Cryptography

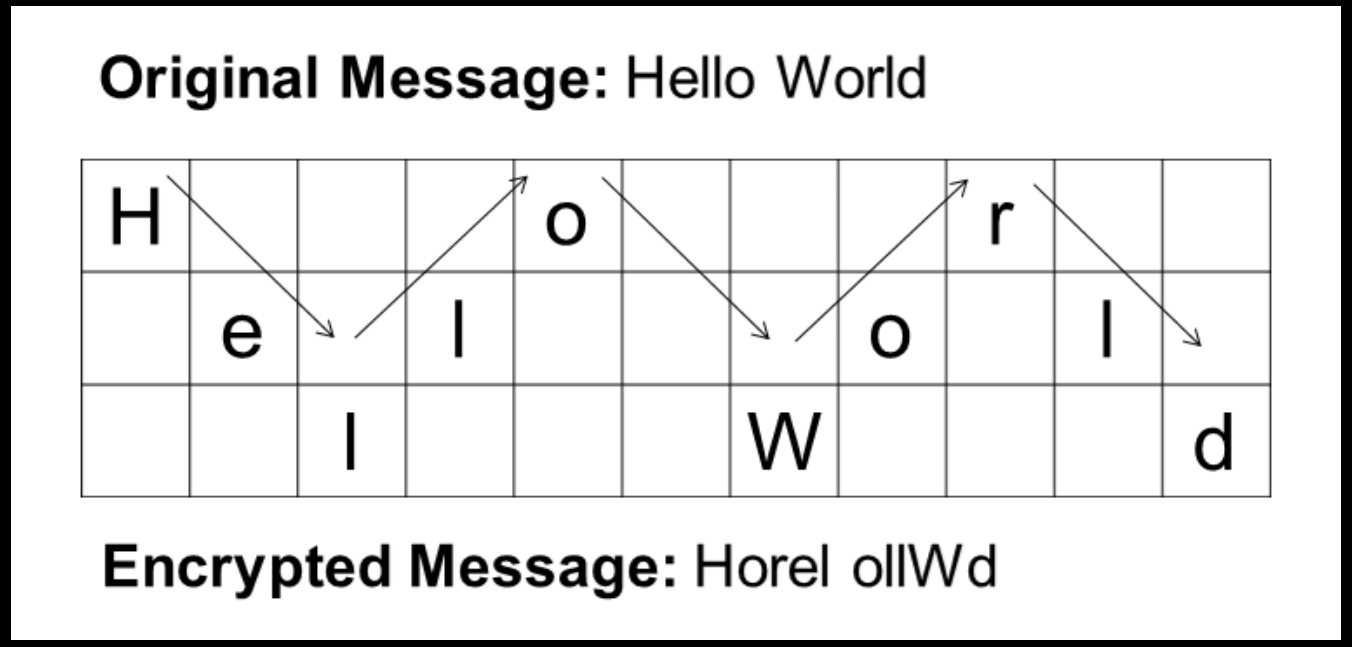
Classical Ciphers PT2

Content

Content
Railfence Cipher
Playfair Cipher
Autokey 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.



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 Encryption

Input: $plaintext$, num_rails, $offset$
Output: $ciphertext$
$ciphertext \leftarrow$ a list of size $num\_rails \times (len(plaintext) + offset)$, initialized to "−" ;
$tmp\_offset \leftarrow offset$;
$rail \leftarrow 0$;
$move \leftarrow 1$;
for $i = 0$ **to** $len(plaintext) + offset - 1$ **do**
  $tmp \leftarrow$ copy of $ciphertext[rail]$;
  if $tmp\_offset > 0$ **then**
    $tmp[i] \leftarrow$ #;
    $ciphertext[rail] \leftarrow tmp$;
    $tmp\_offset \leftarrow tmp\_offset - 1$;
    if $rail == num\_rails - 1$ **then**
      $move \leftarrow move \times (-1)$;
    end
    $rail \leftarrow rail + move$;
    if $rail == 0$ **then**
      $move \leftarrow move \times (-1)$;
    end
    Continue loop;
  end
  $tmp[i] \leftarrow plaintext[i - offset]$;
  $ciphertext[rail] \leftarrow tmp$;
  if $rail == num\_rails - 1$ **then**
    $move \leftarrow move \times (-1)$;
  end
  $rail \leftarrow rail + move$;
  if $rail == 0$ **then**
    $move \leftarrow move \times (-1)$;
  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**
    $tmp \leftarrow$ copy of $ciphertext[rail]$;
    $plaintext = plaintext + tmp[i]$;
    if $rail == num\_rails - 1$ **then**
        $move = move \times -1$;
    end
    $rail = rail + move$;
    if $rail == 0$ **then**
        $move = move \times -1$;
    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

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 below each one.
    2. If both the letters are in the same row: Take the letter to the right 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

- 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.
  - We cannot tell whether an X is part of the message or a filler letter.
  - 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 (string)
Output: key_matrix (5x5 matrix)
key ← prepare_text(key);
key_square ← [];
used_letters ← empty set;
foreach letter in key do
    if letter ∉ used_letters then
        key_square.append(letter);
        used_letters.add(letter);
    end
end
foreach letter in [A:Z]-'J' do
    if letter ∉ used_letters then
        key_square.append(letter);
    end
end
key_matrix ← [key_square[i * 5 : (i + 1) * 5] for i in range(5)];
return key_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')
$(row_a, col_a) \leftarrow$ `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: 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 ← `GenerateKeySquare`(*key*);
plaintext ← `PrepareText`(*plaintext*);
ciphertext ← empty string;
for $i \leftarrow 0$ **to** *length*(*plaintext*) $- 1$ **by** 2 **do**
    digraph ← *plaintext*[*i*] + *plaintext*[*i* + 1];
    encrypted_digraph ← `EncryptDigraph`(*key_square*, *digraph*);
    ciphertext ← *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 ← `GenerateKeySquare`(*key*);
plaintext ← empty string;
for *i* ← 0 **to** `length`(*ciphertext*) − 1 **by** 2 **do**
    digraph ← *ciphertext*[*i*] + *ciphertext*[*i* + 1];
    decrypted_digraph ← `DecryptDigraph`(*key_square*, *digraph*);
    plaintext ← *plaintext* + *decrypted_digraph*;
end
return *plaintext*;

Content

Content
Railfence Cipher
Playfair Cipher
Autokey 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

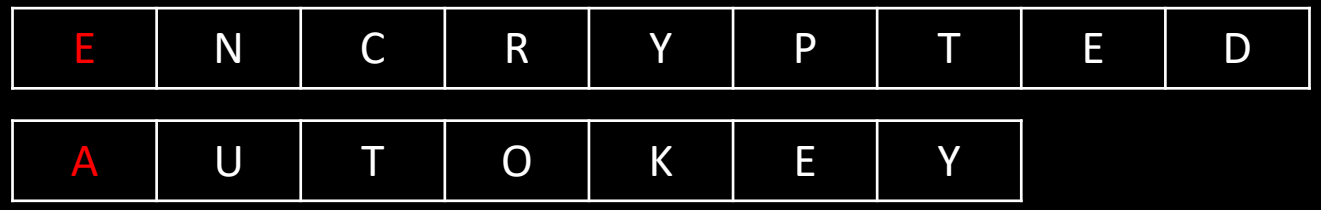
1. Represent the plaintext and the key as vectors:

E	N	C	R	Y	P	T	E	D

A	U	T	O	K	E	Y

Autokey Cipher

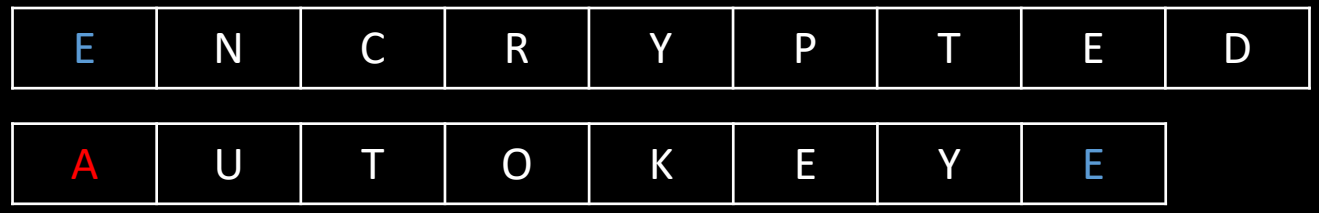
1. Represent the plaintext and the key as vectors:



2. Compute “E” + “A” % 26 → “E”

Autokey Cipher

1. Represent the plaintext and the key as vectors:



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:

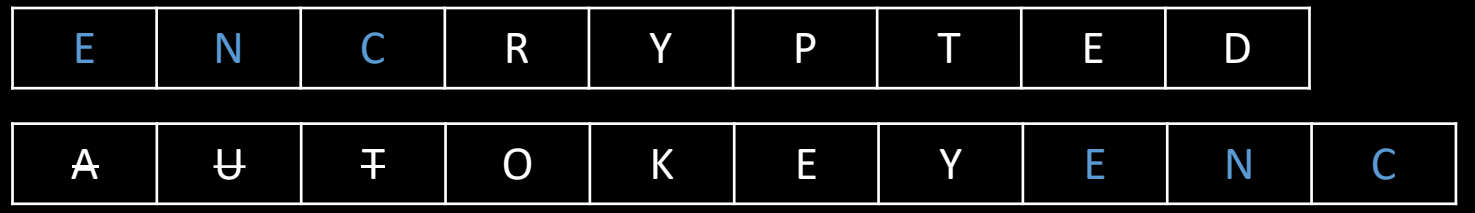
E	N	C	R	Y	P	T	E	D

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

- Ciphertext = EHV

Autokey Cipher

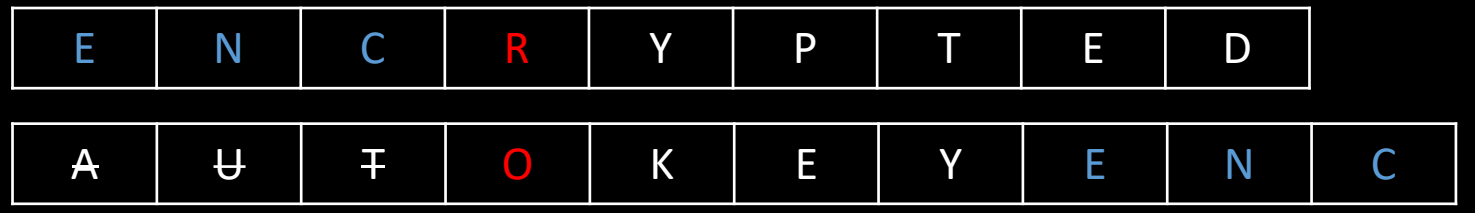
1. Represent the plaintext and the key as vectors:



- Ciphertext = EHV

Autokey Cipher

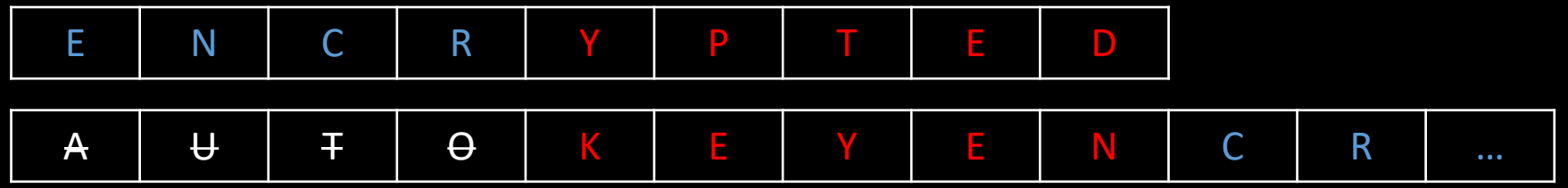
1. Represent the plaintext and the key as vectors:



- Ciphertext = EHVF

Autokey Cipher

1. Represent the plaintext and the key as vectors:



- 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 ← [];
key_index ← 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
$plaintext \leftarrow []$;
$key\_index \leftarrow 0$;
for $i = 0$ **to** $length(ciphertext) - 1$ **do**
    if $ciphertext[i]$ *is alphabetic* **then**
        $k \leftarrow key[key\_index] -$ 'A';
        $plain\_char \leftarrow (ciphertext[i] -$ 'A' $-k)$ mod 26 + 'A';
        Append $plain\_char$ to $plaintext$;
        $key \leftarrow key + plain\_char$;
        $key\_index \leftarrow key\_index + 1$;
    end
    else
        Append $ciphertext[i]$ to $plaintext$;
    end
end
return $plaintext$;