

MIT World Peace University

Information and Cyber Security

Assignment 3

NAMAN SONI ROLL No. 10

Contents

1	Aim	2
2	Objectives	2
3	Theory	2
3.1	<i>Explain Simplified Advanced Encryption Standard (S-AES) algorithm.</i>	2
4	Programming Language Used	2
5	Code	2
6	FAQ's	9

1 Aim

Write a program using JAVA or Python or C++ to implement S-AES symmetric key algorithm.

2 Objectives

To understand the concept of block cipher and symmetric key cryptographic system.

3 Theory

3.1 *Explain Simplified Advanced Encryption Standard (S-AES) algorithm.*

Simplified Advanced Encryption Standard (S-AES) is a symmetric encryption algorithm that provides secure and reliable data encryption. It is based on the Advanced Encryption Standard (AES) algorithm, but has been modified to have simpler and highly optimized rounds of encryption process.

The S-AES algorithm works by accepting a single input key, which is then split into two parts. This key is used to create multiple intermediate keys, which are then used for the encryption process. These intermediate keys are then combined after the encryption process is complete, making sure that the encryption key is completely randomized each time.

All of these keys are combined using an XOR operation in order to produce the cipher text. This algorithm has been proven to be very secure and is considered to be one of the best and most reliable encryption algorithms available.

4 Programming Language Used

Python

5 Code

```
1  import sys binary_to_decimal = { (0, 0): 0, (0, 1): 1, (1, 0): 2, (1, 1):3 }
2
3  s_box =[[0x9, 0x4, 0xA, 0xB],
4  [0xD, 0x1, 0x8, 0x5],
5  [0x6, 0x2, 0x0, 0x3],
6  [0xC, 0xE, 0xF, 0x7],
7  ]inv_s_box =[[0xA, 0x5, 0x9, 0xB],
8  [0x1, 0x7, 0x8, 0xF],
9  [0x6, 0x0, 0x2, 0x3],
10 [0xC, 0x4, 0xD, 0xE],
11 ]R_CON =[[1, 0, 0, 0, 0, 0, 0, 0],
12 [0, 0, 1, 1, 0, 0, 0, 0],
13 [0, 0, 0, 0, 1, 1, 0, 0],
14 [0, 0, 0, 0, 0, 0, 1, 1],
15 ]MIX_COLUMN_TABLE = {
16     1:[0x 0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE,
17     0xF],
18     2:[0x0, 0x2, 0x4, 0x6, 0x8, 0xA, 0xC, 0xE, 0x3, 0x1, 0x7, 0x5, 0xB, 0x9, 0xF,
19     0xD],
20     4:[0x0, 0x4, 0x8, 0xC, 0x3, 0x7, 0xB, 0xF, 0x6, 0x2, 0xE, 0xA, 0x5, 0x1, 0xD,
21     0x9],
22     9:[0x0, 0x9, 0x1, 0x8, 0x2, 0xB, 0x3, 0xA, 0x4, 0xD, 0x5, 0xC, 0x6, 0xF, 0x7,
23     0xE],
24 }
25
26 MIX_COLUMN_MATRIX =[[1, 4],
27 [4, 1]] MIX_COLUMN_MATRIX_DECRYPT =[[9, 2],
28 [2, 9]] def caesar_cipher (plain_text, key):
29     """ "Function to encrypt plain text using Ceaser Cipher.
```

```

30
31 Args:
32 plain_text (string): plain text to be encrypted.
33 key (int): key to be used for encryption.
34 """ def get_ascii (some_char):
35 if some_char
36 .islower ():
37 return ord (some_char) - 97 elif some_char.isupper ():
38 return ord (some_char) - 65
39 else
40 :
41 return -1 cipher_letter = "" cipher = [] for i
42 in plain_text:
43 if i
44 == " " or not i.isalpha ():
45 cipher.append (i) continue if i
46 .islower ():
47 cipher_letter = chr (((get_ascii (i) + key) % 26) + 97).upper ()
48 else
49 :
50 cipher_letter = chr (((get_ascii (i) + key) % 26) + 65).lower () cipher.append (
    cipher_letter) return cipher def decrypt_caesar_cipher (cipher_text, caesar_key):
51
52 """ Function to decrypt cipher text using Caesar Cipher.
53
54 Args:
55 cipher_text (string): cipher text to be decrypted.
56 caesar_key (int): key to be used for decryption.
57 """ def get_ascii (some_char):
58 if some_char
59 .islower ():
60 return ord (some_char) - 97 elif some_char.isupper ():
61 return ord (some_char) - 65
62 else
63 :
64 return -1 plain_letter = "" plain_text = [] for i
65 in cipher_text:
66 if i
67 == " " or not i.isalpha ():
68 plain_text.append (i) continue if i
69 .islower ():
70 plain_letter =
71 chr (((get_ascii (i) - caesar_key) % 26) + 97).upper ()
72 else
73 :
74 plain_letter = chr (((get_ascii (i) - caesar_key) % 26) + 65).lower () plain_text.append (
    plain_letter) return "".join (plain_text) def decimal_to_binary (ip_val, reqBits):
75 """ Function to convert decimal to binary. Returns a list that has integers 0 and 1
76 represented in binary.
77 Args:
78 ip_val (_type_): input_value in decimal.
79 reqBits (_type_): required number of bits in the output. 4, 8, etc. """ def
    decimalToBinary_rec (ip_val, list):
80 if ip_val
81 >=1:
82 #recursive function call
83 decimalToBinary_rec (ip_val // 2, list) list.append(ip_val % 2)
84 list = [] decimalToBinary_rec (ip_val, list) if len (list) < reqBits:
85 while len (list) < reqBits: list.insert (0, 0) if len (list) > reqBits: list.pop (0)
    return list def nibble_substitution_encrypt (nibble):
86 """ Performs and returns substitution of nibble using S-Box.
87 Args:
88 nibble (list of integers 0 and 1): nibble to be substituted.
89 """ s_box_row_num = binary_to_decimal.get ((nibble[0], nibble[1])) s_box_col_num =
    binary_to_decimal.get ((nibble[2], nibble[3])) nibble_after_s_box = s_box[s_box_row_num
    ][s_box_col_num] nibble_after_s_box = decimal_to_binary (nibble_after_s_box, 4) return
    nibble_after_s_box def nibble_substitution_decrypt (nibble):
90 """ Performs and returns substitution of nibble using S-Box.

```

```

91 Args:
92 nibble (list of integers 0 and 1): nibble to be substituted.
93 " " s_box_row_num = binary_to_decimal.get ((nibble[0], nibble[1])) s_box_col_num =
    binary_to_decimal.get ((nibble[2], nibble[3])) nibble_after_s_box = inv_s_box[
        s_box_row_num][s_box_col_num] nibble_after_s_box = decimal_to_binary (nibble_after_s_box
        , 4) return nibble_after_s_box def key_expansion_function_g (key_w, round_number):
94 #divide into 2 parts. N0, and N1
95 n_0 = key_w[: 4] n_1 = key_w[4:]
96 #Perform nibble substitution on N0 and N1
97 n_0_after_s_box =
98 nibble_substitution_encrypt (n_0)
99 n_1_after_s_box =
100 nibble_substitution_encrypt (n_1)
101 #XOR N0 and N1 with RCON
102 sub_nib = n_1_after_s_box + n_0_after_s_box return [x ^ y for x, y in zip (sub_nib, R_CON[
    round_number])] def make_keys (key):
103 " "
104 key = 16 bits.
105 " " key_w0, key_w1, key_w2, key_w3, key_w4, key_w5 =
106 (0, 0, 0, 0, 0, 0)
107 #divide the key into 2 parts. key_w0 and key_w1
108 key_w0 = key[: 8] key_w1 = key[8: ] key_w1_after_g = key_expansion_function_g (key_w1, 0)
    key_w2 = [x ^ y for x, y in zip (key_w0, key_w1_after_g)] key_w3 = [x ^ y for x, y in zip
        (key_w1, key_w2)] key_w3_after_g = key_expansion_function_g (key_w3, 1) key_w4 = [x ^ y
        for x, y in zip (key_w2, key_w3_after_g)] key_w5 = [x ^ y for x, y in zip (key_w3, key_w4
        )] return key_w0 + key_w1, key_w2 + key_w3, key_w4 + key_w5 def col_matrix_table_lookup
        (x, y):
109 " " Returns the result of multiplication of x and y in GF(2^8) using MIX_COLUMN_TABLE.
110 Args:
111 x (int): first number to be multiplied. y (int): second number to be multiplied.
112 " " answer = MIX_COLUMN_TABLE.get (y)[x] return decimal_to_binary (int (answer), 4) def
    mix_columns (s_matrix, mix_column_matrix):
113 #returns a 16 bit answer.
114 result_matrix = [[[0, 0, 0, 0],
115 [0, 0, 0, 0]], [[0, 0,
116 0, 0],
117 [0, 0,
118 0,
119 0]],
120 ]
121 #clearly, multiplication by another 2d matrix while seemingly easy, doesnt work for
122 some reason.
123 #So we will take advantage of the fact that this is a SIMPLIFIED AES cipher, and do it
124 manually.
125 #multiply 2 dimensional matrices
126 #for k in range(len(mix_column_matrix)):
127 #for i in range(len(mix_column_matrix[0])):
128 #for j in range(len(mix_column_matrix[0])): # table_lookup = col_matrix_table_lookup(
129 #int("".join([str(i) for i in s_matrix[k][j]]), base=2), # mix_column_matrix[i][k],
130 #)
131 #result_matrix[i][j] = [
132 #x ^ y for x, y in zip(result_matrix[i][j], table_lookup)
133 #]
134 #1st row, 1st column
135 #table_lookup(value, mat[0][0]) ^ table_lookup(s[0][1], mat[1][0])
136 table_lookup_left =
137 col_matrix_table_lookup (int
138 ("".
139 join ([str
140 (i)
141 for i
142 in
143 s_matrix
144 [0]
145 [0]]),
146 base =
147 2),
148 mix_column_matrix

```

```

149 [0][0],
150
151
152 )table_lookup_right =
153 col_matrix_table_lookup (int
154 ("".
155 join ([str
156 (i)
157 for i
158 in
159 s_matrix
160 [1]
161 [0]]),
162 base = 2),
163 mix_column_matrix
164 [0][1],)
165 result_matrix[0][0] =
166 [x ^ y for x,
167 y in zip (table_lookup_left,
168 table_lookup_right)]
169 #1st row, 1st column
170 #table_lookup(value, mat[0][0]) ^ table_lookup(s[0][1], mat[1][0])
171 table_lookup_left =
172 col_matrix_table_lookup (int
173 ("".
174 join ([str
175 (i)
176 for i
177 in
178 s_matrix
179 [0]
180 [1]]),
181 base = 2),
182 mix_column_matrix
183 [0][0],)
184 table_lookup_right =
185 col_matrix_table_lookup (int
186 ("".
187 join ([str
188 (i)
189 for i
190 in
191 s_matrix
192 [1]
193 [1]]),
194 base = 2),
195 mix_column_matrix
196 [0][1],)
197 result_matrix[0][1] =
198 [x ^ y for x,
199 y in zip (table_lookup_left,
200 table_lookup_right)]
201 #1st row, 1st column
202 #table_lookup(value, mat[0][0]) ^ table_lookup(s[0][1], mat[1][0])
203 table_lookup_left =
204 col_matrix_table_lookup (int
205 ("".
206 join ([str
207 (i)
208 for i
209 in
210 s_matrix
211 [0]
212 [0]]),
213 base = 2),
214 mix_column_matrix
215 [1][0],)
216 table_lookup_right =

```

```

217 col_matrix_table_lookup (int
218 ("".
219 join ([str
220 (i)
221 for i
222 in
223 s_matrix
224 [1]
225 [0]]),
226 base = 2),
227 mix_column_matrix
228 [1][1],)
229 result_matrix[1][0] =
230 [x ^ y for x,
231 y in zip (table_lookup_left,
232 table_lookup_right)]
233 #1st row, 1st column
234 #table_lookup(value, mat[0][0]) ^ table_lookup(s[0][1], mat[1][0])
235 table_lookup_left =
236 col_matrix_table_lookup (int
237 ("".
238 join ([str
239 (i)
240 for i
241 in
242 s_matrix
243 [0]
244 [1]]),
245 base = 2),
246 mix_column_matrix
247 [1][0],)
248 table_lookup_right =
249 col_matrix_table_lookup (int
250 ("".
251 join ([str
252 (i)
253 for i
254 in
255 s_matrix
256 [1]
257 [1]]),
258 base = 2),
259 mix_column_matrix
260 [1][1],)
261 result_matrix[1][1] =
262 [x ^ y for x,
263 y in zip (table_lookup_left,
264 table_lookup_right)]
265 return (result_matrix[0][0] +
266 result_matrix[1][0]
267 #no idea why im shifting this and the next line + result_matrix [0][1]
268 + result_matrix[1][1]) def encrypt_SAES_cipher (plain_text, key):
269 key_0, key_1, key_2 =
270 make_keys (key)
271 #round 0 - Only Add round key
272 round_0 =
273 [x ^ y for x,
274 y in zip (plain_text, key_0)]
275 #STARTING ROUND 1
276 #Making nibbles
277 s_0, s_1, s_2, s_3 = (round_0[: 4], round_0[4: 8], round_0[8: 12], round_0[12:])
278 s_0_after_sub = nibble_substitution_encrypt (s_0) s_1_after_sub =
279 nibble_substitution_encrypt (s_1)
280 s_2_after_sub =
281 nibble_substitution_encrypt (s_2)
282 s_3_after_sub =
283 nibble_substitution_encrypt (s_3)
284 #Shifting Rows, exchanging s1 ands s3

```

```

284 s_1_after_sub, s_3_after_sub =
285 s_3_after_sub, s_1_after_sub
286 #Mixing Columns
287 s_matrix =
288 [[s_0_after_sub, s_2_after_sub],
289 [s_1_after_sub,
290 s_3_after_sub]] mix_col_result =
291 mix_columns (s_matrix,
292 MIX_COLUMN_MATRIX) round_1
293 =
294 [x ^ y for x,
295 y in zip (mix_col_result, key_1)]
296 #STARTING ROUND 2
297 s_0, s_1, s_2, s_3 = (round_1[: 4], round_1[4: 8], round_1[8: 12], round_1[12:])
298 s_0_after_sub = nibble_substitution_encrypt (s_0) s_1_after_sub =
299 nibble_substitution_encrypt (s_1)
300 s_2_after_sub =
301 nibble_substitution_encrypt (s_2)
302 s_3_after_sub =
303 nibble_substitution_encrypt (s_3)
304 #Shifting Rows, exchanging s1 and s3
305 s_1_after_sub, s_3_after_sub = s_3_after_sub, s_1_after_sub s_box = s_0_after_sub +
306 s_1_after_sub + s_2_after_sub + s_3_after_sub round_2 =[x ^ y for x, y in zip (s_box,
307 key_2)] return round_2 def decrypt_SAES_cipher (cipher_text, key):
308 key_0, key_1, key_2 =
309 make_keys (key)
310 #round 0 - Only Add round key
311 round_0 =
312 [x ^ y for x,
313 y in zip (cipher_text, key_2)]
314 #STARTING ROUND 1
315 #Inverse nibbles substitution
316 s_0, s_1, s_2, s_3 = (round_0[: 4], round_0[4: 8], round_0[8: 12], round_0[12:])
317 s_0_after_sub = nibble_substitution_decrypt (s_0) s_1_after_sub =
318 nibble_substitution_decrypt (s_1)
319 s_2_after_sub =
320 nibble_substitution_decrypt (s_2)
321 s_3_after_sub =
322 nibble_substitution_decrypt (s_3)
323 #Inverse Shifting Rows, exchanging s1 and s3
324 s_1_after_sub, s_3_after_sub =
325 s_3_after_sub, s_1_after_sub nib_sub =
326 s_0_after_sub + s_1_after_sub +
327 s_2_after_sub + s_3_after_sub
328 #Add Round key
329 round_1 =[x ^ y for x, y in zip (nib_sub, key_1)] s_0, s_1, s_2, s_3 = (round_1[: 4],
330 round_1[4: 8], round_1[8: 12], round_1[12:])
331 #Inverse Mixing Columns
332 s_matrix =[[s_0, s_2],[s_1, s_3]]
333 round_1 =
334 mix_columns (s_matrix,
335 MIX_COLUMN_MATRIX_DECRYPT)
336 #STARTING ROUND 2
337 #making nibbles
338 s_0, s_1, s_2, s_3 = (round_1[: 4], round_1[4: 8], round_1[8: 12], round_1[12:])
339 #Inverse Shifting Rows, exchanging s1 and s3
340 s_1, s_3 = s_3, s_1
341 #Inverse nibbles substitution
342 s_0_after_sub = nibble_substitution_decrypt (s_0) s_1_after_sub =
343 nibble_substitution_decrypt (s_1) s_2_after_sub = nibble_substitution_decrypt (s_2)
344 s_3_after_sub = nibble_substitution_decrypt (s_3) s_box = s_0_after_sub + s_1_after_sub
345 + s_2_after_sub + s_3_after_sub round_2 =[x ^ y for x, y in zip (s_box, key_0)] return
346 round_2 def main ():
347 plain_text =
348 input
349 ("Enter Text to be encrypted via S-AES:")
350 key =
351 input

```



```

342 ("Enter 4 digit Key to be used for encryption:")
343 #Make keys
344 ceaser_key = 0 for i in key[: 2]:ceaser_key += int (i)
345 key =
346 [decimal_to_binary (int (i), 4) for i
347 in key] key =
348 [j for i in key for j in i]
349 ceaser_ciphered_text =
350 ceaser_cipher (plain_text, ceaser_key)
351 #make plain_text list of 16 bits
352 plain_text =[decimal_to_binary (ord (i), 8) for i in ceaser_ciphered_text] plain_text =[j
    for i in plain_text for j in i] plain_texts =[plain_text[i: i + 16] for i in range (0,
    len (plain_text), 16)] for i in plain_texts:
353 if len (i) < 16:
354 i +=[0 for i in range (16 - len (i))] ciphers =[for plain_text in plain_texts:
355 cipher_text =
356 encrypt_SAES_cipher (plain_text,
357 key) ciphers.
358 append (cipher_text) final_cipher_text
359 = ""
360 #decrypting
361 for cipher in ciphers:
362 cipher =[str (i) for i in cipher] cipher =["".join (cipher[i:i + 8]) for i in range (0,
363 len (cipher),
364 8)] cipher =
365 [chr (int (i, base = 2)) for i in
366 cipher if i != "00000000"] cipher =
367 "".join (cipher) final_cipher_text +=
368 cipher print ("Your Cipher Text is: ",
369 final_cipher_text)
370 final_decrypted_text = ""
371 #decrypting
372 for cipher in ciphers:
373 plain_text = decrypt_SAES_cipher (cipher, key) plain_text =[str (i) for i in plain_text]
    plain_text =["".join (plain_text[i:i + 8]) for i in range (0,
374 len
375 (plain_text),
376 8)] plain_text
377 =
378 [chr (int (i, base = 2)) for i in
379 plain_text if i !=
380 "00000000"] plain_text =
381 ""
382 join (plain_text) final_decrypted_text
383 +=
384 decrypt_ceaser_cipher (plain_text,
385 ceaser_key)
386 print ("The decrypted plain text is: ",
387 final_decrypted_text)
388 #plain_text = [1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
389 #key = [0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1]
390 #print("The plain text is: ", plain_text)
391 #print("The key is: ", key)
392 ## till here we are good. now we need to encrypt the plain text.
393 #cipher_text = encrypt_SAES_cipher(plain_text , key)
394 #print("The cipher text is: ", cipher_text)
395 ## DECRYPTING
396 #plain_text = decrypt_SAES_cipher(cipher_text , key) # print("The decrypted plain text is:
    ", plain_text)
397 main ()

```

Listing 1: Input Code

6 FAQ's

1. Differentiate between DES and AES.

Ans. *DES Encryption:* The Data Encryption Standard, often known as DES, is a symmetric key block cypher developed by IBM in 1977.

- Plaintext is divided into two halves in DES encryption, and then DES uses a 64-bit plaintext and a 56-bit key to generate a 64-bit ciphertext, which is an encrypted representation of the data.
- The key length used for encryption in DES is 56 bits, although the block size is 64 bits (the remaining 8 bits are check bits only; they are not used by the encryption algorithm). DES entails 16 rounds of identical procedures, regardless of key length.
- The key length used for encryption in DES is 56 bits, although the block size is 64 bits (the remaining 8 bits are check bits only; they are not used by the encryption algorithm). DES entails 16 rounds of identical procedures, regardless of key length.

DES is a symmetric key algorithm used to encrypt digital data. Its short key length of 56 bits makes it too weak to secure most current applications that is based on encryption.

AES Encryption: Advanced Encryption Standard, or AES, is a symmetric key block cipher developed by Vincent Rijmen and Joan Daemen in 2001. AES is implemented worldwide, both in hardware and software, to encrypt sensitive data. AES is widely used while transmitting data over computer networks, particularly in wireless networks.

- AES uses a 128-bit plaintext and a 128-bit secret key to create a 128-bit block, which is then processed to produce 16 bytes (128-bit) ciphertext.
- In the case of AES, the key length might be 128 bits, 192 bits, or 256 bits, with 10 rounds (128 bits), 12 rounds (192 bits), or 14 rounds (256 bits).
- AES, on the other hand, is more secure than DES encryption and has become the de facto international standard.

The encryption process of Advanced Encryption Standard is based upon substitution and permutation operations in iterative manner. The 16 bytes of data are arranged in a matrix of four columns and four rows. On this matrix, AES performs rounds of substitution-permutation operations. Each of these rounds uses a different cipher key, which is calculated from the original AES key. The number of rounds of operations depends upon the size of the key in the following manner –

- For 128-bit cipher key, 10 rounds
- For 192-bit cipher key, 12 rounds
- For 256-bit cipher key, 14 rounds

2. What are the different advantages and Limitations of AES?

Ans. *Advantages of AES Encryption:*

- **Fast:** The Advanced Encryption Standard (AES) is much faster and efficient than its predecessors and can be implemented in both hardware and software.
- **Secure:** There has been extensive research into the AES algorithm, and it is now considered very secure. It is widely used by governments and other organizations to protect sensitive data.
- **Widely Supported:** AES is widely supported and is included in many software libraries, making it easy to implement.

Limitations of AES Encryption:

- **Key Length:** AES only supports a limited set of key lengths, which means it may not be suitable for some high-security applications.
- **Hardware Requirements:** For maximum security, AES requires specialized hardware, which can be expensive and difficult to obtain.