بسم الله الرحمن الرحیم

تمرین اول درس مبانی امنیت شبکه های کامپیوتری

دکتر سعیدی

مهدی وجهی - ۸۱۰۱۰۱۵۵۸

سوال ۱

در ابتدا باینری متون را به دست می آوریم. این کار را با یک اسکریپت ساده پایتونی انجام می دهیم.

```
>>> to_ascii= lambda s: ''.join(format(ord(ch),'08b') for ch in s) #This
lambda generated with LLM
>>> alpha = to_ascii('alpha'); alpha
'01100001011011000111000001101000011000001'
>>> bravo = to_ascii('bravo'); bravo
'01100010011100100110000101110110011011111'
>>> delta = to_ascii('delta'); delta
'01100100011001010110110001110100011000001'
>>> gamma = to_ascii('gamma'); gamma
'01100111011000010110110101101101011000001'
```

با توجه به این که در این روش کلید با متن خام XOR میشه و متن رمز شده رو میده می توانیم با XOR متن رمز شده و متن خام به کلید برسیم. پس کافیست برای ۲ متن کلید را به دست بیاوریم و مقایسه کنیم.

```
>>> c1 = '11111001011110011100110000010111110000110'
>>> c2 = '11111010011001111101110100001001100010000'
>>>
>>> Op_XOR = lambda a,b: ''.join([str(int(a[i]) ^ int(b[i])) for i in
range(len(a))])
>>>
>>> k1 = Op_XOR(alpha, c1); k1
'10011000000101011011110001111111111100111'
>>> k2 = Op_XOR(bravo, c2); k2
'10011000000101011011110001111111111100111'
>>> k1 == k2
True
>>>
>>> k1 = Op_XOR(delta, c1); k1
'10011101000111001010000001100011111100111'
>>> k2 = Op_XOR(gamma, c2); k2
'10011101000001101011000001100100111101001'
>>> k1 == k2
False
```

پس حالت اول، حالت درستی است و دوم غلط است.

کلید برابر است با `'10011000000101011011110001111111111100111'`.

سوال ۲

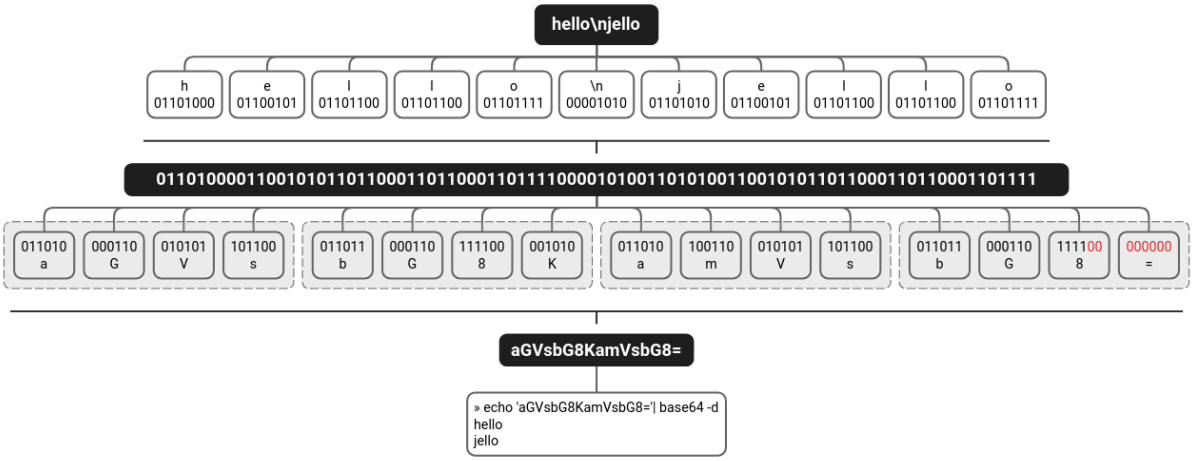
برای تبدیل به base64 گام های زیر را طی می کنیم:

1. تبدیل متن به باینری می کنیم.
2. به گروه های ۲۴ تایی و زیر گروه های ۶ تایی می شکنیم. (گروه آخر را در صورت لزوم pad می کنیم)
3. طبق جدول به کاراکتر های گفته شده تبدیل می کنیم. (کاراکتر padding '=' است)

جدول آن به شکل زیر است:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/

در نهایت با گام های زیر به جواب می رسیم.

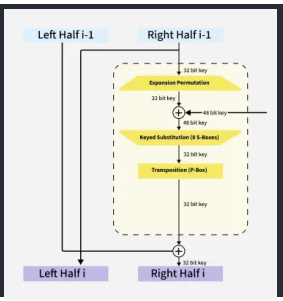


استفاده از مساوی برای پدینگ است. Base64 در دوره های ۲۴ بیتی کار می کند و بلاک آخر هر مقدار که کم باشد صفر اضافه می کنیم اما برای این که این صفر ها اشتباها به A تفسیر نشود آن ها را به = تبدیل می کنیم.

سوال ۳

ابتدا یک تابع برای بلاک تکرار شونده می نویسیم که عملکرد آن طبق تصویر مشخص است.

```
def DES_block(LE, RE, round_key):
    eRE = RE.permute( expansion_permutation )
    out_xor = eRE ^ round_key
    out_s = substitute( out_xor )
    out_p = RE.permute( p_box_permutation )
    new_RE = out_p ^ LE
    return (RE, new_RE)
```



در نهایت تابع DES را می نویسم که یک بلاک را کد می کند. شامل گام های زیر

1. پد کردن
2. پرمیوت کردن اولیه
3. اجرای بلاک
4. پرمیوت کردن نهایی

```
def DES(bitvec, round_keys):
    if bitvec._getsize() < 64:
        bitvec.pad_from_right(64 - bitvec._getsize())
    bitvec = bitvec.permute( init_permutation )
    [LE, RE] = bitvec.divide_into_two()
    for round_key in round_keys:
        (LE, RE) = DES_block( LE, RE, round_key )
    final_string = RE + LE
    final_string = final_string.permute( final_permutation )
    return final_string
```

برای کد کردن گام های زیر را طی می کنیم:

5. گرفتن کلید
6. تولید مجموعه کلید
7. خواندن بلاک بلاک
8. کد کردن بلاک با DES
9. نوشتن در فایل

```
def encrypt():
```

```
    key = get_encryption_key()
    round_keys = generate_round_keys( key )
    bv = BitVector( filename='filename.txt' )
    while (bv.more_to_read):
        bitvec = bv.read_bits_from_file( 64 )
        final_string = DES( bitvec, round_keys )
        with open('output.txt.enc', 'ab') as f:
            final_string.write_to_file( f )
```

نتیجه به صورت زیر است:

```
mvajhi@mahdi-laptop HW1/Q3 (main) » rm -rf output.*; echo "12345678\n12345678" | python hw1_starter.py
Enter a string of 8 characters for the key: Enter a string of 8 characters for the key: %
mvajhi@mahdi-laptop HW1/Q3 (main) » hexdump -C filename.txt
00000000  68 69 0a 6d 79 20 6e 61  6d 65 20 69 73 20 61 6c  |hi.my name is al|
00000010  69 20 76 61 6a 68 69 2e                           |i vajhi.|
00000018
mvajhi@mahdi-laptop HW1/Q3 (main) » hexdump -C output.txt.enc
00000000  b1 64 a9 5a 71 90 50 ce  09 ea 86 1f f5 ad 37 28  |.d.Zq.P.......7(|
00000010  fc 7d e1 41 f8 ee 7c 40                           |.}.A..|@|
00000018
mvajhi@mahdi-laptop HW1/Q3 (main) » hexdump -C output.txt
00000000  68 69 0a 6d 79 20 6e 61  6d 65 20 69 73 20 61 6c  |hi.my name is al|
00000010  69 20 76 61 6a 68 69 2e                           |i vajhi.|
00000018
```

سوال ۴

با عبارت زیر می توانیم s box تصادفی تولید کنیم.

```
generate_random_s_boxes = lambda: {i: [random.sample(range(16), 16) for _ in range(4)] for i in range(8)}
```

یک تابع می نویسم برای سنجش اثر بهمنی که موارد زیر را انجام می دهد.

1. تولید کلید و متن تصادفی
2. رمز کردن با دو روش
3. تغییر یک بیت
4. رمز کردن مجدد با دو روش
5. ثبت تفاوت در هر کدام
6. تست t و نمایش نتایج

```
# Write with LLM
# https://g.co/gemini/share/35a049a6914b
def calculate_avalanche_effect(num_trials=100):
    flips_default = []
    flips_random = []

    generate_random_s_boxes = lambda: {i: [random.sample(range(16), 16) for _
```

```
in range(4)] for i in range(8)}

   print(f"Starting avalanche effect calculation for {num_trials}
trials...")
   for i in tqdm(range(num_trials), desc="Processing..."):
       # 1. Create identical random key and plaintext for both methods
       key_bv = BitVector(intVal=random.getrandbits(56), size=56)
       plaintext_bv = BitVector(intVal=random.getrandbits(64), size=64)

       # Generate round keys
       round_keys = generate_round_keys(key_bv)

       # Generate s-box
       rand_s_boxes = generate_random_s_boxes()

       # 2. Encrypt using both methods
       illustrate_des_substitution.s_boxes = ORIGINAL_S_BOX
       cipher_default1 = DES(plaintext_bv, round_keys)
       illustrate_des_substitution.s_boxes = rand_s_boxes
       cipher_random1 = DES(plaintext_bv, round_keys)

       # 3. Flip one bit in the original plaintext
       flip_pos = random.randint(0, 63)
       plaintext_flipped = plaintext_bv.deep_copy()
       plaintext_flipped[flip_pos] = 1 - plaintext_flipped[flip_pos]

       # 4. Encrypt the flipped plaintext with both methods
       illustrate_des_substitution.s_boxes = ORIGINAL_S_BOX
       cipher_default2 = DES(plaintext_flipped, round_keys)
       illustrate_des_substitution.s_boxes = rand_s_boxes
       cipher_random2 = DES(plaintext_flipped, round_keys)

       # 5. Calculate the Hamming distance for both
       flips_default += [(cipher_default1 ^ cipher_default2).count_bits()]
       flips_random += [(cipher_random1 ^ cipher_random2).count_bits()]

   # Calculate the average
   avg_flips_default = mean(flips_default)
   avg_flips_random = mean(flips_random)

   print("\n--- Final Results ---")
   print(f"Default S-box: Average bits flipped = {avg_flips_default:.2f} /
64")
   print(f"Random S-box:  Average bits flipped = {avg_flips_random:.2f} /
64")

   # Using LLM https://g.co/gemini/share/3c4181eceaef
   t_statistic, p_value = ttest_ind(flips_default, flips_random)
```

```
print(f"T-statistic: {t_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

alpha = 0.05
if p_value < alpha:
    print("✅ Result is different")
else:
    print("❌ Result is not different")
```

در نهایت نتیجه به صورت زیر است و طبق تست آماری، موفق نشدیم که تفاوت و تاثیر s-box را نشان دهیم.

```
Starting avalanche effect calculation for 10000 trials...
Processing...:
100%|
10000/10000 [03:09<00:00, 52.89it/s]

--- Final Results ---
Default S-box: Average bits flipped = 32.03 / 64
Random S-box:  Average bits flipped = 32.02 / 64
T-statistic: 0.2932
P-value: 0.7694
❌ Result is not differnt
```

سوال ۵

ECB

https://g.co/gemini/share/97bacaea1536

مفروضات

ECB	$C_j = \mathrm{E}(K, P_j) \quad j = 1, \ldots, N$	$P_j = \mathrm{D}(K, C_j) \quad j = 1, \ldots, N$

1.
2. دریافت C'_i به جای C_i
3. سالم دریافت شدن باقی موارد (K و تمامی C ها به غیر i)

حکم

- دقیقا یک بسته را نمی توانیم بازیابی کنیم.

برهان خلف

برهان خلف می زنیم. فرض می کنیم با خراب دریافت کردن C_i (مقدار دریافتی را C'_i فرض می کنیم) بیش از یک بسته خراب می شود یا کمتر از یک بسته.

حالت کمتر از یک بسته: تعداد بسته ها نمی تواند منفی باشد زیرا:

A. شمارش "بلوک‌های متن" با بررسی وجود آن‌ها آغاز می‌شود.

B. اگر هیچ بلوکی یافت نشود، تعداد آن‌ها صفر است.

C. به محض یافتن اولین بلوک، شمارش به سمت اعداد مثبت حرکت می‌کند (۱, ۲, ۳, ...).

بنابراین، تلاش برای نسبت دادن یک عدد منفی به تعداد بلوک‌های متن، یک تناقض فلسفی ایجاد می‌کند. این کار به منزله‌ی ادعای وجودِ "کمتر از هیچ" برای یک موجودیت است که ذاتاً یا وجود دارد (و قابل شمارش است (طبق فرض C))یا وجود ندارد(فرض A). همانطور که نمی‌توان اتاقی داشت که "منفی یک صندلی" در آن باشد، نمی‌توان تعداد بلوک متنی خراب داشت که دارای "منفی یک بلوک" باشد.

همچنین طبق فرمول تنها راه به دست آوردن متن i استفاده از $P_i = D(K, C_i)$ است (طبق فرض ۱). پس به دست آوردن P_i بدون C_i با این الگوریتم یک تناقض است.

حالت بیشتر از یک بسته: حداقل ۲ بسته خراب داریم. یکی از بسته های خراب که نامساوی i است را در نظر بگیرد مثلا j. همچنین طبق فرض ۳ ما C_j و K و تابع D را داریم پس می توانیم با استفاده از فرمول $P_j = D(K, C_j)$ (فرض ۱) مقدار P_j را حساب کنیم که با فرض عدم بازیابی آن در تناقض است.

هر دو حالت برهان خلف به تناقض رسید پس حکم اثبات می شود.

OFB

مفروضات

OFB	$I_1 = Nonce$ $I_j = O_{j-1} \quad j = 2, \ldots, N$ $O_j = \mathrm{E}(K, I_j) \quad j = 1, \ldots, N$ $C_j = P_j \oplus O_j \quad j = 1, \ldots, N-1$ $C_N^* = P_N^* \oplus \mathrm{MSB}_u(O_N)$	$I_1 = Nonce$ $I_j = O_{j-1} \quad j = 2, \ldots, N$ $O_j = \mathrm{E}(K, I_j) \quad j = 1, \ldots, N$ $P_j = C_j \oplus O_j \quad j = 1, \ldots, N-1$ $P_N^* = C_N^* \oplus \mathrm{MSB}_u(O_N)$

1.

2. دریافت C'_i به جای C_i

3. سالم دریافت شدن باقی موارد (K و Nonce و تمامی C ها به غیر i)

حکم

- دقیقا یک بسته را نمی توانیم بازیابی کنیم.

برهان خلف

برهان خلف می زنیم. فرض می کنیم با خراب دریافت کردن C_i (مقدار دریافتی را C'_i فرض می کنیم) بیش از یک بسته خراب می شود یا کمتر از یک بسته. **حالت کمتر از یک بسته** دقیقا مشابه استدلال موجود در ECB است و از شرح مجدد آن صرف نظر می کنیم.

حالت بیشتر از یک بسته: این هم مشابه ECB است. تنها توجه کنید که O ها مستقل از تمامی C ها هستند و تمامی P ها (غیر i) می توانند مستقل از C_i حساب شوند.

پس مجدد **هر دو حالت برهان خلف به تناقض رسید پس حکم اثبات می شود.**

OFB

مفروضات

CBC	$C_1 = \mathrm{E}(K, [P_1 \oplus \mathrm{IV}])$ $C_j = \mathrm{E}(K, [P_j \oplus C_{j-1}]) j = 2, \ldots, N$	$P_1 = \mathrm{D}(K, C_1) \oplus \mathrm{IV}$ $P_j = \mathrm{D}(K, C_j) \oplus C_{j-1}\, j = 2, \ldots, N$

1.
2. دریافت C'_i به جای C_i
3. سالم دریافت شدن باقی موارد (K و IV و تمامی C ها به غیر i)

حکم

- دقیقا دو (یا یک) بسته را نمی توانیم بازیابی کنیم.

برهان خلف

برهان خلف می زنیم. فرض می کنیم با خراب دریافت کردن C_i (مقدار دریافتی را C'_i فرض می کنیم) بیش از دو بسته خراب می شود یا کمتر از دو بسته.

حالت کمتر از دو بسته: حالت منفی مانند موارد قبلی مردود است. طبق فرض ۱ دو بلوک زیر مستقیما وابسته به C_i هستند:

$$P_i = D(K, C_i) \oplus C_{i-1}\ if\ i = 1: P_i = D(K, C_i) \oplus IV$$

$$P_{i+1} = D(K, C_{i+1}) \oplus C_i$$

پس این دو از با این الگوریتم به هیچ عنوان قابل بازیابی نیستند و این با فرض در تناقض است.

اگر N=i دیگر i+1 وجود ندارد پس می تواند کمتر از دو بسته خراب باشد که این یعنی صورت سوال اشتباه است و می توان در CBC کمتر از ۲ بسته از دست داد. صورت سوال را به ۲ یا ۱ تغییر می دهیم تا درست باشد.

حالت بیشتر از دو بسته: حداقل ۳ بسته خراب داریم. یکی از بسته های خراب که نامساوی i و i+1 است را در نظر بگیرد مثلا j. همچنین طبق فرض ۳ ما C_j و K و IV و تابع D را داریم. همچنین چون j نامساوی i+1

C_{j-1} هم داریم. پس می توانیم با استفاده از فرمول(فرض ۱) مقدار P_j را حساب کنیم که با فرض عدم بازیابی آن در تناقض است.

هر دو حالت برهان خلف به تناقض رسید (با تغییر صورت سوال) پس حکم اثبات می شود.

OFB

https://chatgpt.com/share/68e6891b-ed1c-8001-a798-bdaf87c893fb

مفروضات

CFB	$I_1 = IV$ $I_j = \mathrm{LSB}_{b-s}(I_{j-1}) \| C_{j-1} \quad j = 2, \ldots, N$ $O_j = \mathrm{E}(K, I_j) \quad j = 1, \ldots, N$ $C_j = P_j \oplus \mathrm{MSB}_s(O_j) \quad j = 1, \ldots, N$	$I_1 = IV$ $I_j = \mathrm{LSB}_{b-s}(I_{j-1}) \| C_{j-1} \quad j = 2, \ldots, N$ $O_j = \mathrm{E}(K, I_j) \quad j = 1, \ldots, N$ $P_j = C_j \oplus \mathrm{MSB}_s(O_j) \quad j = 1, \ldots, N$

1.
2. دریافت C'_i به جای C_i
3. سالم دریافت شدن باقی موارد (K و IV و تمامی C ها به غیر i)

حکم

- دقیقا دو (یا یک) بسته را نمی توانیم بازیابی کنیم.

برهان خلف

برهان خلف می زنیم. فرض می کنیم با خراب دریافت کردن C_i (مقدار دریافتی را C'_i فرض می کنیم) بیش از دو بسته خراب می شود یا کمتر از دو بسته.

حالت کمتر از دو بسته: حالت منفی مانند موارد قبلی مردود است. طبق فرض ۱ دو بلوک زیر مستقیما وابسته به C_i هستند:

$$P_i = C_i \oplus MSB_s(O_i)$$
$$P_{i+1} = C_{i+1} \oplus MSB_s(E(K, LSB_{b-s}(I_{i-1}) || C_i))$$

پس این دو از با این الگوریتم به هیچ عنوان قابل بازیابی نیستند و این با فرض در تناقض است. (همان ایراد بخش قبل)

حالت بیشتر از دو بسته: حداقل ۳ بسته خراب داریم. یکی از بسته های خراب که نامساوی i و i+1 است را در نظر بگیرد مثلا j. همچنین طبق فرض ۳ ما C_j و C_{j-1} و K و IV را داریم(C_j و C_{j-1} را مشابه استدلال بخش قبل داریم). پس می توانیم با استفاده از فرمول(فرض ۱) مقدار P_j را حساب کنیم که با فرض عدم بازیابی آن در تناقض است.

هر دو حالت برهان خلف به تناقض رسید (با تغییر صورت سوال) پس حکم اثبات می شود.

منابع

Base64 encoding: What sysadmins need to know

Base64 Encoding – How it works?