Homework 2 - Prehistoric to Modern Crypto

Cryptography and Security 2020

- 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 all `Q` values specified in the questions below. You can download your **personal** files from the following link:
`http://lasec.epfl.ch/courses/cs20/hw2/index.php`
- You will find an example parameter and answer file on the moodle. You can use this parameters' file to test your code and also ensure that the types of `Q` values you provided match what is expected. For instance, the variable `Q1_order` should be an integer, whereas `Q2` is an ASCII string. **Please do not put any comment or 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. Notebook files are allowed, but we prefer if you export your code as a textfile with a sage/python script.
- 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/corrections in 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 **Sunday the 25th of October** at 23h59.

Exercise 1 Lateralus

After attending the crypto lecture on Diffie-Hellman keyexchange, our beloved crypto-apprentice Maynard decides to create his own cryptosystem. After reading some conspiracy theory blogs and believing that the Fibonacci sequence carries some subliminal messages, he decides to make his construction based on this sequence. So here is how the protocol works. It consists of 2 algorithms $\mathsf{GenKeys}(\lambda)$ and $\mathsf{KeyAgreement}(pk', sk)$.

$\mathsf{GenKeys}$ takes a security parameter, selects a prime p and a non-negative integer sk, which will be our secret key. Then it sets the public key to be $pk = (F_{sk} \mod p, F_{sk-1} \mod p, p)$, where F_i is the i^{th} Fibonacci number, i.e. $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $\ldots$. Finally $\mathsf{KeyAgreement}(pk_B, sk_A)$ takes the other parties public key $pk_B = (pk_1, pk_2, p)$ and outputs the sk_A^{th} Fibonacci number after pk_1. For instance if $pk_B = (5, 3, 13)$ and $sk_A = 3$, the derived key will be $21 \mod 13 = 8$.

In your parameter file you will find your own secret key sk and another parties public key pk. Your goal is to implement the $\mathsf{KeyAgreement}$ function and output the derived key (as an integer between 0 and p) in your solution file. To verify your solution you can use the test parameters for this question.

Exercise 2 VGhlIGJhc2Ugb2YgdHJhbnNwb3NpdGlvbg==

Because of the pandemic, our new Crypto Apprentice was stuck all summer in the Military History Archive in Bratislava. In order to learn some cryptography before starting the COM-401 course, she decided to implement her own version of a WWII Slovak cipher based on triple columnar transposition [1].

In a columnar transposition, a key of length ℓ_k is a permutation of the list $[1, 2, \ldots, \ell_k]$. The message is written in a 2D array of width ℓ_k and if necessary, some padding is added in order to fill the array. Then the columns are permuted according to the key and the ciphertext is the text resulting from reading column by column the resulting array. As an example, the plaintext *lake town* with padding '?' and the key $3, 1, 4, 2$ is written in matrix form as

3	1	4	2
l	a	k	e
	t	o	w
n	?	?	?

Reading the columns in the order given by the key, the resulting ciphertext is *at?ew?l nko?*.

Coming back to our apprentice, she decided to modify the Slovak cipher by using two columnar transpositions instead of three and to first encode the plaintext in base64, because "come on, we are in 2020!". The resulting encryption function takes as inputs two keys I, II, a secret offset N and the message pt. It proceeds as follows.

1. The message is first encoded in an ascii bitstring and the latter is encoded in a base64 **string**. We call this result $\mathsf{pt64}$.

2. The key I is first rotated to the right by N (e.g. with $N = 2$, $[1, 2, 3, 4, 5]$ becomes $[4, 5, 1, 2, 3]$). The resulting key is used to encrypt $\mathsf{pt64}$ with columnar transposition **with space as padding** (i.e. ' '). This gives a ciphertext ct_1.

[1] see `https://doi.org/10.3384/ecp2020171004` if interested.

3. The key II is rotated to the right by N. Finally, ct_1 is encrypted with columnar transposition using the rotated II and ' ' as padding (as before). The function outputs the resulting ciphertext ct.

Using this cipher, the Crypto Apprentice sent you a postcard with an encrypted short "sentence" (`Q2_ct` in your parameter file). While you managed to retrieve the secret keys I, II (`Q2_I` and `Q2_II` in your parameter file), she forgot to tell you the offset N. Your goal is to recover the plaintext (which is a "sentence" in english) and to put it in the `Q2` variable of your answer file.
Hint: The `encode` function of strings and the `base64` library in python might be useful.

Exercise 3 The Return of GEDEFU

Throughout this exercise, we use the following notations:

- $\mathbf{A} = \{A, \ldots, Z, 0, \ldots, 9\}$ (capital alphanumeric),
- $\mathbf{C} = \{A, D, F, G, V, X\}$.
- $\mathfrak{S}_\ell$ denotes the group of permutations over $\{1, \ldots, \ell\}$,
- $||$ denotes the concatenation operator,

During World War I, the German army used the ADFGVX cipher to encrypt non-empty plaintexts $(x_i)_i \in \mathbf{A}^*$ to non-empty ciphertexts $(y_i)_i \in \mathbf{C}^*$. Historically, messages were not meant to contain special characters such as white spaces and the cipher has an ambiguous decryption because of an internal padding operation. The encryption algorithm described by Algorithm 1 is parametrized by a secret key $\mathsf{sk} = (\phi, \gamma, \sigma)$ consisting of an invertible substitution $\phi \colon \mathbf{A} \longrightarrow \mathbf{C} \times \mathbf{C}$ called a *Polybius square*, a *padding character* $\gamma \in \mathbf{C}$ and a *permutation* $\sigma \in \mathfrak{S}_\ell$. We denote by $\mathcal{K}$ the space of all secret keys. In practice, ϕ is represented either by a 6×6 matrix $S = (s_{uv})_{u,v \in \mathbf{C}}$ indexed by $\mathbf{C} \times \mathbf{C}$ such that $\phi(x)$ is defined to be the unique pair (u, v) for which $s_{uv} = x$ or by a flattened row-major ordered matrix as illustrated on Figure 1. Figure 2 illustrates the ambiguous behaviour of the ADFGVX cipher with S and σ as given on Figure 1. Ambiguity arises from the ill-defined reverse padding operation.

Algorithm 1: $\mathsf{enc}(\mathsf{sk}, x)$

Input: A secret key $\mathsf{sk} = (\phi, \gamma, \sigma)$ and a plaintext x of length $n > 0$.
Output: A ciphertext y.

$x_1 || \cdots || x_n \leftarrow x$
for $i = 1, \ldots, n$ **do**
  $y'_{2i-1} || y'_{2i} \leftarrow \phi(x_i)$
$y' \leftarrow y'_1 || \cdots || y'_{2n}$
Apply the columnar transposition of padding γ described by σ on y' to get a string y.
return y

$$S = \begin{bmatrix} A & B & C & D & E & F \\ G & H & I & J & K & L \\ M & N & O & P & Q & R \\ S & T & U & V & W & X \\ Y & Z & 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 & 8 & 9 \end{bmatrix} \leftrightarrow A \cdots Z 0 \cdots 9$$

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 3 & 5 & 4 & 6 \end{pmatrix} \leftrightarrow (2, 1, 3, 5, 4, 6)$$

Figure 1: Example of a substitution and a permutation.

$$x = \texttt{GNU1} \overset{\phi}{\to} \texttt{DAFDGFVG} \overset{pad}{\to} \begin{bmatrix} \texttt{D} & \texttt{A} & \texttt{F} & \texttt{D} & \texttt{G} & \texttt{F} \\ \texttt{V} & \texttt{G} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} \end{bmatrix} \overset{\sigma}{\to} \begin{bmatrix} \texttt{A} & \texttt{D} & \texttt{F} & \texttt{G} & \texttt{D} & \texttt{F} \\ \texttt{G} & \texttt{V} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} \end{bmatrix} \to \texttt{AGDVFXGXDXFX}$$

$$y = \texttt{AGDVFXGXDXFX} \overset{\sigma^{-1}}{\to} \begin{bmatrix} \texttt{D} & \texttt{A} & \texttt{F} & \texttt{D} & \texttt{G} & \texttt{F} \\ \texttt{V} & \texttt{G} & \texttt{X} & \texttt{X} & \texttt{X} & \texttt{X} \end{bmatrix} \overset{unpad}{\to} \begin{cases} \texttt{DAFDGFVGXXXX} \\ \texttt{DAFDGFVGXX} \\ \texttt{DAFDGFVG} \end{cases} \overset{\phi^{-1}}{\to} \begin{cases} \texttt{GNU199} \\ \texttt{GNU19} \\ \texttt{GNU1} \end{cases}$$

Figure 2: Example of an ambiguous encryption/decryption with a padding character $\gamma = X$. Since $\sigma \in \mathfrak{S}_6$ and the substitution doubles the length of a plaintext, only even padding lengths are appropriate for a ciphertext with even length, whence the three cases.

- ▷ Given a ciphertext `Q3a_y`, the secret key $(\texttt{Q3a\_S}, \texttt{Q3a\_c}, \texttt{Q3a\_s}) \in \mathcal{K}$ used for encryption and a plaintext length `Q3a_n`, recover the original plaintext and report it in the answers file under `Q3a_x`.

In order to encrypt human-readable messages containing non-alphanumeric uppercase characters such as punctuation or spaces, we need to extend the alphabet supported by the cipher. Given a set of special characters $\mathbf{P}$, let $\Delta\mathbf{P} = \{(x, x) \in \mathbf{P} \times \mathbf{P} : x \in \mathbf{P}\}$ and extend a substitution $\phi \colon \mathbf{A} \longrightarrow \mathbf{C} \times \mathbf{C}$ to $\phi_+ \colon \mathbf{A} \sqcup \mathbf{P} \longrightarrow (\mathbf{C} \times \mathbf{C}) \sqcup \Delta\mathbf{P}$ by $\phi_+(x) = \phi(x)$ if $x \in \mathbf{A}$ and by $\phi_+(x) = (x, x)$ if $x \in \mathbf{P}$. We then define the ADFGVX$_+$ to be the usual ADFGVX with ϕ_+ instead of ϕ. One may imagine another extension of the cipher to support special symbols, but for simplicity, we only duplicate the special characters so that the key space for the ADFGVX$_+$ cipher coincides with the key space of the ADFGVX cipher.

- ▷ Encrypt the given plaintext `Q3b_x` using the given secret key $(\texttt{Q3b\_S}, \texttt{Q3b\_c}, \texttt{Q3b\_s}) \in \mathcal{K}$ and report the ciphertext in the answers file list `Q3b_y`.

We assume that the original plaintexts are human-readable English messages consisting of capital letters and numbers, punctuation symbols and white spaces. In Python, the set of punctuation and white spaces is given by:

```
>>> import string
>>> P = string.punctuation + string.whitespace
```

To uniquely characterize unknown plaintexts up to a negligible probability, we also provide an additional data computed as the `md5` hexadecimal digest of the `base64` representation of the plaintext. Since the hash functions and digests have not officially been seen in the lectures yet, we provide the following code snippet that was used to compute those digest values.

```
>>> import hashlib
>>> import base64
>>> s = 'test'                         # Python 3.x: s = 'test'.encode()
>>> r = base64.b64encode(s)            # str in Python 2, bytes in Python 3
>>> H = hashlib.md5(r).hexdigest()     # '5fa62ae6176f3746142503a6ebe96cb3'
```

A substitution matrix S is said to be *incomplete* if S contains "missing" entries represented by a character z_0 that is never used in a plaintext or a ciphertext, say $z_0 =$ `\x00`.

- ▷ Given an incomplete substitution[2] `Q3c_S`, a padding character `Q3c_c` and knowing that the secret permutation σ satisfies $\sigma \in \mathfrak{S}_\ell$ for some $2 \le \ell \le 6$, decrypt the given ciphertext `Q3c_y1` and report the plaintext in the answers file under `Q3c_x1`. To avoid ambiguities, the following data is provided:
  - A plaintext-ciphertext pair (`Q3c_x0`, `Q3c_y0`) encrypted using the same[3] secret key.
  - The `md5` digest (`Q3c_H`) of the `base64` encoding of the unknown plaintext `Q3c_x1`.
- ▷ Given an incomplete substitution[2] `Q3d_S`, a padding character `Q3d_c` and the secret permutation `Q3d_s`, decrypt the given ciphertext `Q3d_y1` and report the plaintext in the answers file under `Q3d_x1`. To avoid ambiguities, the following data is provided:
  - A plaintext-ciphertext pair (`Q3d_x0`, `Q3d_y0`) encrypted using the same[3] secret key.
  - The `md5` digest (`Q3d_H`) of the `base64` encoding of the unknown plaintext `Q3d_x1`.

[2]The incomplete substitution matrix is flattened in a row-major order and is given as a Python `str` in the parameters file, e.g. a matrix $\begin{bmatrix} \text{'A'} & \text{'\textbackslash x00'} \\ \text{'\textbackslash x00'} & \text{'B'} \end{bmatrix}$ is given as `'A\x00\x00B'`.

[3]The plaintext-ciphertext pairs are encrypted using the full secret key with a complete substitution.