Homework 1 – Classical Cryptography

Cryptography and Security 2022

- ◇ 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 `[id]-answers.txt` file where `[id]` is an integer derived from the student ID. This means you need to provide us with all `Q`-values specified in the questions below. The **personal** files can be retrieved at

  http://lasec.epfl.ch:80/courses/cs22/hw1/index.php.

  Depending on the nature of the exercise, an example of submission may be provided.
- ◇ **Please do not put any comment or strange character or any new line** in the submission file. Submissions that do not respect the expected format may lose points.
- ◇ Do **NOT** modify the `SCIPER`, `id` and `seed` headers in the `[id]-answers.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 normal textual files containing Python/SAGE code. If an answer is incorrect, we may grant partial marks depending on the implementation.
- ◇ Be careful to always cite external code that was used in your implementation if the latter is not part of the public domain and include the corresponding license if needed. Submissions that do not meet this guideline may be flagged as plagiarism or cheating.
- ◇ Some plaintexts may contain random words. Do not be offended by them and search them online at your own risk. Note that they might be really strange.
- ◇ Please list the names of **every** person you worked with in the designated area of the answers file. Write down the list as a single line comma-separated list.
- ◇ Corrections and revisions on this homework may be announced on Moodle in the “news” forum. By default, 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 **November 02, 2022** at 23h59.

Exercise 1 Return of the Imitation Game

During World War II, the German army encrypted their communications using the Enigma cipher, invented at the end of World War I by the German engineer Arthur Sherbius. There exist many variants of the Enigma machine, such as Enigma **M3** (presented during the lecture). While historically defined to encrypt the Latin alphabet, it is possible to use the Enigma cipher over any alphabet $\mathscr{L}$ of literals $\ell_0, \dots, \ell_{N-1}$. The purpose of this exercise is to assess the security of the generalized Enigma cipher in a context close to World War II. Since the complete cryptanalysis of this cipher exceeds the scope of the lecture, we will focus on simple attacks where the adversary has access to partial knowledge.

Notation 1. We denote by $\tau\colon \mathscr{L} \longrightarrow \mathscr{L}$ the cyclic permutation defined by $\tau(\ell_i) \triangleq \ell_{(i+1) \bmod N}$, where $0 \leq (i \bmod N) \leq N-1$ by convention. In particular, $\tau^k(\ell_i) = \ell_{(i+k) \bmod N}$ for all $k \geq 0$.

Notation 2. The set of fixed points $\mathscr{L}^\sigma$ of a bijection $\sigma\colon \mathscr{L} \longrightarrow \mathscr{L}$ is defined by

$$\mathscr{L}^\sigma \triangleq \{x \in \mathscr{L} \colon \sigma(x) = x\}.$$

Definition 1. An *involution* is a function $\sigma\colon \mathscr{L} \longrightarrow \mathscr{L}$ such that $\sigma \circ \sigma = id$ where $id\colon \mathscr{L} \longrightarrow \mathscr{L}$ denotes the identity function. More generally, a *t-fixing involution* is an involution $\sigma\colon \mathscr{L} \longrightarrow \mathscr{L}$ with $t \geq 0$ fixed points.

Definition 2. A *rotor* $R_i = \big(\rho_i, p_{i,0}, (p_{i,1}, \dots, p_{i,k_i})\big)$ is defined by a permutation $\rho_i\colon \mathscr{L} \longrightarrow \mathscr{L}$, an integer $0 \leq p_{i,0} \leq N-1$ and $k_i \geq 0$ distinct integers $0 \leq p_{i,0} < \cdots < p_{i,k_i} \leq N-1$. We write $R_i = (\rho_i, p_{i,0}, \perp)$ to indicate that $k_i = 0$.

The integer $p_{i,0}$ is called the (initial) *position* of the rotor and the set $P_i \triangleq \{p_{i,1}, \dots, p_{i,k_i}\}$ is called the set of *turnover positions* of the rotor. The initial position corresponds to the initial *Ringstellung* (ring setting) of the Enigma cipher, whereas the turnover positions correspond to the physical notches on the rotor. We will provide an explanation on their purpose when presenting the encryption algorithm.

Definition 3. An *Enigma configuration* $K = (R_1, \dots, R_s, \pi, \sigma)$ consists of s rotors $R_1, \dots, R_s$, a 0-fixing involution $\pi\colon \mathscr{L} \longrightarrow \mathscr{L}$ and a t-fixing involution $\sigma\colon \mathscr{L} \longrightarrow \mathscr{L}$.

The permutations π and σ are called the *reflector* and the *plugboard transformation* of the configuration respectively. Consider the following Enigma configuration:

$$K = \Big(R_1, \dots, R_i = \big(\rho_i, p_{i,0}, (p_{i,1}, \dots, p_{i,k_i})\big), \dots, R_s, \pi, \sigma\Big).$$

In Python, K is encoded as follows:

- A subset $\mathscr{L} = \{\ell_0, \dots, \ell_{N-1}\}$ of the UTF-8 charset (defined by ISO/IEC 10646) is encoded as an object $L = [L[0], \dots, L[N-1]]$ of type `list` such that $L[i] = \texttt{ord}(\ell_i) = \omega_i$ where `ord` maps a one-character string to its Unicode code point ω_i (henceforth: *ordinal*) of type `int`. For instance, the encoding of $\mathscr{L} = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ is $[97, 98, 99]$. The reverse transformation `chr` maps an ordinal $0 \leq \omega \leq \texttt{0x10ffff}$ to its Unicode one-character string (henceforth: *character*).

- An UTF-8 message $m = m_0 || \cdots || m_{\nu-1}$ of length ν, where $||$ denotes the concatenation operator, is encoded as a `list` of integers $M = [M[0], \dots, M[\nu-1]]$ such that $M[i]$ is the ordinal of m_i. For instance, the encoding of $m = \texttt{abca}$ is $[97, 98, 99, 97]$.

- A permutation $\bar{\sigma}$ of $\{0,\dots,N-1\}$ is encoded as a `list` $S = [S[0],\dots,S[N-1]]$ of `int` such that $S[i] = \bar{\sigma}(i)$. In particular, a permutation of $\sigma\colon \mathscr{L} \longrightarrow \mathscr{L}$ is identified to a permutation of $\bar{\sigma}$ of $\{0,\dots,N-1\}$ such that $\sigma(\ell_i) = \ell_{\bar{\sigma}(i)}$. For instance, the encoding of the permutation $(\ell_0 \to \ell_2, \ell_1 \to \ell_0, \ell_2 \to \ell_1)$ is $[2,0,1]$.

- A rotor $R_i = \big(\rho_i, p_{i,0}, (p_{i,1},\dots,p_{i,k_i})\big)$ with permutation $\rho_i\colon \mathscr{L} \longrightarrow \mathscr{L}$, initial position $p_{i,0}$ and turnover positions $p_{i,1} < \cdots < p_{i,k_i}$ is encoded as a `list` $R = [\bar{\rho}_i, p_{i,0}, [p_{i,1},\dots,p_{i,k_i}]]$ where $\bar{\rho}_i$ is the encoding of ρ_i. Note that $R[2]$ may possibly be the empty. In particular, the list of rotors of an Enigma configuration are encoded as a `list` of `list`.

- An Enigma configuration K is encoded as a `list` $K = [K[0], K[1], K[2]]$ such that $K[0]$ is a list of encoded rotors, $K[1]$ is the encoding of the fixed-point free involution π and $K[2]$ is a pair containing the encoding of the t-fixed point involution σ and the number of fixed points t as an integer. For instance, consider the Enigma configuration

$$K = \Big(R_1 = \big(\rho_1, p_{1,0}, (p_{1,1}, p_{1,2})\big), R_2 = (\rho_2, p_{2,0}, \perp), \pi, \sigma\Big) \tag{1.1}$$

  over the alphabet $\mathscr{L} = \{\ell_0,\ell_1,\ell_2,\ell_3\}$ where $\rho_1 = (\ell_0 \to \ell_1, \ell_1 \to \ell_3, \ell_2 \to \ell_2, \ell_3 \to \ell_0)$ and $\rho_2 = (\ell_0 \to \ell_0, \ell_1 \to \ell_1, \ell_2 \to \ell_2, \ell_3 \to \ell_3)$ are the permutations of rotor 1 and 2 respectively, $p_{1,0} = 2$ and $p_{2,0} = 1$ are the rotor initial positions, $(p_{1,1}, p_{1,2}) = (1,3)$ and the rotor 1 turnover positions, $\pi = (\ell_0 \to \ell_3, \ell_1 \to \ell_2, \ell_2 \to \ell_1, \ell_3 \to \ell_0)$ is the reflector and $\sigma = (\ell_0 \to \ell_1, \ell_1 \to \ell_0, \ell_2 \to \ell_2, \ell_3 \to \ell_3)$ is the plugboard transformation. Then, K is encoded as

$$\Big[\Big[[[1,3,2,0],2,[1,3]],[[0,1,2,3],1,[]]\Big],[3,2,1,0],[[1,0,2,3],2]\Big]. \tag{1.2}$$

A message $m = m_1 \| \cdots \| m_n$ is a sequence of literals in $\mathscr{L}$ and extra characters not in $\mathscr{L}$. We denote by $\mathscr{X}$ the set of extra characters, which upon encryption have no effect. In practice, the initial Enigma configuration evolves during the encryption as illustrated by Algorithm 1. Updating $p_{i,0} \leftarrow (p_{i,0}+1) \bmod N$ is called the *stepping of* R_i. If the rotor at index $1 \le i \le s-1$ steps from a turnover position, then rotor at index $i+1$ also steps. Each rotor steps at most once per round. Furthermore, the last rotor R_s does not cause another rotor to step even if $p_{s,0}$ is a turnover position. Table 1 illustrates the initial positions evolution for an Enigma configuration $((\rho_1, p_{1,0}, P_1), (\rho_2, p_{2,0}, P_2), (\rho_3, p_{3,0}, P_3)), \pi, \sigma)$ defined over $\mathscr{L} = \{\mathtt{A},\dots,\mathtt{Z}\}$ such that $(p_{1,0}, p_{2,0}, p_{3,0}) = (\mathtt{O},\mathtt{D},\mathtt{K})$ and $(P_1,P_2,P_3) = (\{\mathtt{Q}\},\{\mathtt{E}\},\{\mathtt{V}\})$.

call	$p_{1,0}$	$p_{2,0}$	$p_{3,0}$
0	O	D	K
1	P	D	K
2	Q	D	K
3	R	E	K
4	S	F	L

Table 1: Evolution of $(p_{1,0}, p_{2,0}, p_{3,0}) = (\mathtt{O},\mathtt{D},\mathtt{K})$ with $(P_1,P_2,P_3) = (\{\mathtt{Q}\},\{\mathtt{E}\},\{\mathtt{V}\})$.

Algorithm 1: $\mathsf{Enc}(K, \mathsf{pt})$

```
Input: An Enigma state $K$ and a plaintext $\mathsf{pt} = m_1 \| \cdots \| m_n \in (\mathscr{L} \sqcup \mathscr{Z})^*$.
Output: An Enigma state $K'$ and a ciphertext $\mathsf{ct} = c_1 \| \cdots \| c_n \in (\mathscr{L} \sqcup \mathscr{Z})^*$.
1  $K \to \big((\rho_1, p_{1,0}, P_1), \ldots, (\rho_s, p_{s,0}, P_s), \pi, \sigma\big)$
2  for $j = 1, \ldots, n$ do
3      if $m_j \notin \mathscr{L}$ then
4          $c_j \leftarrow m_j$                         ▷ "non-encryptable" characters are ignored
5      else
6          $\mathcal{B} \leftarrow \emptyset$
7          $p_{1,0}^{old} \leftarrow p_{1,0}$
8          $p_{1,0} \leftarrow (p_{1,0} + 1) \bmod N$                         ▷ rotor 1 always steps
9          if $p_{1,0}^{old} \in P_1$ then
10             $p_{2,0} \leftarrow (p_{2,0} + 1) \bmod N$
11             $\mathcal{B} \leftarrow \mathcal{B} \cup \{2\}$              ▷ rotor 2 will not step again in this round
12         for $i = 2, \ldots, s - 1$ do
13             if $i \notin \mathcal{B}$ and $p_{i,0} \in P_i$ then
14                 $p_{i,0} \leftarrow (p_{i,0} + 1) \bmod N$           ▷ rotor $i$ steps because of turnover
15                 $p_{i+1,0} \leftarrow (p_{i+1,0} + 1) \bmod N$       ▷ rotor $i + 1$ steps because of rotor $i$
16                 $\mathcal{B} \leftarrow \mathcal{B} \cup \{i + 1\}$
17         for $i = 1, \ldots, s$ do
18             $\alpha_i \leftarrow \tau^{-p_{i,0}} \circ \rho_i \circ \tau^{p_{i,0}}$                  ▷ $\tau$ is the cyclic shift
19         $c_j \leftarrow \big(\sigma^{-1} \circ (\alpha_s \circ \cdots \circ \alpha_1)^{-1} \circ \pi \circ (\alpha_s \circ \cdots \circ \alpha_1) \circ \sigma\big)(m_j)$
20 $K' \leftarrow \big((\rho_1, p_{1,0}, P_1), \ldots, (\rho_s, p_{s,0}, P_s), \pi, \sigma\big)$
21 $\mathsf{ct} \leftarrow c_1 \| \cdots \| c_n$
22 return $(K', \mathsf{ct})$
```

The construction above generalizes the Enigma cipher by allowing an arbitrary number of rotors and turnover positions per rotor. On an Enigma **M3** unit, the second (resp. third) rotor moves forward once the first (resp. second) rotor completes a full revolution. This can be interpreted as having a single turnover position on rotor 1 and rotor 2. According to the specifications of the **M3** unit, the plugboard transformation has a fixed number of fixed points whereas our construction allows for an arbitrary number.

Question 1.1

An algorithm $\mathsf{Dec}(-, -)$ is said to complete Enc if for every plaintext pt and for every Enigma configuration K, we have $\mathsf{Dec}(K, \mathsf{ct}) = (K', \mathsf{pt})$ where $(K', \mathsf{ct}) = \mathsf{Enc}(K, \mathsf{pt})$. Note that Dec takes as input the Enigma configuration *before* it was modified by the encryption algorithm.

▷ Implement the decryption algorithm for Enc. The implementation can be tested using the plaintext-ciphertext pair $(\mathsf{pt}, \mathsf{ct}) = (\mathsf{Q1a\_pt}, \mathsf{Q1a\_ct})$ encrypted with the Enigma configuration $K = (\mathsf{Q1a\_R}, \mathsf{Q1a\_pi}, \mathsf{Q1a\_sigma})$. The alphabet and the extra characters sets are specified by Q1a_L and Q1a_Z respectively and encoded as lists of ordinals.

Question 1.2

- ▷ Given an encoding Q1b_L of a subset $\mathscr{L} = \{\ell_0, \dots, \ell_{N-1}\}$ of the UTF-8 charset, an encoding Q1b_Z of the set of extra characters $\mathscr{Z}$, an UTF-8 ciphertext Q1b_ct and an Enigma state $K = (\mathsf{Q1b\_R}, \mathsf{Q1b\_pi}, \mathsf{Q1b\_sigma})$, recover the corresponding UTF-8 plaintext and report its `base64` encoding (of type `str`) under Q1b_pt.

Question 1.3

In the second part of the exercise, the goal is, given partial information, to recover a secret message that was encrypted and the Enigma configuration used for the encryption.

- ▷ Given an encoding Q1c_L of a subset $\mathscr{L} = \{\ell_0, \dots, \ell_{N-1}\}$ of the UTF-8 charset, an encoding Q1c_Z of the set of extra characters $\mathscr{Z}$, an UTF-8 ciphertext $\mathsf{ct} = \mathsf{Q1c\_ct}$, the evolution Q1c_trace of the Enigma state encoded as a `dict` whose keys are integers $1 \le j \le n$ and values are partial (encoded) Enigma states *before* encrypting the j-th character m_j of the plaintext $m = m_1 || \cdots || m_n \in (\mathscr{L} \sqcup \mathscr{Z})^*$ and a partial view Q1c_pt_view of the UTF-8 plaintext m, recover the *initial* Enigma state (as encoded by (1.2)) and the `base64` encoding (of type `str`) of m and report them under Q1c_sk and Q1c_pt respectively. Stated otherwise, $\mathsf{Q1c\_pt} = \mathsf{Enc}(\mathsf{Q1c\_sk}, \mathsf{Q1c\_ct})$.

  To reduce the size of the parameters file, permutations are encoded as dictionaries whose keys and values are integers $0 \le i \le N-1$. For instance, $\{0 : 3\}$ describes a permutation mapping $\ell_0 \to \ell_3$. Unknown ordinal values are represented by -1.

  To avoid the case where two secret keys give the correct plaintext, a checksum for the initial Enigma state and the `base64` encoding of the plaintext are given[1] by Q1c_sk_checksum and Q1c_pt_checksum respectively and are computed by the following function:

```
import hashlib

def checksum(*args, sep=';'):
    data = sep.join(map(str, args)).encode()
    return hashlib.new('md5', data=data).hexdigest()
```

  For instance, the following snippet computes the checksum of the Enigma state defined by (1.1) and whose encoding is given by (1.2):

```
R1 = [[1, 3, 2, 0], 2, [1, 3]]
R2 = [[0, 1, 2, 3], 1, []]
state = [[R1, R2], [3, 2, 1, 0], [[1, 0, 2, 3], 2]]
assert checksum(state) == "1d40bc1bd8fba1685c29abaf85adad3c"
```

  Similarly, the checksum of an UTF-8 plaintext is computed as follows:

```
import base64

mess = "hello world"
data = base64.b64encode(mess.encode()).decode()  # type: str
assert checksum(data) == "bef52dc1307a739dccaf74f068dbc8df"
```

[1]For completeness, the plaintext checksum for the previous question is given by Q1b_pt_checksum.

Exercise 2 SCIPHER

In this exercise, we consider a tweaked ElGamal Cryptosystem defined over the alphabet $\Sigma = \{\ell_0, \ldots, \ell_{25}\} = \{\texttt{A}, \ldots, \texttt{Z}\}$ which is a subset of the UTF-8 charset. Plaintexts are elements of Σ^* whereas ciphertexts are hexadecimal strings separated by colons (:) or semi-colons (;). We denote the empty string by $\perp$ and the concatenation operator by $||$.

```
KeyGen(1^λ)
1 :  $(p, g) \leftarrow\$ \mathsf{GroupGen}(\lambda)$
2 :  $\mathsf{sk} \leftarrow\$ \mathbf{Z}_p^*$
3 :  $pp \leftarrow (p, g)$
4 :  $\mathsf{pk} \leftarrow g^{\mathsf{sk}}$
5 :  return $(\mathsf{sk}, \mathsf{pk}, pp)$
```

```
Enc(pk, m, pp)
1 :  parse $pp \rightarrow (p, g)$
2 :  parse $m \rightarrow (m_1, \ldots, m_n)$
3 :  $C \leftarrow \perp$
4 :  for $i = 1 \ldots n$
5 :      $r \leftarrow\$ \mathbf{Z}_p^*$
6 :      $c_1 \leftarrow \mathsf{pk}^r \cdot g^{\mathtt{ord}(m_i)}$
7 :      $c_2 \leftarrow g^r$
8 :      $C \leftarrow C || \mathtt{hex}(c_1) : \mathtt{hex}(c_2) ;$
9 :  return $C$
```

- The $\mathsf{GroupGen}(\lambda)$ procedure outputs a 1024-bit prime $p = 2q + 1$ such that q is prime and an element $g \in \mathbf{Z}_p^*$ of order q.
- $\mathtt{ord}(\ell)$ is the Unicode code point of $\ell \in \Sigma$, namely $\mathtt{ord}(\ell) \in \{65, \ldots, 90\}$.
- $\mathtt{hex}(c)$ is the hexadecimal representation of c without a `0x` prefix, e.g., $\mathtt{hex}(255) = \mathtt{ff}$.
- Both `:` and `;` are used as literal characters (i.e. if $c_1 = 255$ and $c_2 = 110$, then C is concatenated with `ff:6e;`).

Question 2.1

▷ Implement the decryption $\mathsf{Dec}(\mathsf{sk}, C, \mathsf{pp})$ procedure for this cryptosystem.

▷ Given public parameters $\mathsf{pp} = (\mathsf{Q2a\_p}, \mathsf{Q2a\_g})$, a secret key sk under Q2a_sk, recover the ASCII plaintext m of the ciphertext $\mathsf{Q2a\_C} = \mathsf{Enc}(\mathsf{pk}, m, \mathsf{pp})$ and report it under Q2a_m.

Question 2.2

Given an ASCII message $m = m_1 || \cdots || m_n \in \Sigma^*$ and an integer $K = k_1 || \cdots || k_\nu$ with digits (from left to right) given by $k_1, \ldots, k_\nu$, let $\mathtt{SCIPHER}_K(m)$ be the textbook Vigenère cipher over the message space Σ^* and secret key K which shifts m_i by the i-th digit k_i of K. Stated otherwise, if m_j is $\ell_i \in \Sigma$, then its shift by the digit $0 \leq k_j \leq 9$ is $\ell_{(i+k_j) \bmod 26} \in \Sigma$. By convention, the key is repeated to handle messages of arbitrary length.

▷ Given public parameters $\mathsf{pp} = (\mathsf{Q2b\_p}, \mathsf{Q2b\_g})$, a public key pk under Q2b_pk, a ciphertext $C = \mathsf{Enc}(\mathsf{pk}, m, \mathsf{pp})$ under Q2b_C1 for an unknown plaintext m and a Vigenère key K under Q2b_KEY, find a ciphertext C' such that $\mathtt{SCIPHER}_K(m) = \mathsf{Dec}(\mathsf{sk}, C', \mathsf{pp})$ and that all second components (i.e. c_2 in line 8 of Enc) remain unchanged (i.e. equivalent to the ones in Q2b_C1). Report C' under Q2b_C2. You can check whether your solution is correct or not by using the checksum value Q2b_C2hash. The assertion to check is:

```
hashlib.sha256(Q2b_C2.encode()).hexdigest() == Q2b_C2hash
```

Important Note: You must omit the $(\cdot) \bmod 26$ in your computations and assume that the plaintext characters never overflow (or that it is handled by the receiver). You are only responsible for shifting.

Exercise 3 Steganography: Caesar meets Vigenère.

The goal of steganography is similar to the one of encryption, namely you want to transfer some information to someone without anyone else being able to read it. However, the means of doing it are completely different. In steganography, you hide the information (message, image, ...) within another object, and the "key" to recover it is simply the location of the hidden information.

Typically, the information is represented in binary form within an image. An image is usually encoded in the **RGB** format where each pixel is composed of a triplet of bytes containing the **R**ed, **G**reen, **B**lue values of the corresponding channels. A common way to hide information is then to put the bits of information within the least or most significant bit(s) of the Red, Green or Blue channel values. This allows to hide quite a large amount of information but avoids that the latter can be visually detected.

In this exercise, you will explore a mix of steganography and ancient cryptography as follows. Given two ciphertexts as Q3a_ct and Q3b_ct and an image encoded as an $N \times M$ integral array in Q3_img where each pixel is represented by the integer $(r \ll 16) + (g \ll 8) + b$ with r, g and b being the RGB values of that pixel, your goal is to recover an **English passphrase** (in lowercase and no whitespace) hidden in the image. While this exercise may be solved entirely using arrays, displaying the image might help you understand the hints given by the ciphertexts. In Python, images may be handled as follows:

```
import matplotlib.pyplot as plt
import numpy as np
from skimage.io import imshow

data = np.array(Q3_img, dtype=int) # N x M
data = [(data >> (16 - 8 * i)) & 0xFF for i in range(3)]
img = np.dstack(data).astype(np.uint8) # N x M x 3
plt.plot(), imshow(img)
plt.show()
```

Question 3.1

▷ Find the decryption algorithm under which Q3a_ct was encrypted and report the secret key that was used as an integer under Q3a_sk. The corresponding plaintext provides a hint for the next question.

Question 3.2

▷ Find the decryption algorithm under which Q3b_ct was encrypted and report the secret key that was used as an integer under Q3b_sk. The corresponding plaintext provides a hint for the next question.

Question 3.3

▷ Report the hidden English passphrase as a string under Q3c_pt.