secret message Q_3.

Exercise 4 Time-dependent ElGamal

After having some problems with RSA, the apprentice cryptographer decided to try a probabilistic encryption scheme and he picked ElGamal. He understood the cryptosystem very well, but he was not sure how to generate the randomness for the encryption. After a lot of thinking, he got the idea that time can be used as a source of randomness. Basically in his idea, the random exponent r for the encryption is picked as a concatenation of the decimal representation of the minute, hour, day, month, and year of the moment when encryption started. For example, if the beginning date and time of encryption was on 28.10.2016, 13:48, the random number is 481328102016.

He used ElGamal in $\mathbf{Z}_{p_4}^*$ and picked $g_4 \in \mathbf{Z}_{p_4}^*$ as a generator. The public key is y_4. In your parameter file, you have the ciphertext (u_4, v_4), the public parameters of ElGamal p_4, g_4 and the public key y_4. The ciphertext (u_4, v_4) is an encryption of an (ASCII) English phrase encoded with the same encoding as in Exercise 2. Recover the secret phrase and write it under Q_4.

Hint: The ciphertext was computed sometime between last Wednesday and the day of publishing Homework 3, i.e. between 12th of October $00:00$ and 20th of October $00:00$.

[1] Since ASCII characters can be represented by 8-bit binary strings, this encoding is injective and can be inverted.

Exercise 5 ElGamal and subgroup problem

Our crypto-apprentice was filling a form of a company that asks some private yes/no questions. He was worried about leakage of his private information to some third-party. Therefore, he decided to encrypt his answer. He looked through the encryption schemes that he saw in the Cryptography and Security course. He saw previously that RSA is not appropriate to encrypt only yes or no messages because it is deterministic. The only probabilistic encryption scheme that he knew was ElGamal. So, he decided to give it one more chance and hopefully use it in a proper way. In the end, he arranged everything with the company and received their ElGamal-public key y_5 along with the public parameters $\mathbf{Z}_{p_5}^*$ and its generator g_5.

After having problems with the randomization in ElGamal, he decided to use a proper randomness generator. In addition to that, since he can only encrypt messages in $\mathbf{Z}_{p_5}$, he used '1' to encrypt 'yes' and '-1' for no.

In your parameter file, you have the six ciphertexts $\{(u_{5,1}, v_{5,1}), ..., (u_{5,6}, v_{5,6})\}$ where each of them is the encryption of one of our crypto-apprentice's answers. You also have the public parameters of ElGamal p_5, g_5 and the public key y_5. Recover each plaintext and write them in form of an ordered list under Q_5.[2]

Hint: Look for a subgroup of index 2!

[2]I.e. the plaintext of $u_{5,1}$ should appear in the list on the first position etc.