Solutions for week 1, Cryptography Course - TDA 352/DIT 250

In this weekly exercise sheet: you will use some historical ciphers, the OTP, the definition of semantic security and some combinatorial problems.
Completing the ex. sheet: you will be able to prove semantic security, get familiar with the definitions, build your own personal stream cipher and develop some ideas on *why* randomness is important in Cryptography.

Easy

1. **Answers**

   (a) Eve **can not** find the Alice's original PIN without knowing the correct permutation used by Alice to create the ciphertext or additional informations about the PIN such as *"the second digit is even" or* "the last digit in 3 or 4". The only information that Eve can deduce from the ciphertext PIN 53287, and the fact that Alice is using a substitution cipher, is that the plaintext PIN has no repeating digits (can you see why?).
   If Eve tries a random (and wrong) substitution as *guess* for Alice's secret substitution (key), she would get a 5-digit number different from Alice's PIN but still a possible PIN.

   (b) Alice uses the substitution

Original	0	1	2	3	4	5	6	7	8	9
Substitution	5	3	2	9	0	1	8	4	7	6

   The encrypted PIN is 53287.
   We can now decrypt the encrypted PIN and get 01268.

2. **Questions and tasks**

   (a) The secret number is 827.
   The encryption of the GPS-coordinates are

Coor. 1	2	7	7	5	1	7	7	5
+ Secret	8	2	7	8	2	7	8	2
Encryption	0	9	4	3	3	4	5	7

Coor. 2	9	8	0	6	9	9	7	4	
+ Secret	8	2	7	8	2	7	8	2	
Encryption	7	0	7	4	1	6	5	6	mod 10

Coor. 3	2	7	5	5	1	2	9	2
+ Secret	8	2	7	8	2	7	8	2
Encryption	0	9	2	3	3	9	7	4

Coor. 4	9	9	4	5	1	8	8	2	
+ Secret	8	2	7	8	2	7	8	2	
Encryption	7	1	1	3	3	5	6	4	mod 10

   (b) The secret number is 827.
   The decryption of the encrypted GPS-coordinates are

Enc Coor. 1	0	9	4	3	3	4	5	7
- Secret	8	2	7	8	2	7	8	2
Coor. 1	2	7	7	5	1	7	7	5

Enc Coor. 2	7	0	7	4	1	6	5	6	
- Secret	8	2	7	8	2	7	8	2	
Coor. 2	9	8	0	6	9	9	7	4	mod 10

Enc Coor. 3	0	9	2	3	3	9	7	4
- Secret	8	2	7	8	2	7	8	2
Coor. 3	2	7	5	5	1	2	9	2

Enc Coor. 4	7	1	1	3	3	5	6	4	
- Secret	8	2	7	8	2	7	8	2	
Coor. 4	9	9	4	5	1	8	8	2	mod 10

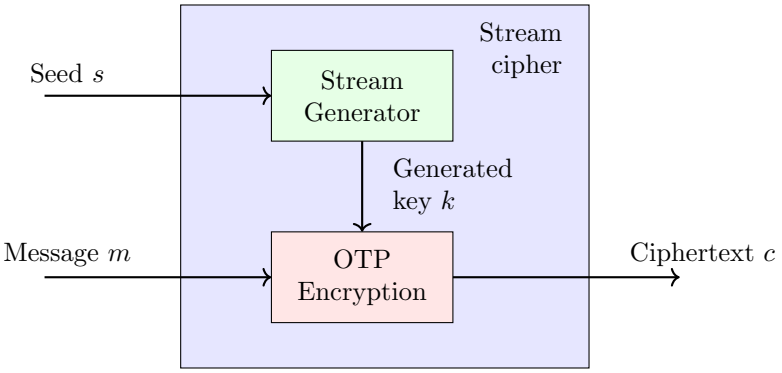
   All the decryption are correct.

3. **Answer:** To be a perfect cipher, (E, D) must satisfy the following property: for every two messages $m_0, m_1 \in \mathcal{M}$ with $\text{len}(m_0) = \text{len}(m_1)$ and every ciphertext $c \in \mathcal{C}$, it holds that $\Pr(\text{E}(k, m_1) = c) = P(\text{E}(k, m_2) = c)$ where k is chosen uniformly at random from $\mathcal{K}$.
   This implies that the adversary, Eve who sees only the ciphertext c, is not able to determine whether c is an encryption of m_0 or m_1

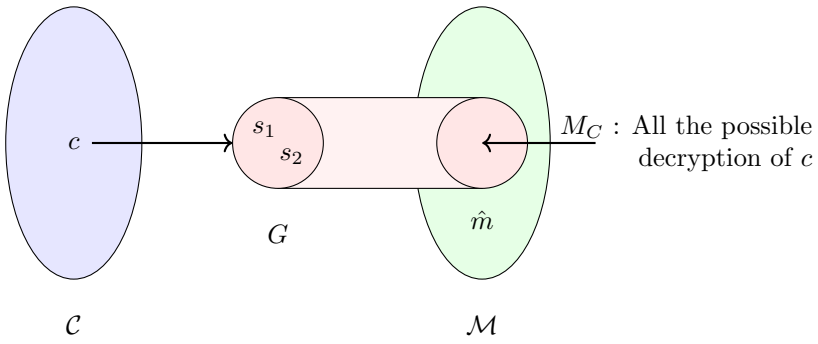
   The main goal of stream ciphers is to substitute the long key with a small seed and use this seed (and a PRG) to generate longer *encryption keys*.
   Let $\mathcal{K}$ be the set of all the possible seeds (the secret keys), $\mathcal{M}$ the message space, $\mathcal{C}$ the ciphertext space. Consider $\mathcal{G}$ the set of keys generated by a function Gen for all the different seeds $s \in \mathcal{K}$, formally $\mathcal{G} = \{\text{Gen}(s) : s \in \mathbb{K}\}$. We have that $\mathcal{G}$ is a subset of $\mathcal{M}$.
   The stream cipher works in this way:

- Choose randomly a secret seed s.
- Input the seed s in the stream cipher.
- Generate a longer key k from Gen
- Use k as a secret key in an OTP scheme, i.e., encrypt a message m by doing $m \oplus k = c$

Below you can find a diagram on how encryption with a stream cipher works:



Let's fix a ciphertext c and look at all the possible decryption of c using all the possible keys in G. Let us call this set $M_C = \{m \in \mathcal{M} : m = \mathrm{D}(G(s), c) \quad \text{for some key } s \in \mathcal{K}\}$. This set will be a subset of $\mathcal{M}$ and so there will exist a plaintext $\hat{m}$ that is not a decryption of c.
The picture below should help you visualising this:



so the probability $\Pr(\mathrm{E}(k, \hat{m}) = c)$, i.e. , the probability that a randomly chosen key k encrypts $\hat{m}$ (note that $\hat{m}$ is not in the subset M_C) in exactly the ciphertext c is equal to 0. But, if we take a message $m \in M_C$, i.e. m is a possible decryption of c, then we get $\Pr(\mathrm{E}(k, m) = c) > 0$.

This reasoning demonstrates that a stream cipher is not perfectly secret.

Remark: The main idea is formally explained in the theorem 2.10 of the Katz-Lindell which states **Theorem 2.10**: If $(\mathrm{Gen}, \mathrm{E}, \mathrm{D})$ is a perfect cipher with message-space $\mathcal{M}$ and key-space $\mathcal{K}$, then $|\mathcal{K}| \geq |\mathcal{M}|$

4. **Definition (Perfect Secrecy):**
Let $\mathcal{M}$ be the message space, $\mathcal{K}$ the key space, $\mathcal{C}$ the ciphertext space.
Let (E, D) be an encryption scheme defined over $(\mathcal{M}, \mathcal{K}, \mathcal{C})$. (E, D) has **perfect secrecy** if:

$$\forall m_0, m_1 \in \mathcal{M} \text{ such that } \mathrm{len}(m_0) = \mathrm{len}(m_1) \quad \text{and} \quad \forall c \in \mathcal{C}$$

$$\Pr[\mathrm{E}(k, m_0) = c] = P[\mathrm{E}(k, m_1) = c]$$

where k is chosen uniformly at random from $\mathcal{K}$ ($k \xleftarrow{R} \mathcal{K}$)

Informally, an encryption scheme (cipher) is perfectly secret if, without knowning the secret key k, a single ciphertext c could be the encryption of any message $m \in \mathcal{M}$. In other words, without the

knowledge of the secret key, fixed a ciphertext c, all messages have the same probability to be the corresponding plaintext. This means that there exist many different key-message pairs (k_i, m_i) for which we have $m_i = \mathrm{D}(k_i, c)$.

Not having the perfect secrecy means that we can select some candidates in the message space because they have bigger probability. This means that the ciphertext *"contains"* some information about *"which is"* the corresponding plaintext message.

5. **Definition:** One Time Pad (OTP)
The OTP has$\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0,1\}^n$ where $\mathcal{M}$ is the message-space, $\mathcal{K}$ is the key-space and $\mathcal{C}$ is the ciphertext-space.
Let $m \in \mathcal{M}$, $k \in \mathcal{K}$ and $c \in \mathcal{C}$. The OTP encryption algorithm is defined as

$$\mathrm{E}(k, m) = m \oplus k$$

The OTP decryption algorithm is defined as

$$\mathrm{D}(k, c) = c \oplus k$$

Medium

6. **Definition (Semantic Security Game)**:
Let (E, D) be a encryption scheme on $(\mathcal{M}, \mathcal{K}, \mathcal{C})$. Let $\mathcal{A}$ be an Adversary and $\mathcal{C}$ be the Challenger. We define the **semantic security game** as the game with the following procedure:

   (a) $\mathcal{A}$ challenges $\mathcal{C}$ in the semantic security game

   (b) $\mathcal{A}$ chooses m_0, m_1 with $\mathrm{len}(m_0) = \mathrm{len}(m_1)$, from some distribution D over $\mathcal{M}$

   (c) $\mathcal{A}$ sends to $\mathcal{C}$ the messages m_0 and m_1

   (d) $\mathcal{C}$ selects at random a secret key $k \xleftarrow{R} \mathcal{K}$

   (e) $\mathcal{C}$ flips a random coin, i.e. $\mathcal{C}$ picks $b \in \{0,1\}$ uniformly at random

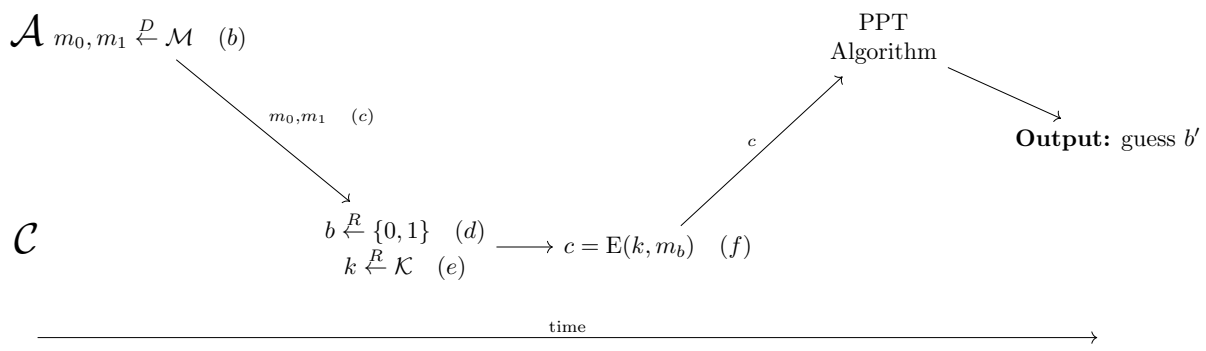
   (f) $\mathcal{C}$ encrypts m_b using k and obtains the ciphertext c, i.e. $c = \mathrm{E}(k, m_b)$

   (g) $\mathcal{C}$ sends c to $\mathcal{A}$

   (h) $\mathcal{A}$ uses a probabilistic polynomial time (PPT) algorithm to obtain a guess b' for which bit $\mathcal{C}$ chose in step (e)

   (i) $\mathcal{A}$ output his guess b'

The attacker $\mathcal{A}$ wins the semantic security game if $b = b'$.
As a diagram:



Definition (Semantic Security):
Let (E, D) be a encryption scheme on $(\mathcal{M}, \mathcal{K}, \mathcal{C})$. The encryption scheme is **semantically secure** if for all the PPT adversaries $\mathcal{A}$, it holds that

$$\Pr(b' = b) \leq \frac{1}{2} + \epsilon \qquad \text{where } \epsilon \text{ is a negligible value}$$

In order to prove the semantic security, we should either exhibit an adversary that can correctly guess b with some probability or conclude that in any case $\mathcal{A}$'s best guess is a random choice.
Remark: proofs based on games are not that easy (for everyone).
For this reason, in this exercise, I will try to highlight some tricks that can be used and will help you in understanding how to make these kind of proof *"easily"*.

Trick 1: *be the adversary.* Always start trying to be an adversary: write down **all** the steps and search for *strange* behavior studying the differences between the case $\mathcal{C}$ chooses $b = 0$ and $b = 1$.

(a) $E'(k, m) = 0 \| E(k, m)$
(i.e. prepend 0 to the ciphertext)

First, use the **trick 1**.
We choose two messages m_0, m_1 and, after the challenger's phase, we receive $c = E'(k, m_b) = 0 \| E(k, m_b)$.

- If $b = 0$, we have $c = 0 \| E(k, m_0)$
- If $b = 1$, we have $c = 0 \| E(k, m_1)$

In both of the cases, we have that the first bit is 0, so we can remove it from the ciphertext. It doesn't give us any additional information about whether c is an encryption of m_0 or m_1.
We remain with trying to find clues on $E(k, m_b)$ but (E, D) is semantic secure.
So we can conclude that E' is semantic secure.

(b) $E'(k, m) = E(k, m) \| k$

First, use the **trick 1**.
We choose two messages m_0, m_1 and, after the challenger's phase, we receive $c = E'(k, m_b) = E(k, m_b) \| k$.

- If $b = 0$, we have $c = E(k, m_0) \| k$
- If $b = 1$, we have $c = E(k, m_1) \| k$

As we can see, there are no different behavior between the two possibilities.

Trick 2: *see if the ciphertext has some strange property that leak secret information.* if something is not encrypted, it should alert you. **Always** look at the *non-encrypted* messages (or encrypted but you have the ability to decrypt them).

Using this trick on the message received ciphertext, we see that the encryption key k is always present. We can use the key k and compute[1]:

- $c_0 = E(k, m_0)$
- $c_1 = E(k, m_1)$

If the first part of c is equal to c_0, output 0. Output 1 otherwise.
We can conclude that E' is **not** semantic secure.

(c) $E'((k, k'), m) = E(k, m) \| E(k', m)$
(i.e. the key is splitted in two part, and then the two encryptions are concatenated together)

- If $b = 0$, we have $c = E(k, m_0) \| E(k', m_0)$
- If $b = 1$, we have $c = E(k, m_1) \| E(k', m_1)$

No strange behavior and no plaintext informations.

Trick 3: *sometimes is better to divide and analyze simpler problems*[2]. If a problem looks splittable in smaller problems, split them and study the smaller ones.

Lets divide the ciphertext $c = c_1 \| c_2 = E(k, m_b) \| E(k', m_b)$ and observe that if we consider just c_1 (respectively c_2), we are playing the semantic security game on (E, D), which is semantic secure.

[1] Or equivalently decrypt.
[2] You can see it as the latin proverb *"Divide et impera"* (Divide and conquer)

Trick 4: *repetitions sometimes means repeating.* Sometimes you will find really similar constructions (formulas, equations, protocol, etc...). You should look really well if they are really the same thing or not.

From the observation and the repetition *"rule"*, we can observe that playing with E' is, basically, like playing the semantic game for E twice with some special behaviours of the challenger $\mathcal{C}$:

- $\mathcal{C}$ does not change b between the games
- $\mathcal{C}$ does change the key from k to k' if k and k' are not equal

Consider the case when $k = k'$:

- If $b = 0$, we have $c = E(k, m_0) \| E(k, m_0)$
- If $b = 1$, we have $c = E(k, m_1) \| E(k, m_1)$

We can divide in the two ciphertext and observe that they are equal in this case: it is the same as playing the semantic security game with (E, D)

Taking all together, E' is semantic secure.

(d) $E'(k, m) = E(0, m)$

- If $b = 0$, we have $c = E(0, m_0)$
- If $b = 1$, we have $c = E(0, m_1)$

You can easily decrypt c since you know the key $k = 0$.
E' is not semantic secure.

(e) $E'(k, m) = E(k, m) \| \text{LSB}(m)$

- If $b = 0$, we have $c = E(k, m_0) \| \text{LSB}(m_0)$
- If $b = 1$, we have $c = E(k, m_1) \| \text{LSB}(m_1)$

From the trick 1, you can find that the $\text{LSB}(m_b)$ is *suspicious* since it can 0 or 1.
With this knowledge, we can pick two different messages m_0, m_1 in which $\text{LSB}(m_0) = 0$ and $\text{LSB}(m_1) = 1$.
If we play the semantic security game with this choice, we obtain:

- If $b = 0$, we have $c = E(k, m_0) \| 0$
- If $b = 1$, we have $c = E(k, m_1) \| 1$

We can output b' as the last bit of c and get $b' = b$.
E' is not semantic secure.

7. (a) We flip 10 times a coin that outputs 0 or 1 with probability $\frac{1}{2}$. This means that after the 10 flips, we have generated a 10-bits strings.
Since every flip can have 2 possible results and different flips are independent one from the other, flipping 10 times will have $2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^{10}$ possible results.
So we generated $2^{10} = 1024$ different strings which are exactly the number of 10-bit strings.

Since the flips are independent, the probability of getting a particular bit-string is $\left(\frac{1}{2}\right)^{10} = \frac{1}{2^{10}} = \frac{1}{1024} \sim 0,001$ or $0,1\%$.

(b) Since we flip the coin 5 times, we get $2^5 = 32$ different strings. Then, by repeating this string, we only generate 32 different 10-bit strings.
The probability of getting a particular 10-bit string is:

- The probability is almost 0 if the string is not a concatenation of two identical 5-bit string (see point (a))
- Otherwise the probability is $\frac{1}{32}$.

(c) When we consider the binary representation of a number between 0 and 1000, we are taking the 10-bit strings that will be represent that number.
For this reason, the number of keys we have is 1000 and:

- the probability of a draw a specific key is 0 if the binary string is a representation of a number between 1001 and 1023 (because these values are above 1000 and strictly below $2^{10} = 1024$)

- otherwise, the probability of a single key will be $\frac{1}{1000}$

(d) Similarly to the previous point, we get all the possible 10-bit strings. The probability of a single key will be:

- The probability is $\frac{1}{2000}$ if the random number chosen is in the range from $2000-1024+1 = 977$ to 1023
- Otherwise the probability is $\frac{2}{2000}$

8. Hope you had fun building your personal stream cipher!

9. **Suggested answer/thought**:

(a)

(b) PRGs **can** achieve the absence of **precise** patterns that we measure with tests, like the one in the exercise and other more complicated ones.
So not passing a single test does not means that the generator is not random. It just mean that *statistically* it has a pattern.
Even true random generators sometimes do not pass statistical tests, althrough they should. The reason is that their unpredictability may have some *"patterned randomness"* that appears with a certain probability.

10. **Proof:** OTP is perfect secret.
To prove that the OTP is semantic secure, you should observe that $\mathcal{M} = \mathcal{K} = \mathcal{C}$. This implies $|\mathcal{M}| = |\mathcal{K}|$.
Therefore, we just need to prove that for every two messages m_0, m_1 and for every ciphertext c, it holds that $\Pr(\mathrm{E}(k, m_0) = c) = \Pr(\mathrm{E}(k, m_1) = c)$ for some random key $k \in \mathcal{K}$.
We can rewrite $\mathrm{E}(k, m_0) = c$ as $k \oplus m_0 = c$, from which we have $k = c \oplus m_0$ (a similar reasoning hold for m_1).
Let us fix m_0 and c, then $\Pr(\mathrm{E}(k, m_0) = c) = \Pr(k = c \oplus m_0) = ($ the probability of choosing randomly the keys k such that $k = c \oplus m_0 \ \ ) = \frac{1}{|\mathcal{K}|}$.
The same reasoning (and probability) holds for m_1. This concludes the proof of the fact that the OTP encryption scheme is perfect secret.

Using the tricks explained in the solution of exercise 6:

- If $b = 0$, we have $c = k \oplus m_0$
- If $b = 1$, we have $c = k \oplus m_1$

We don't have any strange behaviour. We don't have plaintext informations. We don't have repetitions. We don't have any clue on how to divide the problem.

Trick 5: *if you not able to find anything new, try combining what you have.* A lot of times, you should look at your information from a different point of view: you should *combine what you know* and see if this tells you something new.

We know that the OTP is perfectly secure and so $\Pr(k \oplus m_0 = c) = \Pr(k \oplus m_1 = c)$ for any pair of (distinct) messages. This equality means that there exists an unique key k_0 such that $m_0 \oplus c = k_0$ and an unique k_1 such that $m_1 \oplus c = k_1$. Since m_0 and m_1 are different, also k_0 and k_1 must be different.

Going back to the exercise, we can compute $c \oplus m_0$ (where c is the ciphertext returned by the challenger $\mathcal{C}$). According to the value of b, we can distinguish the following two cases:

- $b = 0$, in this case $c \oplus m_0 = k$
- $b = 1$, in this case $c \oplus m_0 = k \oplus m_1 \oplus m_0$

Since $m_0 \neq m_1$, the two messages should be different in at least 1 bit. Without loss of generality, let us assume that the bit in which m_1 differs from m_2 is the first bit.
Lets denote the first bit of the key k as $\hat{k} \in \{0, 1\}$. Observe that the first bit of $m_1 \oplus m_0$ will be 1 since it is the bit in which they differ.
We have

- If we compute $\hat{k}$, $b' = 0$

- If we compute $\hat{k}+1$, $b'=1$

Since k is uniformly and randomly generated, we have that every bit of k will be uniformly and randomly generate (try to think using exercise 7a and the *"coin flipping"*). This means that $\hat{k}$ is uniform and random in $\{0,1\}$. So the probability $\Pr(\hat{k}=0)=\Pr(\hat{k}=1)=\frac{1}{2}$.
Since we are trying to predict the result of a random coin flip, we can conclude that our best guess for b' is flip a coin, i.e. $\frac{1}{2}$. So we (the adversary) have no non-negligible advantage.
Therefore the OTP cipher is semantically secure.

Hard

---

11. Let us recall the two definitions we need to use:
Definition: A PRG $G: K \to \{0,1\}^n$ is said to be *secure* if for any efficient statistical test (distinguisher) D, we have $Adv_{\text{PRG}}(D,G)$ is negligible.

Definition: A PRG $G: K \to \{0,1\}^n$ is said to be *predictable* if there exists an efficient algorithm A that given the first i bits of the output of G can construct the $i+1$ output of G, i.e. exists $i \in \{1,\dots,n-1\}$ such that

$$\Pr_{k \xleftarrow{R} K}[A(G(k)|_{1,\dots,i}) = G(k)_{i+1}] = \frac{1}{2} + \Delta$$

where Δ is a non-negligible term.

We want to prove the negation: predictable PRG $\implies$ PRG is insecure.

Given G is predictable, we know that there exists an algorithm A that can predict the next output of G for some index $i \in \{1,\dots,n-1\}$.

We use A to construct a distinguished D that can distinguish G from a completely random generator:

$$D(x) = \begin{cases} 1 & \text{if } A(x|_{1,\dots,i}) = x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

We have

$$Adv_{\text{PRG}}(D,G) = \left|\Pr_{r \xleftarrow{R} \{0,1\}^n}[D(r)=1] - \Pr_{k \xleftarrow{R} K}[D(G(k))=1]\right| \geq \left|\frac{1}{2} - (\frac{1}{2}+\Delta)\right| \geq \Delta$$

This proves that D has a non-negligible advantage in distinguishing the output of G from a truly random string.

12. Hill's cipher with matrices:

$$E = \begin{pmatrix} 1 & 3 & 0 \\ 3 & 1 & 2 \\ 0 & 2 & 1 \end{pmatrix} \qquad D = \begin{pmatrix} 3 & 3 & 5 \\ 3 & 10 & 2 \\ 5 & 2 & 8 \end{pmatrix}$$

$$E \cdot m = \begin{pmatrix} 1 & 3 & 0 \\ 3 & 1 & 2 \\ 0 & 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 4\cdot 1 + 0\cdot 3 + 2\cdot 0 \\ 4\cdot 3 + 0\cdot 1 + 2\cdot 2 \\ 4\cdot 0 + 0\cdot 2 + 2\cdot 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 16 \\ 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 5 \\ 2 \end{pmatrix} \mod 11$$

the encryption is $(4,5,2)$.

- **Task:** Encryption:
  (a) $E \cdot m = E \cdot (1,1,1) = (4,6,3) \mod 11$
  (b) $E \cdot m = E \cdot (7,10,3) = (4,4,1) \mod 11$

(c) $E \cdot m = E \cdot (5, 2, 7) = (0, 9, 0) \mod 11$

- **Task:** Decryption:

  (a) $D \cdot m = D \cdot (1, 1, 1) = (0, 4, 4) \mod 11$
  (b) $D \cdot m = D \cdot (7, 10, 3) = (0, 6, 2) \mod 11$
  (c) $D \cdot m = D \cdot (5, 2, 7) = (1, 5, 8) \mod 11$

13. **Suggested (and so personal) answer/thought about randomness**: this suggested answer contains a lot of different reading links and informations. If you are interest in randomness, you should read, think and discuss with other about this text.

    Testing that a string is random is really a hard task (as you can see by looking at the strange names contained in the Wikipedia's page on randomness tests) because randomness should mean *"not having patterns"*. The National Institute of Standard and Technology (NIST) has a standardized test suite that contains really complex statistical tests.
    When we talk about randomness, we always talk about the **entropy** of a system and you should look at this fancy word as *"how unpredictable the system is"*. The larger the entropy of the system, the more its unpredictability. You can imagine the connection between the two notions: **unpredictable system** and **system without pattern**.

    In applied cryptography, we mainly use randomness to generate keys or during secure communications. There are different methods to generate randomness[3] and we can subdivide them in **physical randomness** and **algorithmic pseudorandomness**:

    - The first method exploits real world physics laws. The generator is a real object that samples a specific physical signal and then encode it in a bit string. For example, a lot of computer generate entropy (and so randomness) by measuring incredibly small electro-magnetic oscillation in the space surrounding some special measurement sensors!
      These sources are incredibly random and the physics laws that they use are really well studies and so we can predict them on object with *"human-size"* dimensions. But when we look at them at the atomic level, they seems to not follow exactly the Newtonian physics laws: in the atomic dimension, we achieve to find unpredictability.
      The main problem is that these generators depends on their surrounding environment. Thus creating two equal entropy source is quite impossible.
      Never the less, random generators are used in real-life and the *price* for real randomness is really high[4]. For this reason *"true randomness"* is generally used to secure important data like government top secrets documents or banks databases.
    - The second method is *how* cryptographers construct their randomness in the cheaper way so that everyone can afford to generate *"random"* keys.
      Basically you create an algorithm, or some function, that has a fixed mathematical/programmed structure and from a particular and fixed input (the seed), you start obtaining a sequence that will be always be the same. This sequence is *pseudo*-random: you can think of it being *"kinda"* random: the string is not random to whom knows the seed. Otherwise it looks random. The reason why these generators are *cheap* is that they do not need a physical object but just some computational power[5].
      Stream ciphers are pseudo-random generators.

Think

14. (a) In general, to obtain the key in a substitution cipher, an adversary should ask for the encryption of every possible digit.
    In our case, the adversary should ask the encryption of 0123456789 which is all the possible digits.
    The challenger will reply with the substitution of every single digit.
    So the adversary can easily reconstruct the secret substitution function.

[3] Wikipedia: Random Number Generation - Generation methods
[4] For example, the IDQuantique Quantis-USB-4M module costs around 9600 sek and it's the cheapest one!
[5] You can see on wikipedia a little summary on the computation cost.

(b) Vigenère cipher are a general version of a substitution cipher, so the attack is quite similar.
In our case, the adversary should start by asking the encryption of the message 0 then 00 and so on. $\mathcal{A}$ will stop this type of queries when it receives an encryption that is 2 repetition of the same number. This means that $\mathcal{A}$ is encrypting the string of all 0s with length twice the secret key length.
This will tell the adversary the length n of the Vigenère key. On the other hand, he found even the encryption of 0 for every position in the code and since 0 is the neutral element for the sum in $\mathbb{Z}_{10}$, the adversary just received the key repeated twice!

(c) Hill's Cipher uses linear algebra and this is the real weakness of the cipher. Even OTP has the same problem.
To break Hill's cipher in our construction, the adversary should just ask the encryption of the messages $(1,0,0)$, $(0,1,0)$,$(0,0,1)$. This is the canonical basis of the message space. The encryptions are just the rows of the matrix.

15. **Proof:** Consider the case in which $m_1 \neq m_2$ and $c_1 = c_2$. If we choose randomly m_1, m_2 we have that $\Pr(M_1 = m_1 \wedge M_2 = m_2) = \Pr(M_1 = m_1)\Pr(M_2 = m_2) > 0$.
The probability $\Pr(C_1 = c_1 \wedge C_2 = c_2) = \Pr(C_1 = c_1 \wedge C_2 = c_1)$ means that we get the same ciphertext two times. In this case the conditional probability that m_1 and m_2 (with $m_1 \neq m_2$) are the original messages knowing that the received ciphertexts are equal ($c_1 = c_2$) is zero since the encryption scheme fixes the key k for the two messages, i.e., $\Pr(M_1 = m_1 \wedge M_2 = m_2 | C_1 = c_1 \wedge C_2 = c_2) = 0$.
We have

$$0 = \Pr(M_1 = m_1 \wedge M_2 = m_2 | C_1 = c_1 \wedge C_2 = c_2) \neq \Pr(M_1 = m_1 \wedge M_2 = m_2) = \Pr(M_1 = m_1)\Pr(M_2 = m_2) > 0$$

This proves that no such an encryption scheme exists.

16. **Answer:** Depends on the meaning and abstraction we want to achieve.
It is **not important** that the message length is the same if we consider just the probability distribution of the messages, keys and ciphertexts. For example in the exercise 1, we do not care if the binary representation of the digits has different length in the binary representation.
But, if we are trying to find every kind of information in order to break an encryption scheme, the lengths of the messages can give leak useful hints.
For example, in the OTP encryption scheme, encrypting a message with a specific length will give us a ciphertext with that length and so we can break security by sending two messages of different length.