

تکلیف دوم

مهدی حقوردی

۹ اردیبهشت ۱۴۰۳

فهرست مطالب

۱	۱
۳	۲
۳	۱۰۲ توضیح تصاویر
۴	۲۰۲ پاسخ‌ها
۴	۳
۴	۴ پیوست‌ها
	۱

با توجه به سیستم رمزنگاری DES به سوالات زیر پاسخ دهید.

(آ) تعداد کل عملیات‌های xor را بدست آورید.

از آنجایی که DES یک ساختار فیستلی ۱۶ دوری است، در بیرون از تابع F، ۱۶ تا xor قرار دارد. و چون درون تابع F پس از عملیات extend یک بار با کلید xor انجام می‌گیرد پس اینجا هم ۱۶ تا عملیات xor داریم و در مجموع ۳۲ عملیات xor.

(ب) هدف از s-box ها را بنویسید.

نوشتن رابطه‌ی جبری برای بیت‌های خروجی بر حسب بیت‌های ورودی و کلید به دلیل وجود s-box بسیار دشوار است.

(ج) پیچیدگی حمله‌ی جست‌وجوی جامع به این سیستم از چه مرتبه‌ای می‌باشد؟

کلید DES، ۶۴ بیتی است که ۸ بیت آن بیت‌های parity هستند پس کلید مخفی آن تنها ۵۶ بیت طول دارد ← جست‌وجوی کامل در DES از مرتبه‌ی 2^{56} است.

(د) دلیل استفاده از expansion s-box در DES Function چیست؟

کلید ۵۶ بیتی DES توسط Key Scheduler به ۱۶ کلید ۴۸ بیتی تبدیل می‌شود و از آنجایی که طول بلاک DES ۶۴ بیت است و در ساختار فیستل تنها ۳۲ بیت آن به داخل تابع F می‌رود باید ۳۲ بیت ورودی را به ۴۸ بیت گسترش بدهیم تا بتوانیم آن را با کلید xor کنیم.

ه) اگر خروجی سیستم رمزنگاری به یک سیستم رمزنگاری دیگر داده شود، چه تغییری در امنیت آن حاصل می‌شود؟ (double des) اگر این کار سه بار تکرار شود چطور؟ (triple des)

• double des

در این حالت برای شکستن می‌توان از حمله‌ی تطابق در میانه استفاده کرد که مرتبه‌ی آن از 2^{112} به 2^{57} تقلیل می‌یابد.

• triple des

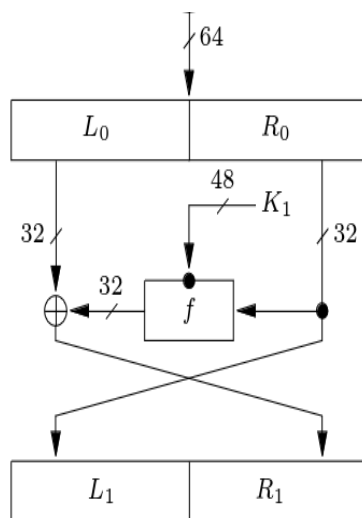
در این حالت هم (با استفاده از حمله‌ی تطابق در میانه) مرتبه بجای 2^{168} می‌شود: 2^{112} که البته در عمل قابل انجام نیست. در سال ۲۰۱۷ NIST منسوخ شدن 3DES را اعلام کرد.

و) ویژگی مکمل بودن این سیستم را ثابت کنید و توضیح دهید در آن صورت حمله به این سیستم از چه مرتبه‌ایست و چرا؟

خاصیت مکمل بودن DES:

$$DES_K(M) = C \Rightarrow DES_{\bar{K}}(\bar{M}) = \bar{C} \quad (1)$$

(برای اثبات فقط یک دور را در نظر می‌گیریم) با توجه به ساختار فیستلی DES ورودی ابتدا به دو قسمت تقسیم می‌کند و سپس نیمه‌ی راست را (درون تابع F) با کلید xor می‌کند و سپس خروجی را با قسمت سمت چپ xor می‌کند.



که یعنی:

$$\begin{cases} P = L_0 \cdot R_0 \\ L_1 = R_0 \\ B = f(R_0 \oplus K_1) \\ R_1 = L_0 \oplus B \end{cases} \Rightarrow C = L_1 \cdot R_1 \quad (2)$$

حال اگر P و K را not کنیم:

$$\begin{cases} \overline{P} = \overline{L_0.R_0} \\ L_1 = \overline{R_0} \\ B = f(\overline{R_0} \oplus \overline{K_1}) \\ R_1 = \overline{L_0} \oplus B \end{cases} \Rightarrow \overline{C} = \overline{L_1.R_1} \quad (3)$$

پس در نتیجه:

$$E_k(P) = C \iff E_{\overline{k}}(\overline{P}) = \overline{C} \quad (4)$$

اگر تحت حمله‌ی chosen-plaintext attack به DES باشیم، با توجه به ویژگی مکمل بودن آن، ما می‌توانیم وقتی $E_k(P) = C$ را از DES عبور دادیم و C را گرفتیم، بار دوم بدون محاسبه‌ی مجدد $E_{\overline{k}}(\overline{P}) = \overline{C}$ را داشته باشیم. پس پیچیدگی شکستن آن نصف شده و به 2^{55} می‌رسد.

۲

با استفاده از یک کلید رمز واحد، هر یک از تبدیلات زیر را بر متن آشکار که تنها در بیت اول با هم تفاوت دارند، اعمال کنید. تعداد بیت‌های تغییر یافته پس از هرتبديل را پیدا کنید. هرتبديل را بطور مستقل اعمال کنید. در مورد اثر بهمنی پس از هرتبديل بطور مستقل و سپس اثر بهمنی پس از اعمال یک راند توضیح دهید.

برای نوشتن این سوال هر عملیات AES را در پایتون پیاده سازی کردم که source code آن در پوشه‌ی AES همراه تکلیف ارسال شده است. پاسخ هر بخش در تصویری که جلوی شما نوشته شده است نوشته شده است.

۱.۲ توضیح تصاویر

```
ON: Op Name
P1:0000000000
C1:1000000001
    ^      ^

P2:1000000000
C2:0111001110
    ^^^^  ^^^

C1:1000000001
C2:0111001110
    ^^^^  ^^^

ON: Op Name
```

Changed bit No.	Bit change ratio
Cnt	8
ratio	80%

اول از همه نام عملیات در بالای تصویر نوشته شده است،

سپس متن آشکار و متن تغییر یافته و کاراکترهایی که تغییر یافته‌اند نشان داده شده‌اند،

دوباره همین کار روی متن آشکاری که بیت اول آن فرق کرده است تکرار شده است،

سپس تفاوت‌های بین دو متن تغییر یافته نوشته شده،

و در آخر در جدولی تعداد بیت‌های تغییر یافته و درصد تغییر یافتن متن رمز شده‌ی دوم نوشته شده است.

۲.۲ پاسخ‌ها

آ SB: Sub Byte تصویر ۱(آ)

ب SR: Shift Row تصویر ۱(ب)

ج MC: Mix Columns تصویر ۱(ج)

د ARK: Add Round Key تصویر ۱(د)

ه FR: Full Round تصویر ۱(ه)

همانطور که مشاهده شد، اثر بهمنی در عملیات‌های مختلف درصد کمی داشته و در یک دور (آن هم در این مورد خاص که اولین بیت تغییر کرده است) به ۱۸% رسید. اگر ما با کلیدی ۱۲۸ بیتی و عملیات کامل رمزنگار AES که شامل ۱۶ دور است، (طبق مستندات) اثر بهمنی به نزدیک حداکثر آن، یعنی ۵۰% می‌رسد.

۳

از بین مدهای عملیاتی ECB، CBC، OFB، CFB و CTR در کدام یک امکان افزایش سرعت در عمل رمزگذاری با استفاده از parallel processing یا پردازش موازی وجود دارد؟
مدهای: CTR • ECB

۴ پیوست‌ها

```
1 """This module implements 4 operations present in AES
2 - SB: Sub Bytes
3 - SR: Shift Rows
4 - MC: Mix Columns
5 - ARK: Add Round Key
6 - FR: Full Round (on of the 16 rounds)
7 """
8
9 from functools import wraps
10 from inspect import getfullargspec
11 from itertools import islice
12 from typing import TypeAlias
13
14 import galois
15
16 __all__ = ['sub_bytes', 'isub_bytes', 'shift_row', 'mix_column',
17           'add_round_key', 'full_round']
```

```

18 BitMat: TypeAlias = list[list[int]]
19 GF128 = galois.GF(2, 8, irreducible_poly='x^8 + x^4 + x^3 + x +
    1')
20
21
22 def batched(iterable, n):
23     # batched('ABCDEFGG', 3) -> ABC DEF G
24     it = iter(iterable)
25     while batch := tuple(islice(it, n)):
26         yield batch
27
28
29 def stream_to_matrix(stream: str) -> BitMat:
30     """Make the stream a matrix converted as an integers
31
32     Arguments:
33         stream: b0b1b2...b15
34
35     Return:
36         [[b0, b4, b8, b12],
37          [b1, b5, b9, b13],
38          [b2, b6, b10, b14],
39          [b3, b7, b11, b15]]
40     """
41     ranges = []
42     s = 0
43     for i in range(0, 128 // 8):
44         e = s + 8
45         ranges.append((s, e))
46         s = e
47
48     mat = [[], [], [], []]
49     for r in batched(ranges, 4):
50         for idx, ra in enumerate(r):
51             s, e = ra
52             _stream = int(stream[s:e], 2)
53             mat[idx].append(_stream)
54     return mat
55
56
57 def matrix_to_stream(matrix: BitMat) -> str:
58     """Make the matrix a stream
59
60     Arguments:
61         matrix: [[b0, b4, b8, b12],
62                 [b1, b5, b9, b13],

```

```

63         [b2, b6, b10, b14],
64         [b3, b7, b11, b15]]
65
66     Return:
67         'b0b1b2b3...b15'
68     """
69     stream = ''
70     for col in range(4):
71         _stream = ''
72         for row in matrix:
73             _stream += f'{bin(row[col])[2:]:0>8}'
74         stream += _stream
75     return stream
76
77
78 def type_and_len_check(func):
79     @wraps(func)
80     def wrapper(*args):
81         argnames = getfullargspec(func).args
82         for arg, name in zip(args, argnames):
83             if not isinstance(arg, str):
84                 raise TypeError(f'{name!r} should be `str`')
85
86         for arg, name in zip(args, argnames):
87             if not len(arg) == 128:
88                 raise ValueError(f'{name!r} should 128 bits')
89
90         return func(*args)
91
92     return wrapper
93
94
95     ##### SB: Sub Bytes ##### # noqa: E266
96     sb_mat = GF128(
97         [[1, 0, 0, 0, 1, 1, 1, 1],
98          [1, 1, 0, 0, 0, 1, 1, 1],
99          [1, 1, 1, 0, 0, 0, 1, 1],
100         [1, 1, 1, 1, 0, 0, 0, 1],
101         [1, 1, 1, 1, 1, 0, 0, 0],
102         [0, 1, 1, 1, 1, 1, 0, 0],
103         [0, 0, 1, 1, 1, 1, 1, 0],
104         [0, 0, 0, 1, 1, 1, 1, 1]],
105     )
106
107     isb_mat = GF128(
108         [[0, 0, 1, 0, 0, 1, 0, 1],

```

```

109         [1, 0, 0, 1, 0, 0, 1, 0],
110         [0, 1, 0, 0, 1, 0, 0, 1],
111         [1, 0, 1, 0, 0, 1, 0, 0],
112         [0, 1, 0, 1, 0, 0, 1, 0],
113         [0, 0, 1, 0, 1, 0, 0, 1],
114         [1, 0, 0, 1, 0, 1, 0, 0],
115         [0, 1, 0, 0, 1, 0, 1, 0]],
116     )
117
118     b = GF128([[1], [1], [0], [0], [0], [1], [1], [0]])
119     ib = GF128([[1], [0], [1], [0], [0], [0], [0], [0]])
120
121
122     def _sub_byte(byte: int) -> int:
123         """SubByte the integer according to  $s(x) = ax^{-1}+b$ """
124         number = GF128(byte)
125         rev = GF128(number) ** -1 if byte != 0 else 0
126         mat = GF128([[char] for char in
127             reversed(f'{bin(rev)[2:]:0>8}')])
128         _result = (sb_mat @ mat) + b
129         num = ''
130         for bit in reversed(_result):
131             num += str(bit[0])
132         return int(num, 2)
133
134     def _isub_byte(byte: int) -> int:
135         number = GF128(byte)
136         mat = GF128([[char] for char in
137             reversed(f'{bin(number)[2:]:0>8}')])
138         _result = (isb_mat @ mat) + ib
139         num = ''
140         for bit in reversed(_result):
141             num += str(bit[0])
142         rev = int(num, 2)
143         res = GF128(rev) ** -1 if byte != 0 else 0
144         return res
145
146     def _sub_bytes(input_stream: BitMat) -> BitMat:
147         _r = [
148             [0, 0, 0, 0],
149             [0, 0, 0, 0],
150             [0, 0, 0, 0],
151             [0, 0, 0, 0]
152         ]

```

```

153
154     for col in range(4):
155         for row in range(4):
156             sb = _sub_byte(input_stream[row][col])
157             _r[row][col] = sb
158     return _r
159
160
161 def _isub_bytes(input_stream: BitMat) -> BitMat:
162     _r = [
163         [0, 0, 0, 0],
164         [0, 0, 0, 0],
165         [0, 0, 0, 0],
166         [0, 0, 0, 0]
167     ]
168
169     for col in range(4):
170         for row in range(4):
171             sb = _isub_byte(input_stream[row][col])
172             _r[row][col] = sb
173     return _r
174
175
176 @type_and_len_check
177 def sub_bytes(stream: str) -> str:
178     """Sub byte the stream
179
180     Arguments:
181         stream: b1b2b3...b15
182
183     Return: s1s2s3...s15
184     """
185     _is = stream_to_matrix(stream)
186     return matrix_to_stream(_sub_bytes(_is))
187
188
189 @type_and_len_check
190 def isub_bytes(stream: str) -> str:
191     """Inverse sub byte the stream
192
193     Arguments:
194         stream: s1s2s3...s15
195
196     Return: b1b2b3...b15
197     """
198     _is = stream_to_matrix(stream)

```



```

199     return matrix_to_stream(_isub_bytes(_is))
200
201
202     ##### SR: Shift Rows ##### # noqa: E266
203     def _rotate_left(row: list, rotate: int):
204         for _ in range(rotate):
205             got = row.pop(0)
206             row.append(got)
207
208
209     def _shift_rows(input_stream: BitMat) -> BitMat:
210         _input_stream = input_stream.copy()
211         for idx, row in enumerate(_input_stream):
212             _rotate_left(row, idx)
213
214         return _input_stream
215
216
217     @type_and_len_check
218     def shift_row(stream: str) -> str:
219         """ShiftRow the stream
220
221         Arguments:
222             stream: b1b2b3...b15
223
224         Returns:
225             stream of:
226                 [[b0, b4, b8, b12],
227                  [b5, b9, b13, b1],
228                  [b10, b14, b2, b6],
229                  [b15, b3, b7, b11]]
230
231         """
232         strm = stream_to_matrix(stream)
233         return matrix_to_stream(_shift_rows(strm))
234
235     ##### MC: Mix Columns ##### # noqa: E266
236     mc_mat = GF128(
237         [[0x02, 0x03, 0x01, 0x01],
238          [0x01, 0x02, 0x03, 0x01],
239          [0x01, 0x01, 0x02, 0x03],
240          [0x03, 0x01, 0x01, 0x02]]
241     )
242
243
244     def _mix_column(input_stream: BitMat) -> BitMat:

```

```

245     _r = []
246     got = GF128(input_stream)
247     _result = got @ mc_mat
248     for row in _result:
249         _r.append([num for num in row])
250     return _r
251
252
253 @type_and_len_check
254 def mix_column(stream: str) -> str:
255     """MixColumn the stream
256
257     Arguments:
258         stream: b1b2b3...b15
259
260     Return:
261         apply a matrix multiplication and return the stream
262         c1c2c3...c15
263     """
264     _is = stream_to_matrix(stream)
265     return matrix_to_stream(_mix_column(_is))
266
267
268 ##### ARK: Add Round Key ##### # noqa: E266
269 @type_and_len_check
270 def add_round_key(stream: str, key: str) -> str:
271     return f'{bin(int(key, 2) ^ int(stream, 2))[2:]:0>128}'
272
273
274 ##### FR: Full Round ##### # noqa: E266
275 def _full_round(input_stream: BitMat, key: str) -> str:
276     _sb = _sub_bytes(input_stream)
277     _sr = _shift_rows(_sb)
278     _mc = _mix_column(_sr)
279     _ark = add_round_key(matrix_to_stream(_mc), key)
280     return _ark
281
282
283 @type_and_len_check
284 def full_round(stream: str, key: str) -> str:
285     return _full_round(stream_to_matrix(stream), key)

```

SR: Shift Row

P1: 000011001100101001010110011100011000010001001010101111010000110111000100100101101101
C1: 0000110011000010011111001101010011000000111010100100101001011011010011001010001001010010011

P2: 100011001100101001010010110011100011000010001001010101011101000011010111101001001100010010101101
C2: 100011001100001001111100110101001110000001110101001001010010110110100110110001010101010001001010010011

P3: 0000110011000010011111001101010011100000011101010010010100101110100010110010100010001001010010011
C3: 100011001100001001111100110101001110000001110101001001010010110110100110110100110010100001001010010011

SR: Shift Row

Changed bit No.	Bit change ratio
Cnt	1
ratio	0.78125%

SR: Shift Row (ب)

SB: Sub Byte

P1: 00001100110010100101011001110001100001000100101010111101000011101100010011000100100101101101
C1: 1111110011010101011110101001100000111001011001001000001110010010101001110011110110011000110011100

P2: 10001100110010100101001011001110001100001000100101010101110100001110101111010010011000100100101101
C2: 0110010001110100101111101011001100000111001001100000011100100101010011100111110110011000111100

P3: 111111001110100111101011001100000111001001100000011100100101010011100111110110011000011000111100
C3: 01100100011101001011111010110011000001110010011001001010010011100111101110011000110001110110011100

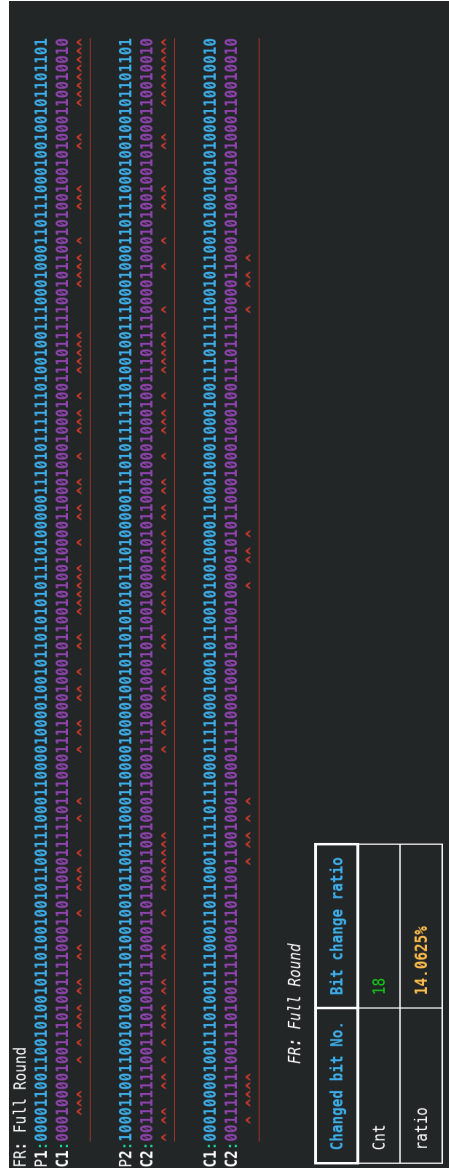
SB: Sub Bytes

Changed bit No.	Bit change ratio
Cnt	4
ratio	3.125%

SB: Sub Byte (ڀ)

ARK: Add Round Key (⌢)

MC: Mix Column (τ)



FR: Full Round (ۛ)

شکل ۱: جزئیات یک دور از AES