Classical Ciphers

Part 2

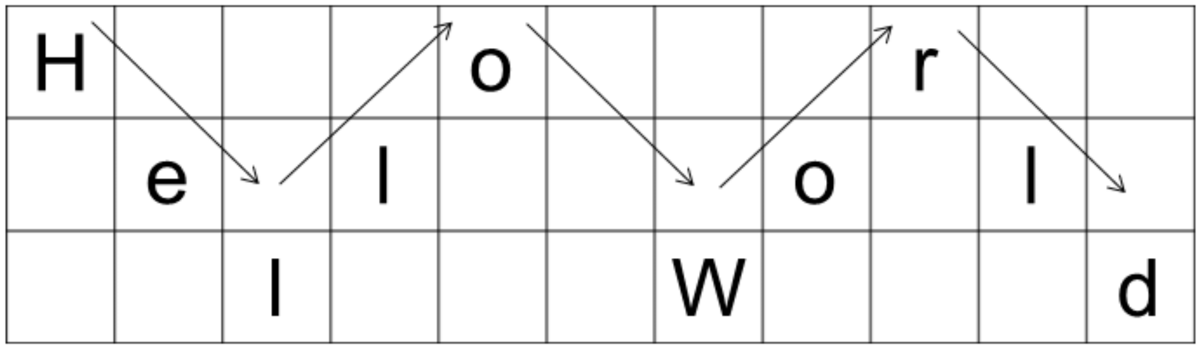
Content

Content
Railfence Cipher
Playfair Cipher
Autokey Cipher
Hill Cipher

Railfence Cipher

- Write the plaintext in a zig-zag pattern that runs over a number of rails.
- If there is no offset, start from the top rail.
- If there is offset, skip some positions before writing.

Original Message: Hello World



Encrypted Message: Horel ollWd

Railfence Cipher

- Example: encrypt the message “THIS MESSAGE WAS ENCRYPTED WITH A TRANSPOSITION CIPHER”
- No offset: “TSAYIAIIHESWSRPWTRNSTCPIMAEECTDHTSOINHRSGNEAPOE”

```
T-----S-----A-----Y-----I-----A-----I-----I----
-H---E-S---W-S---R-P---W-T---R-N---S-T---C-P---
--I-M---A-E---E-C---T-D---H-T---S-O---I-N---H-R
---S-----G-----N-----E-----A-----P-----O-----E-
```

- Offset = 5: “HSSPTNTPTISAAEYTIHASIIIHSEGWNREWARPSOCEMECDTONR”

```
•-----H-----S-----S-----P-----T-----N-----T-----P---
-•---T-I---S-A---A-E---Y-T---I-H---A-S---I-I---I-H--
--•-•---S-E---G-W---N-R---E-W---A-R---P-S---O-C---E-
---•-----M-----E-----C-----D-----T-----O-----N-----R
```

Railfence Cipher

TASK: Write the railfence encryption function that takes a plaintext, number of rails and an optional offset value

Algorithm 47: Railfence Cipher Encryption Algorithm

```
Input: plaintext, num_rails offset
Output: ciphertext
Initialize ciphertext as a list of lists with size num_rails × (len(plaintext) + offset),
  filled with "-" values;
tmp_offset = offset;
rail = 0;
move = 1;
for i = 0 to len(plaintext) + offset − 1 do
    if tmp_offset > 0 then
        Set ciphertext[rail][i] = "#";
        Decrease tmp_offset by 1;
        if rail == num_rails − 1 then
            Change direction by multiplying move by −1;
        end
        Move to the next rail by updating rail with rail + move;
        if rail == 0 then
            Change direction by multiplying move by −1;
        end
        Continue the loop;
    end
    Set ciphertext[rail][i] = plaintext[i − offset];
    if rail == num_rails − 1 then
        Change direction by multiplying move by −1;
    end
    Move to the next rail by updating rail with rail + move;
    if rail == 0 then
        Change direction by multiplying move by −1;
    end
end
Return ciphertext;
```

Railfence Cipher

TASK: Write the railfence decryption function

Algorithm 48: Railfence Cipher Decryption

Input: $ciphertext$, num_rails, $offset$
Output: $plaintext$
Initialize $plaintext$;
Set $rail = 0$, $move = 1$;
for $i = 0$ **to** $len(ciphertext[0]) - 1$ **do**
    Copy the current rail's list into tmp;
    Append $tmp[i]$ to $plaintext$;
    if $rail == num\_rails - 1$ **then**
        Change direction by multiplying $move$ by -1;
    end
    Update $rail$ by $rail + move$;
    if $rail == 0$ **then**
        Change direction by multiplying $move$ by -1;
    end
end
return $plaintext$;

Railfence Cipher

TASK: Write two functions: one that prints the ciphertext and the other for printing the plaintext

Algorithm 49: Print Ciphertext

Input: *ciphertext*
Set $ctxt = ""$;
for $i = 0$ **to** $len(ciphertext) - 1$ **do**
    Set $tmp =$ concatenate the current ciphertext block;
    Append tmp to $ctxt$;
    Print tmp;
end
Remove "-" and "#" from $ctxt$
Print $ctxt$;

Algorithm 50: Print Plaintext

Input: *plaintext*
Remove "-" and "#" from $plaintext$ Print $ptxt$;

Content

Content
Railfence Cipher
Playfair Cipher
Autokey Cipher
Hill Cipher

Playfair Cipher

- Playfair is a *digram substitution cipher.*
  - Substitutes two letters at a time.

- If the plaintext contains two identical adjacent letters, we put X between them.

- If the number of characters in the plaintext is odd, we need to add X at the end.

Playfair Cipher

- Steps:

1. Represent the secret key as a 5*5 square.
    1. Fill the cells it with alphabet, j and i are combined in one cell.

2. The plaintext is processed two letters at a time:
    1. If both the letters are in the same column: Take the letter <u>below</u> each one.
    2. If both the letters are in the same row: Take the letter to the <u>right</u> of each one.
    3. If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the <u>horizontal opposite</u> corner of the rectangle.

Playfair Cipher

- Example: encrypt the plaintext “MESSAGE” with the key “Polybius”

Playfair Cipher

- Example: encrypt the plaintext “MESSAGE” with the key “POLYBIUS”

1. Generate the key square:

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

Playfair Cipher

- Example: encrypt the plaintext “MESSAGE” with the key “POLYBIUS”

1. Generate the key square:

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

2. Split the message into two-letters block: ME SX SA GE

Playfair Cipher

- Ciphertext: VM

ME SX SA GE

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

Playfair Cipher

- Ciphertext: VM AW

ME SX SA GE

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

- Notice: the columns are swapped

Playfair Cipher

- Ciphertext: VM AW AC

ME SX SA GE

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

Playfair Cipher

- Ciphertext: VM AW AC HF

ME SX SA GE

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

Playfair Cipher

- To decrypt, reverse the operations

1. If both the letters are in the same column: Take the letter above each one.
2. If both the letters are in the same row: Take the letter to the left of each one.
3. If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Playfair Cipher

VM AW AC HF

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

- Plaintext:

Playfair Cipher

VM AW AC HF

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

- Plaintext: ME

Playfair Cipher

VM AW AC HF

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

- Plaintext: ME SX

Playfair Cipher

VM AW AC HF

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

- Plaintext: ME SX SA

Playfair Cipher

VM AW AC HF

P	O	L	Y	B
I/j	U	S	A	C
D	E	F	G	H
K	M	N	Q	R
T	V	W	X	Z

- Plaintext: ME SX SA GE

Playfair Cipher

- There is no specific way to recover the original plaintext.
  - o We cannot tell whether an X is part of the message or a filler letter.
  - o We cannot tell whether a letter “I” is an “I” or ”J”.

- Infer the original message from the decrypted ciphertext by reading it.

Playfair Cipher

TASK: Write a function that prepares the plaintext before encryption:

1. Remove any non-alphabetical characters and convert to uppercase
    1. `text = re.sub(r'[^A-Za-z]', '', text).upper()`
2. Replace “J” with “I”
    1. `text = text.replace('J', 'I')`
3. Insert ”X” between identical letters in the same diagraph
    1. `text = re.sub(r'(.)\1', r'\1X\1', text)`
4. If the length of the text is odd, append “X” to it
    1.
    ```
    if len(text) % 2 != 0:
        text += 'X'
    ```

Playfair Cipher

TASK: Write a function that takes a key string and convert it to a 5x5 array

Algorithm 51: Generate Key Square

Input: *key*
Output: 5x5 Key Square matrix
$key \leftarrow PrepareTextkey$;
$key\_square \leftarrow []$;
$used\_letters \leftarrow \{\}$;
for $l \in [A : Z]$ *(excluding J)* **do**
  if $l \notin used\_letters$ **then**
    Append l to key_square;
    Add l to $used\_letters$;
  end
end
for $i \leftarrow 0$ **to** 4 **do**
  $matrix[i] \leftarrow key\_square[i \times 5 : (i+1) \times 5]$;
end
return $matrix$;

Playfair Cipher

TASK: Write a function that takes the key array and a letter and returns the index (row, col) of the encrypted letter according to the key array

Algorithm 52: Find Position of a Letter in Key Square

```
Input: key_square, letter
Output: (row, col)
for row ← 0 to 4 do
    if letter ∈ key_square[row] then
        col ← index of letter in key_square[row];
        return (row, col);
    end
end
return None
```

Playfair Cipher

TASK: Write a function that takes the key array and a diagraph and returns an encrypted diagraph

Algorithm 53: Encrypt Digraph

Input: key_square, Digraph (a, b)
Output: Encrypted Digraph (a', b')
$(row_a, col_a) \leftarrow \texttt{FindPosition}(key\_square, a)$;
$(row_b, col_b) \leftarrow \texttt{FindPosition}(key\_square, b)$;
if $row_a = row_b$ **then**
  $a' \leftarrow key\_square[row_a][(col_a + 1) \mod 5]$;
  $b' \leftarrow key\_square[row_b][(col_b + 1) \mod 5]$;
else if $col_a = col_b$ **then**
  $a' \leftarrow key\_square[(row_a + 1) \mod 5][col_a]$;
  $b' \leftarrow key\_square[(row_b + 1) \mod 5][col_b]$;
else
  $a' \leftarrow key\_square[row_a][col_b]$;
  $b' \leftarrow key\_square[row_b][col_a]$;
end
return (a', b');

Playfair Cipher

TASK: Write a function that takes the key array and an encrypted diagraph and returns a decrypted diagraph

Algorithm 54: Decrypt Digraph

Input: key_square, Digraph (a, b)
Output: Decrypted Digraph (a', b')
$(row_a, col_a) \leftarrow$ `FindPosition`(key_square, a);
$(row_b, col_b) \leftarrow$ `FindPosition`(key_square, b);
if $row_a = row_b$ **then**
  $a' \leftarrow key\_square[row_a][(col_a - 1) \mod 5]$;
  $b' \leftarrow key\_square[row_b][(col_b - 1) \mod 5]$;
else if $col_a = col_b$ **then**
  $a' \leftarrow key\_square[(row_a - 1) \mod 5][col_a]$;
  $b' \leftarrow key\_square[(row_b - 1) \mod 5][col_b]$;
else
  $a' \leftarrow key\_square[row_a][col_b]$;
  $b' \leftarrow key\_square[row_b][col_a]$;
end
return (a', b');

Playfair Cipher

TASK: Implement an encryptor for the Playfair cipher.

Algorithm 55: Encrypt Playfair Cipher

Input: Plaintext *plaintext*, Key string *key*
Output: Ciphertext *ciphertext*

$key\_square \leftarrow \texttt{GenerateKeySquare}(key)$;
$plaintext \leftarrow \texttt{PrepareText}(plaintext)$;
$ciphertext \leftarrow$ empty string;
for $i \leftarrow 0$ **to** $\texttt{length}(plaintext) - 1$ **by** 2 **do**
  $digraph \leftarrow plaintext[i] + plaintext[i+1]$;
  $encrypted\_digraph \leftarrow \texttt{EncryptDigraph}(key\_square, digraph)$;
  $ciphertext \leftarrow ciphertext + encrypted\_digraph$;
end
return $ciphertext$;

Playfair Cipher

TASK: Implement a decryptor for the Playfair cipher.

Algorithm 56: Decrypt Playfair Cipher

Input: Ciphertext *ciphertext*, Key string *key*
Output: Decrypted Plaintext *plaintext*
$key\_square \leftarrow$ `GenerateKeySquare`(key);
$plaintext \leftarrow$ empty string;
for $i \leftarrow 0$ **to** $\mathtt{length}(ciphertext) - 1$ **by** 2 **do**
  $digraph \leftarrow ciphertext[i] + ciphertext[i+1]$;
  $decrypted\_digraph \leftarrow$ `DecryptDigraph`$(key\_square, digraph)$;
  $plaintext \leftarrow plaintext + decrypted\_digraph$;
end
return $plaintext$;

Content

Content
Railfence Cipher
Playfair Cipher
Autokey Cipher
Hill Cipher

Autokey Cipher

- It uses a key stream that begins with a keyword followed by the plaintext.
- The key-stream characters are added to plaintext characters, modulo 26.
- Example: encrypt the message “ENCRYPTED” using keyword “AUTOKEY”.

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

A	U	T	O	K	E	Y

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

A	U	T	O	K	E	Y

2. Compute “E” + “A” % 26 → “E”

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

A	U	T	O	K	E	Y	E

2. Compute "E" + "A" % 26 → "E"
3. Remove the first key from the vector and append the first plaintext character to the key vector.

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

A	U	T	O	K	E	Y	E

2. Compute “E” + “A” % 26 → “E”
3. Remove the first key from the vector and append the first plaintext character to the key vector.
4. Repeat the steps by computing each two corresponding characters and appending the plaintext to the key stream.

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

~~A~~	U	T	O	K	E	Y	E

- Ciphertext = E

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

~~A~~	U	T	O	K	E	Y	E

- Ciphertext = EH

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D
~~A~~	~~U~~	T	O	K	E	Y	E	N

- Ciphertext = EH

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

~~A~~	~~U~~	T	O	K	E	Y	E	N

- Ciphertext = EHV

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D	
~~A~~	~~U~~	~~T~~	O	K	E	Y	E	N	C

- Ciphertext = EHV

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D	
~~A~~	~~U~~	~~T~~	O	K	E	Y	E	N	C

- Ciphertext = EHVF

Autokey Cipher

1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

~~A~~	~~U~~	~~T~~	~~O~~	K	E	Y	E	N	C	R	…

- Ciphertext = EHVFITRIQ

Autokey Cipher

- Decryption uses the same way; instead of adding the letters, subtract them.
- Example: decrypt “EHVFITRIQ” using the keyword “AUTOKEY”

E	H	V	F	I	T	R	I	Q

A	U	T	O	K	E	Y

- Plaintext: E

Autokey Cipher

- Decryption uses the same way; instead of adding the letters, subtract them.
- Example: decrypt “EHVFITRIQ” using the keyword “AUTOKEY”

~~E~~	H	V	F	I	T	R	I	Q

~~A~~	U	T	O	K	E	Y	E

- Plaintext: E

Autokey Cipher

- Decryption uses the same way; instead of adding the letters, subtract them.
- Example: decrypt "EHVFITRIQ" using the keyword "AUTOKEY"

~~E~~	H	V	F	I	T	R	I	Q

~~A~~	U	T	O	K	E	Y	E

- Plaintext: EN

Autokey Cipher

- Decryption uses the same way; instead of adding the letters, subtract them.
- Example: decrypt “EHVFITRIQ” using the keyword “AUTOKEY”

~~E~~	~~H~~	V	F	I	T	R	I	Q

~~A~~	~~U~~	T	O	K	E	Y	E	N

- Plaintext: EN

Autokey Cipher

- Decryption uses the same way; instead of adding the letters, subtract them.
- Example: decrypt “EHVFITRIQ” using the keyword “AUTOKEY”

~~E~~	~~H~~	V	F	I	T	R	I	Q

~~A~~	~~U~~	T	O	K	E	Y	E	N	…

- Plaintext: ENCRYPTED

Autokey Cipher

TASK: Write a function to encipher a plaintext with an autokey cipher.

Algorithm 57: Autokey Encryption

Input: Plaintext, key
Output: Ciphertext
$ciphertext \leftarrow []$;
$key\_index \leftarrow 0$;
for $i = 0$ **to** $length(plaintext) - 1$ **do**
  if $plaintext[i]$ *is alphabetic* **then**
    $k \leftarrow key[key\_index] -$ 'A';
    $cipher\_char \leftarrow (plaintext[i] -$ 'A' $+ k) \mod 26 +$ 'A';
    Append $cipher\_char$ to $ciphertext$;
    $key \leftarrow key + plaintext[i]$;
    $key\_index \leftarrow key\_index + 1$;
  end
  else
    Append $plaintext[i]$ to $ciphertext$;
  end
end
return $ciphertext$;

Autokey Cipher

TASK: Write a function to decipher a ciphertext with an autokey cipher.

Algorithm 58: Autokey Decryption

```
Input: Ciphertext, key
Output: Plaintext
plaintext ← [];
key_index ← 0;
for i = 0 to length(ciphertext) − 1 do
    if ciphertext[i] is alphabetic then
        k ← key[key_index]− 'A';
        plain_char ← (ciphertext[i]− 'A' −k)  mod 26+ 'A';
        Append plain_char to plaintext;
        key ← key + plain_char;
        key_index ← key_index + 1;
    end
    else
        Append ciphertext[i] to plaintext;
    end
end
return plaintext;
```

Content

Content
Railfence Cipher
Playfair Cipher
Autokey Cipher
Hill Cipher

Hill Cipher

- A block cipher that uses $n \times n$ matrix mult. to encrypt each block of n letters.
  - If the last block is not aligned with the block size, a padding is added.
  - A padding can be the letter X.
  - Multiplication is done modulo 26

- Decryption works by
  1. Finding the inverse of the key matrix
  2. Multiply the inverse matrix by the ciphertext blocks.

Hill Cipher

- Example: encrypt the message "ENCRYPTED" using the matrix
    - The block size is 3

$$\mathbf{M} = \begin{pmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{pmatrix}$$

Hill Cipher

- Example: encrypt the message “ENCRYPTED” using the matrix
  - The block size is 3

$$\mathbf{M} = \begin{pmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{pmatrix}$$

1. Split the message into blocks: ENC RYP TED

Hill Cipher

- Example: encrypt the message “ENCRYPTED” using the matrix
  - The block size is 3

$$\mathbf{M} = \begin{pmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{pmatrix}$$

1. Split the message into blocks: ENC RYP TED
2. Represent each block as a column vector

$$ENC = \begin{bmatrix} 69 \\ 78 \\ 67 \end{bmatrix}, \qquad RYP = \begin{bmatrix} 82 \\ 89 \\ 80 \end{bmatrix}, \qquad TED = \begin{bmatrix} 84 \\ 69 \\ 68 \end{bmatrix}$$

Hill Cipher

3. Multiply the key matrix by the column vector:

$$ENC = \begin{bmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{bmatrix} \times \begin{bmatrix} 4 \\ 13 \\ 2 \end{bmatrix} = \begin{bmatrix} 17 \\ 1 \\ 2 \end{bmatrix} = RBC$$

$$RYP = \begin{bmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{bmatrix} \times \begin{bmatrix} 17 \\ 24 \\ 15 \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \\ 15 \end{bmatrix} = IEP$$

$$TED = \begin{bmatrix} 7 & 8 & 11 \\ 11 & 2 & 8 \\ 15 & 7 & 4 \end{bmatrix} \times \begin{bmatrix} 19 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 16 \\ 20 \\ 13 \end{bmatrix} = QUN$$

Hill Cipher

TASK: Write a function to compute the inverse of number w.r.t the modulus 26

Algorithm 59: Modular Inverse of a Number Mod 26

Input: a
Output: Modular inverse of a mod 26

```
for i = 1 to 26 do
    if (a × i) mod 26 == 1 then
        return i;
    end
end
return Error
```

Hill Cipher

TASK: Write a function to compute the inverse of a matrix with elements modulo 26. (use $Numpy.linalg$ module to compute the determinant and inverse of a matrix)

Algorithm 60: Modular Inverse of a Matrix

Input: Matrix $matrix$

Output: Inverse of $matrix$ mod 26

$det \leftarrow |matrix|$;

$det\_inv \leftarrow$ `mod_inv_num(det)`;

$matrix\_adj \leftarrow$ `adj(matrix) % 26`;

$matrix\_inv \leftarrow (det\_inv \times matrix\_adj) \mod 26$;

return $matrix\_inv$;

Hill Cipher

TASK: Implement the Hill Cipher encryption. Allow for any block size. Verify that the matrix is invertible before encryption.

Algorithm 61: Hill Cipher Encryption

```
Input: Plaintext, Key matrix key_matrix
Output: Encrypted Text
mod_inv(key_matrix);
n ← key_matrix.shape[0];
Remove non-alphabetical characters from plaintext ;
if size(plaintext) mod n ≠ 0 then
  | Pad plaintext with 'X'
end
Encode plaintext as integers;
Divide plaintext into blocks, each of size n;
encrypted_text ← [ ];
for block in plaintext_blocks do
  | Reshape block into a vector or size n × 1
  |  encrypted_block ← dot_product(key_matrix, block_matrix) % 26;
  | Flatten the encrypted block and append it to encrypted_text
end
Decode encrypted_text into characters;
return encrypted_text;
```

Hill Cipher

TASK: Implement the Hill Cipher decryptor. Allow for any block size. Remember to find its inverse of the key matrix.

Algorithm 62: Hill Cipher Decryption

Input: Ciphertext, Key matrix *key_matrix*
Output: Decrypted Text
$n \leftarrow key\_matrix.shape[0]$;
Encode *ciphertext* as integers;
Divide *ciphertext* into blocks, each of size n;
key_inv ← `mod_inv(key_matrix)`;
decrypted_text ← [ ] ;
for *block in cipher_blocks* **do**
  Reshape *block* into a vector or size $n \times 1$
  decrypted_block ← `dot_product(key_inv, block_matrix) % 26`;
  Flatten the decrypted block and append it to *decrypted_text*;
end
Decode *decrypted_text* into characters;
return *decrypted_text*;