Question 1)

a)

Answer: Yes you can.

Reasoning: In Order for there to be Ambiguity between the 2 possible messages the following situation would have to occur:

$Dec_{k_1}(c) = ADBE$, and $Dec_{k_2}(c) = AXBY$,

with c being the encrypted message and k_1 and k_2 being 2 different keys out of the 26 possibilities (0-25) .

This cannot happen because both possible messages have an “A” and a “B” at the same spot, which means that when shifting them by 2 different possible values k_1 and k_2 the result would need to be the same i.e.:

$Enc_{k_1}(A) = Enc_{k_2}(A)$ and $Enc_{k_1}(B) = Enc_{k_2}(B)$

This is impossible, therefore when trying all 26 possible keys you will end up with either ADBE or AXBY but not both.

b)

Answer: No you can not.

Reasoning: Take the possible keys $k_1 = AV$ and $k_2 = AB$ for example.

This leads to the following situation:

$Enc_{k_1}(ADBE) = AYBZ$, and $Enc_{k_2}(AXBY) = AYBZ$

Therefore it is possible to end up with both possible messages when trying all possible keys.

Question 2

(2 Points). Assume that a server uses a poly-alphabetic substitution cipher (Lecture 1, slide 22-
a key length of $\ell = 2$ and the alphabet $\Sigma = \{A, \dots, Z\}$. You have the possibility to get the encryp
of one message of arbitrary length of your choice.

- Which message do you pick in order to fully recover the key?

I choose the message `AABBCC...XXYY`. The resulting ciphertext will have its first two letters be $\pi_1(A)\pi_2(A)$, the next two be $\pi_1(B)\pi_2(B)$ etc, which allows us to recover all values of π_1 and π_2. Note that the value of $\pi_i(Z)$ follows from the other values of π_i, since π_i must be bijective.

- How long does your message have to be at least?

At least 50 ($25 \cdot 2$) letters. If there are any less, at least two values of one of the π_i do not appear in the ciphertext, and we have no way of knowing what π_i maps them to (i.e. the ciphertext is the same if they are, say, swapped or left the same).

Question 3

(2 Points). Provide a specification of the Gen, Enc, and Dec algorithms for the Scytale cipher (L
1, slide 16) over the alphabet $\Sigma = \{A, \ldots, Z\}$ and the message space $\mathcal{M} = \Sigma^n$, i.e., messages of fixed
length n. How you specify the three algorithms is up to you (e.g. textual description, pseudocod
as long as the specification is complete.

Gen: In a “physical” implementation, this would involve picking a stick with a specific diameter
a computer implementation, we would pick an arbitrary number between 1 and some maximu
length (which should be longer than any message).

Enc: In a “physical” implementation, we would wrap the cloth around the stick and write our m
laterally. In a computer implementation, we would first demand that the message length n is d
by k (possibly padding with zeros if necessary). Then, we intersperse the letters of the message
following way:

$$\text{Enc}_k(m_1, \ldots, m_n) = (m_1, m_{\frac{n}{k}+1}, m_{2\frac{n}{k}+1}, \ldots, m_{n-\frac{n}{k}+1}, m_2, m_{\frac{n}{k}+2}, \ldots)$$

Dec: “Physically”, we would just wrap the cloth around a stick of the same diameter and read t
message. In a computer, we undo the interspersing in the following way:

$$\text{Dec}_k(c_1, \ldots, c_n) = (c_1, c_{k+1}, c_{2k+1}, \ldots, c_{n-k+1}, c_2, c_{k+2}, \ldots)$$

Question 3 (2 Points). Provide a specification of the **Gen**, **Enc**, and **Dec** algorithms for the Scytale cipher (Lecture 1, slide 16) over the alphabet $\Sigma = \{\mathtt{A}, \ldots, \mathtt{Z}\}$ and the message space $\mathcal{M} = \Sigma^n$, i.e., messages of fixed length n. How you specify the three algorithms is up to you (e.g. textual description, pseudocode, ...), as long as the specification is complete.

(Q3) Theo Jauchling

```
Gen(n):
    return randInt(1, n)

Enc_k(m):                                   if m[x], x ≥ m.length, returns ε
  cols ← ⌈m.length / k⌉
    A ← Matrix(k × cols)
  for r in range(k):
       for c in range(cols):
            A[r][c] ← m[r·cols + c]
  c ← Array(cols·k)
  for a in range(cols):
       for r in range(k):
            c[a·k + r] ← A[r][a]
  return c

Dec_k(c):
  cols ← c.length / k
  A ← Matrix(k × cols)
  m ← Array(c.length)
  for a in range(cols):
       for r in range(k)
            A[r][a] ← c[a·k + r]
  for r in range(k):
       for a in range(cols):
            m[r·cols + a] ← A[r][a]
  return m, without ε
```