تکلیف دوم

مهدی حق‌وردی

۹ اردیبهشت ۱۴۰۳

فهرست مطالب

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-boxها 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 ← جستجوی 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

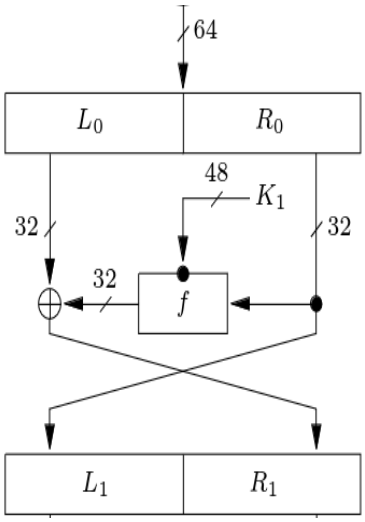
در این حالت هم (با استفاده از حمله‌ی تطابق در میانه) مرتبه بجای 2^{168} می‌شود: 2^{112} که البته در عمل قابل انجام نیست. در سال ۲۰۱۷ NIST منسوخ شدن 3DES را اعلام کرد.

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:

$$\mathrm{DES}_{\mathrm{K}}(\mathrm{M}) = \mathrm{C} \Rightarrow \mathrm{DES}_{\overline{\mathrm{K}}}(\overline{\mathrm{M}}) = \overline{\mathrm{C}} \tag{1}$$

(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:

$$\begin{cases} \mathrm{P} = \mathrm{L}_0.\mathrm{R}_0 \\ \mathrm{L}_1 = \mathrm{R}_0 \\ \mathrm{B} = \mathrm{f}(\mathrm{R}_0 \oplus \mathrm{K}_1) \\ \mathrm{R}_1 = \mathrm{L}_0 \oplus \mathrm{B} \end{cases} \Rightarrow \mathrm{C} = \mathrm{L}_1.\mathrm{R}_1 \tag{۲}$$

حال اگر P و K را not کنیم:

$$\begin{cases} \overline{P} = \overline{L_0.R_0} \\ L_1 = \overline{R_0} \\ B = f(\overline{R_0} \oplus \overline{K_1}) \\ R_1 = \overline{L_0} \oplus B \end{cases} \Rightarrow \overline{C} = \overline{L_1.R_1} \qquad (۳)$$

پس در نتیجه:

$$E_k(P) = C \iff E_{\overline{k}}(\overline{P}) = \overline{C} \qquad (۴)$$

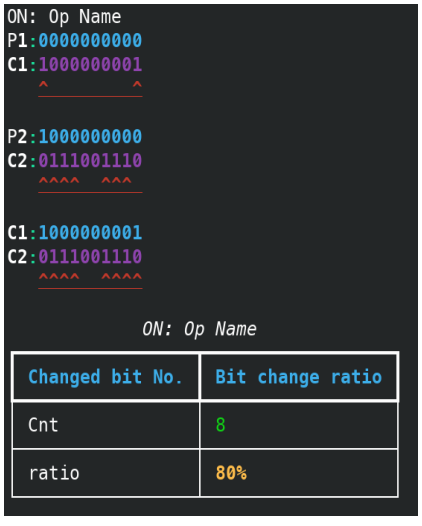
اگر تحت حمله‌ی chosen-plaintext attack به DES باشیم، با توجه به ویژگی مکمل بودن آن، ما می‌توانیم وقتی $E_k(P) = C$ را از DES عبور دادیم و C را گرفتیم، بار دوم **بدون محاسبه‌ی مجدد** $E_{\overline{k}}(\overline{P}) = \overline{C}$ را داشته باشیم. پس پیچیدگی شکستن آن نصف شده و به 2^{55} می‌رسد.

۲

با استفاده از یک کلید رمز واحد، هر یک ازتبدیلات زیر را بر متن آشکار که تنها در بیت اول با هم تفاوت دارند، اعمال کنید. تعداد بیت‌های تغییر یافته پس از هرتبدیل را پیدا کنید. هرتبدیل را بطور مستقل اعمال کنید. در مورد اثر بهمنی پس از هرتبدیل 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 جلویش نوشته شده است نوشته شده است.

۱.۲ توضیح تصاویر



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: • ECB • CTR

۴ پیوست‌ها

```python
"""This module implements 4 operations present in AES
- SB: Sub Bytes
- SR: Shift Rows
- MC: Mix Columns
- ARK: Add Round Key
- FR: Full Round (on of the 16 rounds)
"""

from functools import wraps
from inspect import getfullargspec
from itertools import islice
from typing import TypeAlias

import galois

__all__ = ['sub_bytes', 'isub_bytes', 'shift_row', 'mix_column',
    'add_round_key', 'full_round']

```

```python
BitMat: TypeAlias = list[list[int]]
GF128 = galois.GF(2, 8, irreducible_poly='x^8 + x^4 + x^3 + x + 1')


def batched(iterable, n):
    # batched('ABCDEFG', 3) -> ABC DEF G
    it = iter(iterable)
    while batch := tuple(islice(it, n)):
        yield batch


def stream_to_matrix(stream: str) -> BitMat:
    """Make the stream a matrix converted as an integers

    Arguments:
        stream: b0b1b2...b15

    Return:
        [[b0, b4, b8, b12],
         [b1, b5, b9, b13],
         [b2, b6, b10, b14],
         [b3, b7, b11, b15]]
    """
    ranges = []
    s = 0
    for i in range(0, 128 // 8):
        e = s + 8
        ranges.append((s, e))
        s = e

    mat = [[], [], [], []]
    for r in batched(ranges, 4):
        for idx, ra in enumerate(r):
            s, e = ra
            _stream = int(stream[s:e], 2)
            mat[idx].append(_stream)
    return mat


def matrix_to_stream(matrix: BitMat) -> str:
    """Make the matrix a stream

    Arguments:
        matrix: [[b0, b4, b8, b12],
                 [b1, b5, b9, b13],
```

```
                [b2, b6, b10, b14],
                [b3, b7, b11, b15]]

    Return:
        'b0b1b2b3...b15'
    """
    stream = ''
    for col in range(4):
        _stream = ''
        for row in matrix:
            _stream += f'{bin(row[col])[2:]:0>8}'
        stream += _stream
    return stream


def type_and_len_check(func):
    @wraps(func)
    def wrapper(*args):
        argnames = getfullargspec(func).args
        for arg, name in zip(args, argnames):
            if not isinstance(arg, str):
                raise TypeError(f'{name!r} should be `str`')

        for arg, name in zip(args, argnames):
            if not len(arg) == 128:
                raise ValueError(f'{name!r} should 128 bits')

        return func(*args)

    return wrapper


############ SB: Sub Bytes ############ # noqa: E266
sb_mat = GF128(
    [[1, 0, 0, 0, 1, 1, 1, 1],
     [1, 1, 0, 0, 0, 1, 1, 1],
     [1, 1, 1, 0, 0, 0, 1, 1],
     [1, 1, 1, 1, 0, 0, 0, 1],
     [1, 1, 1, 1, 1, 0, 0, 0],
     [0, 1, 1, 1, 1, 1, 0, 0],
     [0, 0, 1, 1, 1, 1, 1, 0],
     [0, 0, 0, 1, 1, 1, 1, 1]],
)

isb_mat = GF128(
    [[0, 0, 1, 0, 0, 1, 0, 1],
```

```
    [1, 0, 0, 1, 0, 0, 1, 0],
    [0, 1, 0, 0, 1, 0, 0, 1],
    [1, 0, 1, 0, 0, 1, 0, 0],
    [0, 1, 0, 1, 0, 0, 1, 0],
    [0, 0, 1, 0, 1, 0, 0, 1],
    [1, 0, 0, 1, 0, 1, 0, 0],
    [0, 1, 0, 0, 1, 0, 1, 0]],
)

b = GF128([[1], [1], [0], [0], [0], [1], [1], [0]])
ib = GF128([[1], [0], [1], [0], [0], [0], [0], [0]])


def _sub_byte(byte: int) -> int:
    """SubByte the integer according to s(x) = ax^(-1)+b"""
    number = GF128(byte)
    rev = GF128(number) ** -1 if byte != 0 else 0
    mat = GF128([[char] for char in
        reversed(f'{bin(rev)[2:]:0>8}')])
    _result = (sb_mat @ mat) + b
    num = ''
    for bit in reversed(_result):
        num += str(bit[0])
    return int(num, 2)


def _isub_byte(byte: int) -> int:
    number = GF128(byte)
    mat = GF128([[char] for char in
        reversed(f'{bin(number)[2:]:0>8}')])
    _result = (isb_mat @ mat) + ib
    num = ''
    for bit in reversed(_result):
        num += str(bit[0])
    rev = int(num, 2)
    res = GF128(rev) ** -1 if byte != 0 else 0
    return res


def _sub_bytes(input_stream: BitMat) -> BitMat:
    _r = [
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0],
        [0, 0, 0, 0]
    ]
```

```
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)
```

```
    return matrix_to_stream(_isub_bytes(_is))


############ SR: Shift Rows ############ # noqa: E266
def _rotate_left(row: list, rotate: int):
    for _ in range(rotate):
        got = row.pop(0)
        row.append(got)


def _shift_rows(input_stream: BitMat) -> BitMat:
    _input_stream = input_stream.copy()
    for idx, row in enumerate(_input_stream):
        _rotate_left(row, idx)

    return _input_stream


@type_and_len_check
def shift_row(stream: str) -> str:
    """ShiftRow the stream

    Arguments:
        stream: b1b2b3...b15

    Returns:
        stream of:
            [[b0, b4, b8, b12],
             [b5, b9, b13, b1],
             [b10, b14, b2, b6],
             [b15, b3, b7, b11]]
    """
    strm = stream_to_matrix(stream)
    return matrix_to_stream(_shift_rows(strm))


############ MC: Mix Columns ############ # noqa: E266
mc_mat = GF128(
    [[0x02, 0x03, 0x01, 0x01],
     [0x01, 0x02, 0x03, 0x01],
     [0x01, 0x01, 0x02, 0x03],
     [0x03, 0x01, 0x01, 0x02]]
)


def _mix_column(input_stream: BitMat) -> BitMat:
```

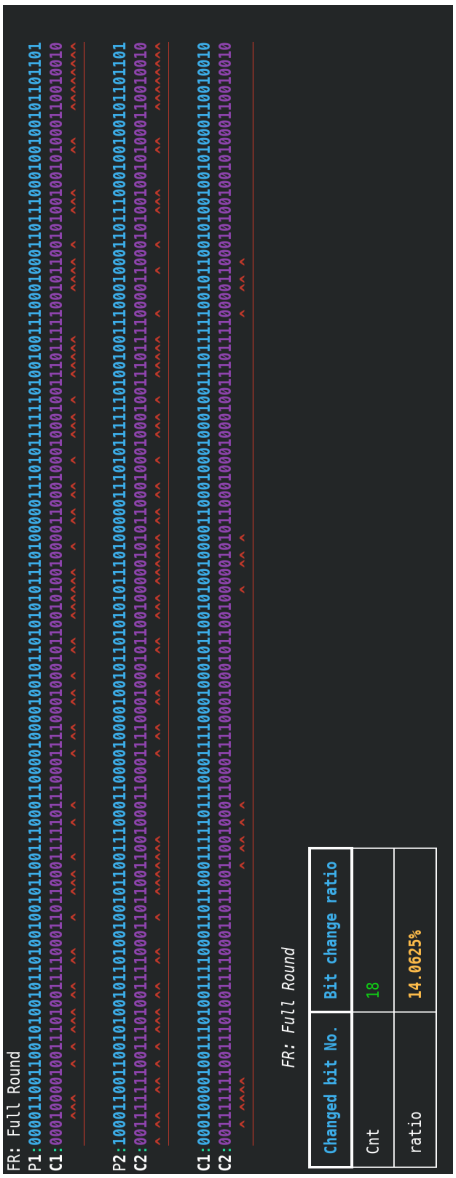
```
    _r = []
    got = GF128(input_stream)
    _result = got @ mc_mat
    for row in _result:
        _r.append([num for num in row])
    return _r


@type_and_len_check
def mix_column(stream: str) -> str:
    """MixColumn the stream

    Arguments:
        stream: b1b2b3...b15

    Return:
        apply a matrix multiplication and return the stream
        c1c2c3...c15
    """
    _is = stream_to_matrix(stream)
    return matrix_to_stream(_mix_column(_is))


############ ARK: Add Round Key ############ # noqa: E266
@type_and_len_check
def add_round_key(stream: str, key: str) -> str:
    return f'{bin(int(key, 2) ^ int(stream, 2))[2:]:0>128}'


############ FR: Full Round ############ # noqa: E266
def _full_round(input_stream: BitMat, key: str) -> str:
    _sb = _sub_bytes(input_stream)
    _sr = _shift_rows(_sb)
    _mc = _mix_column(_sr)
    _ark = add_round_key(matrix_to_stream(_mc), key)
    return _ark


@type_and_len_check
def full_round(stream: str, key: str) -> str:
    return _full_round(stream_to_matrix(stream), key)
```

SB: Sub Byte (أ)

SR: Shift Row (ب)

ARK: Add Round Key (د)

MC: Mix Column (ج)



FR: Full Round (۵)

شکل ۱: جزئیات یک دور از AES