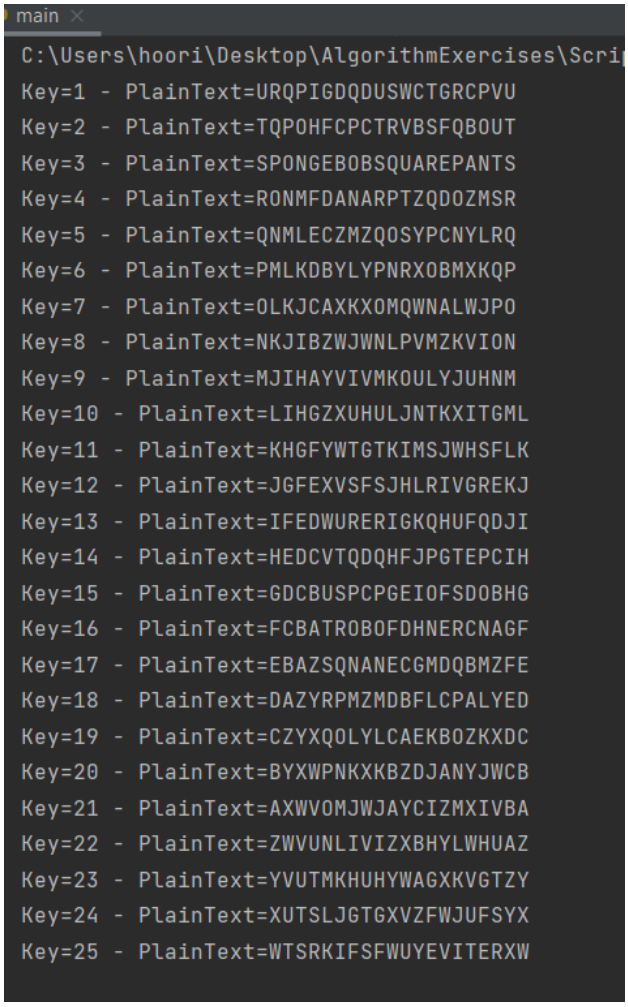
حوری دهش 9821413

سوال 1:

نکته: هر دو راه حل در یک فایل پایتون نوشته شده است.

برای حل این سوال من دوتا راه حل به ذهنم رسید ولی اطمینان ندارم که درست هستند یا خیر.

راه حل اول:

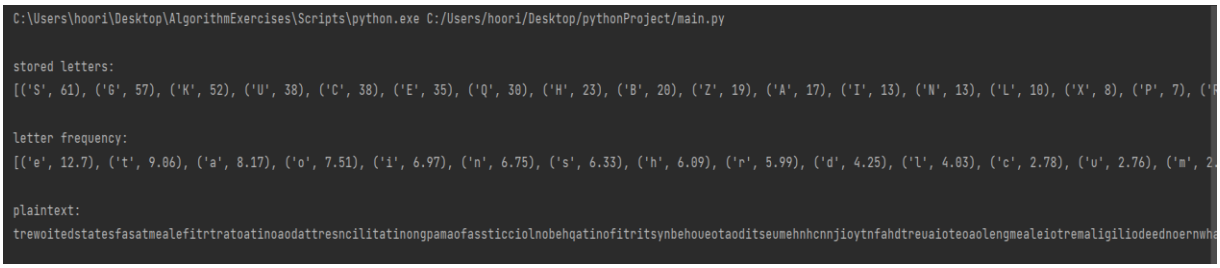


خلاصه ای از توضیح کد:

توی این روش ciphertext و سایز کلید که 26 در نظر گرفتیم را می دهیم سپس متن ciphertext را با کلیدی که در ان لحظه داریم یکی به سمت چپ شیفت میدهیم مثلا اگر کلید را یک در نظر بگیریم همه حروف را یکی به سمت چپ شیفت می دهیم تا plaintext حاصل شود و کار این شیفت به عقب را تابع decrypt انجام می دهد.

راه حل دوم:

توی این روش plaintext ما دقیق نمی باشد و کلید هم نمی دانم



خلاصه ای از توضیح کد:

ابتدا متن ciphertext را به حروف های بزرگ تبدیل میکنیم سپس یک دیکشنری خالی به اسم stored_letters ایجاد میکنیم و در نهایت متن ciphertext را می شماریم که ببینیم چندین حرف داخل متن است سپس با شمارش حروف دیکشنری stored_letters را پر می کنیم.

دیکشنری stored_letters و LETTER_FREQUENCY را به صورت صعودی مرتب می کنیم حالا جای حروف دو دیکشنری را عوض میکنیم به این صورت که در دیکشنری stored_letters حرف s بیشترین تکرار را دارد و در دیکشنری LETTER_FREQUENCY حرف e بیشترین فرکانس را دارد پس باید جای حرف s و e را با هم عوض کنیم تا plaintext به دست اید و برای بقیه حروف هم همین کارو میکنیم.

سوال 2:

Ciphertext:

I	T	N	G	T	W	H
A	E	H	A	N	L	G
E	I	S	O	T	M	H
E	P	H	E	F	L	T
S	O	N	G	W	I	R

برای به دست اوردن plaintext اگر جای:

ستون 2 و 7 - سطر 2 و 5 - ستون 1 و2 - ستون 3 و 5 - ستون 2 و 4 - سطر 1 و 3 - سطر 2 و 3

را عوض کنیم به دست می اید.

plaintext:

S	O	M	E	T	H	I
N	G	I	S	W	R	O
N	G	W	I	T	H	T
H	E	L	E	F	T	P
H	A	L	A	N	G	E

Something is wrong with the left phalange

کلید ما:

یک ماتریس 7 * 5

جایگشت: (3,1,5,4,2) and (7,4,5,1,3,6,2)

سوال 3:

الف)

Ciphertext	241	355	645	668	704	566	530	401	490	670
Additive	118	156	344	217	415	265	407	100	201	369
Plaintext	123	199	301	451	289	301	123	301	289	301
Plaintext	They	don't	know	that	we	know	they	know	we	know

برای plaintext تفاضل انجام میدیم به عنوان مثال: 118-241

ب)

برای به دست اوردن ciphertext بدون additive ینی این که به جای حروف plaintext مرحله قبل فقط عددهایی که از کتاب داریم رو قرار بدیم:

Ciphertext: 123 199 301 451 289 301 123 301 289 301

سوال 4:

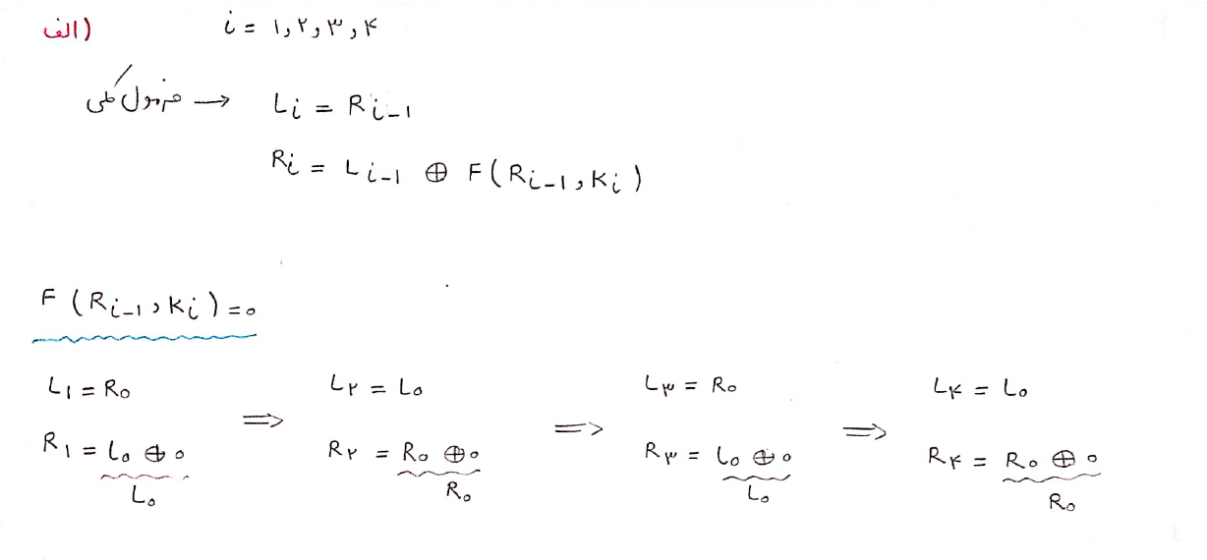
در رمزنگاری RC4 برای هر کلید 2048 بیتی یک خانواده از کلیدهای مرتبط وجود دارد که در یکی از موقعیت‌های بایت متفاوت است. پس جریان‌های کلیدهای تولید شده توسط RC4 برای یک کلید و کلیدهای مرتبط با آن، در صد بایت اولیه قبل از واگرایی، به طور قابل ملاحظه‌ای مشابه هستند و مابین کلیدها یک تابع دو طرفه دلخواه می‌باشد. مهاجم رابطه بین چندین کلید را شناسایی می‌کند و بدین صورت به توابع رمزگذاری کلیدها دسترسی پیدا می‌کند و پس از آن می‌تواند خود کلیدها را پیدا کند و داده رمزنگاری شده را رمزگشایی کند.

سوال 5:

جواب ها را براساس 4 دور می دهیم:

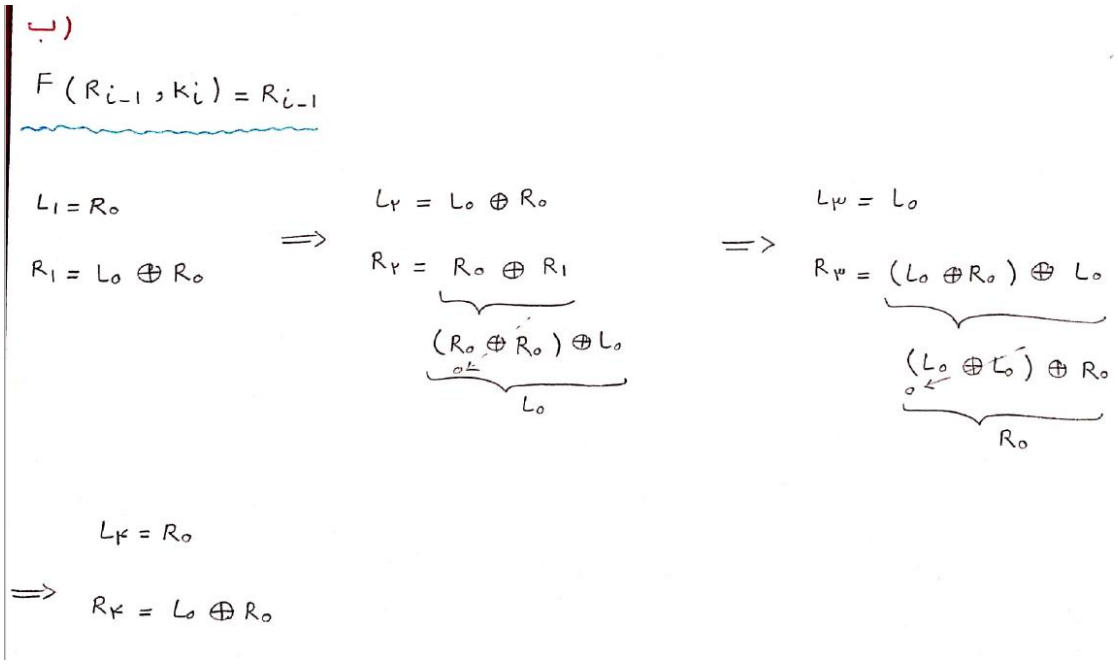
الف) $C = (L_0 , R_0)$

راه حل:



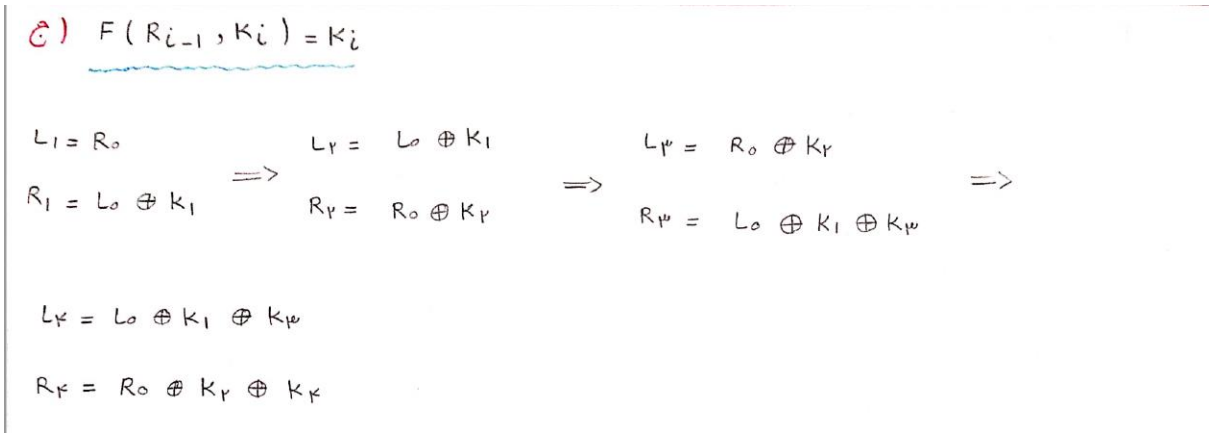
ب) $C = (R_0 , R_0 \, xor \, L_0)$

راه حل:



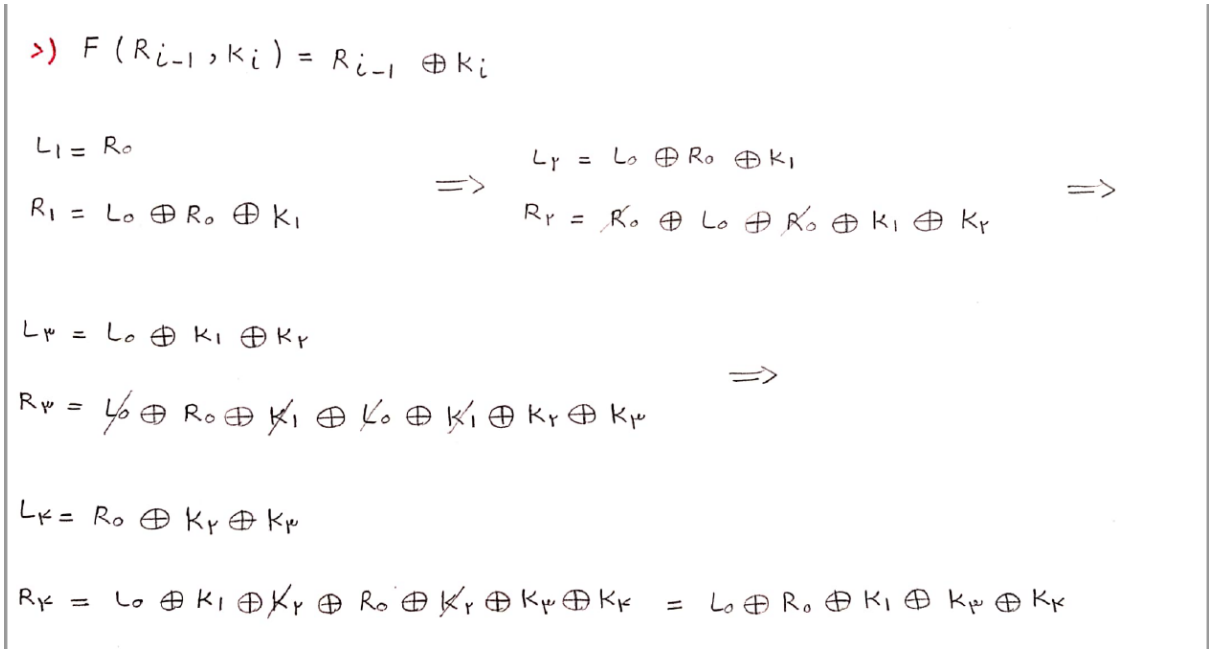
ج) $C = (L_0 \; xor \; K_1 \; xor \; K_3 \; , R_0 \; xor \; K_2 \; xor \; K_4)$

راه حل:



د) $C = (R_0 \; xor \; K_2 \; xor \; K_3 \; , R_0 \; \; xor \; L_0 \; xor \; K_1 \; xor \; K_3 \; xor \; K_4)$

راه حل:



سوال 6:

Encrypt: $C_0 = IV \;\; , \;\; C_i = \; P_i \; xor \; E(\; C_{i-1} \; , K)$

Decrypt: $P_i = \; C_i \; xor \; E(\; C_{i-1} \; , K)$

در حالت کلی برای مقایسه:

برای CBC وCFB ، استفاده کردن مجدد از یک iv برخی از اطلاعات را در مورد اولین قطعه plaintext و درباره هرگونه پیشوند مشترکی که بین دو پیغام به اشتراک گذاشته می‌شود، نشت می‌دهد.

برای CTR استفاده مجدد از یک iv امنیت را به‌طور کامل از بین می‌برد.

مقایسه CFB با CBC :

نکته: در اینجا ciphertext مرحله قبل همان iv ما می باشد

برای حالت encryption :

در مد CBC: $C_0 = E(IV \text{ xor } P_0, K)$ در حالی که برای مد CFB: $C_0 = IV$ است و برای حالت کلی CFB غیر از C_0 می تونیم به این موضوع اشاره کنیم که در این مد ciphertext مرحله قبلی با کلید رمز می شود و در نهایت با plaintext همون مرحله xor میشود ولی در مد CBC برای بقیه حالت ها غیر از C_0 اول ciphertext مرحله قبلی با plaintext همون مرحله xor میشود و در نهایت رمز با کلید صورت میگیرد.

برای حالت decryption:

در مد CFB: ciphertext مرحله قبلی با کلید رمز میشود و بعد در نهایت با ciphertext همون مرحله xor میشود تا plaintext به دست اید در حالی در مد CBC: ابتدا ciphertext همون مرحله با کلید رمزگشایی میشود و بعد با ciphertext مرحله قبل که همان iv ما میشود xor میشود تا plaintext به دست اید.

مقایسه CFB با CTR :

برای حالت encryption :

در مد CTR: ابتدا iv با کلید رمز میشود و بعد با plaintext همان مرحله xor میشود و هر مرحله که جلوتر می رویم به iv یک مقدار اضافه میکنیم و نکته ای که وجود دارد این است که iv ما اینجا ciphertext مرحله های قبلی نیست ولی در مد CFB: C_0 ما ابتدا برابر با iv است و برای بقیه c ها غیر از C_0 می تونیم به این موضوع اشاره کنیم که در این مد ciphertext مرحله قبلی با کلید رمز می شود و در نهایت با plaintext همون مرحله xor میشود توی این مد iv ما ciphertext مرحله قبلی است.

برای حالت decryption:

در مد CTR: ابتدا iv با کلید رمز میشود و بعد با ciphertext همان مرحله xor میشود تا به plaintext برسد و در هر مرحله iv یکی زیاد میشود و مانند encryption در این مد iv ما ciphertext های مرحله قبل نیست ولی در مد CFB: ciphertext مرحله قبلی با کلید رمز میشود و بعد در نهایت با ciphertext همون مرحله xor میشود تا plaintext به دست اید و اینجا هم مانند encryption توی این مد iv ما ciphertext مرحله قبلی است.

سوال 7:

الف) در این صورت مد CBC در مقابل حمله CPA مقاوم می‌شود یعنی دیگر مهاجم نمی‌تواند با ساختن یک دیکشنری از پیام‌های احتمالی و متن‌های رمز مربوط به آن‌ها، رمزنگاری را از بین ببرد. اما در مقابل حملات CCA امن نیست. CCA یعنی این که trudy فرصت دارد که یک یا چندین متن‌های رمز شده شناخته شده را به سیستم وارد کند و در نتیجه plaintext را به دست اورد. توسط این قطعه از اطلاعات، Trudyمی‌تواند برای بازیابی کلید امنیتی مخفی که برای رمزگشایی استفاده می‌شود، تلاش کند.

ب) این مد هم در مقابل حمله CPA مقاومت دارد.

ج) حمله CTR بدتر است چون از یک keystream واحد هر بار استفاده میکند.

سوال 8:

کد زدن در محیط Jupyter انجام گرفته

در مرحله اول عکس را با دستور:

image = cv2.imread(r'C:\Users\hoori\Desktop\HW1\HW1_ dahesh_#8.jpg')

فراخوانی میکنیم و برای اینکه عکس را خاکستری رنگ کرده از دستور:

gray= cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

استفاده میکنیم

حالا که عکس را خاکستری رنگ کردیم ان را رمز میکنیم

با استفاده از mode تعیین میکنیم که می خواهیم رمزگذاری CBC را انجام دهیم یا ECB را.

مراحل را برای رمزگذاری ECB تعریف میکنیم و رمزگذاری CBC هم دقیقا مانند ECB است با این تفاوت که داخل mode ما CBC قرار میگیرد

توضیح:

mode = AES.MODE_ECB رمزگذاری ECB میخواهیم انجام دهیم.

keySize = 32

ivSize = AES.block_size if mode == AES.MODE_CBC else 0

در مرحله بعد برای کلید و iv سایزی در نظر میگیریم و در صورتی از ivSize استفاده میکنیم که مد ما CBC باشد ینی اگر مد ما ECB بود برایش iv در نظر نمی گیریم.

imageOrig = gray

imageOrig = imageOrig[:, :, newaxis]

rowOrig, columnOrig, depthOrig = imageOrig.shape

در این مرحله ابتدا عکس خاکستری رنگ را در متغیر imageOrig می ریزیم و از انجایی که ارایه عکس های سیاه و سفید دو بعدی است و ما برای رمزنگاری به ارایه سه بعدی احتیاج داریم با کتابخانه newaxis عکسمان را سه بعدی می کنیم و سپس سایز هر بعد را در متغیرهای ,rowOrig columnOrig, depthOrig می ریزیم.

```
imageOrigBytes = imageOrig.tobytes()
```

در این مرحله داده های عکس را به بایت تبدیل میکند

```
key = get_random_bytes(keySize)

iv = get_random_bytes(ivSize)

cipher = AES.new(key, AES.MODE_CBC, iv) if mode == AES.MODE_CBC else
AES.new(key, AES.MODE_ECB)

imageOrigBytesPadded = pad(imageOrigBytes, AES.block_size)

ciphertext = cipher.encrypt(imageOrigBytesPadded)
```

در این مرحله کلید و iv سایز را به صورت تصادفی انتخاب میکنیم سپس با AES براساس مد، کلید و iv سایزcipher را می سازیم سپس با pad داده imageOrigBytes را تا سقف block_size مورد نظر پر میکنیم و در پایان داده به دست امده در imageOrigBytesPadded را رمزگذاری می نماییم تا ciphertext حاصل شود.

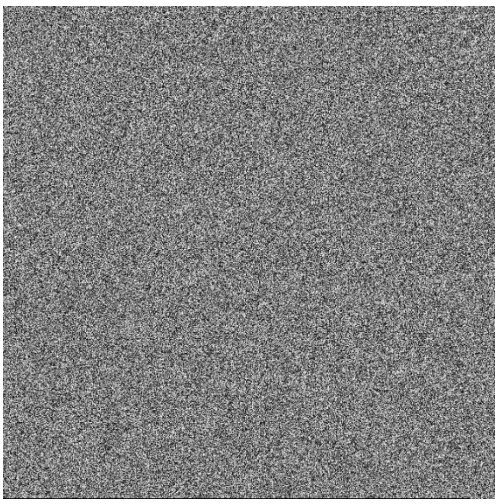
```
paddedSize = len(imageOrigBytesPadded) - len(imageOrigBytes)

void = columnOrig * depthOrig - ivSize - paddedSize

ivCiphertextVoid = iv + ciphertext + bytes(void)

imageEncrypted = np.frombuffer(ivCiphertextVoid, dtype =
imageOrig.dtype).reshape(rowOrig + 1, columnOrig, depthOrig)
```

در این مرحله ciphertext که به صورت بایت بود را تبدیل به داده عکسی میکنیم.

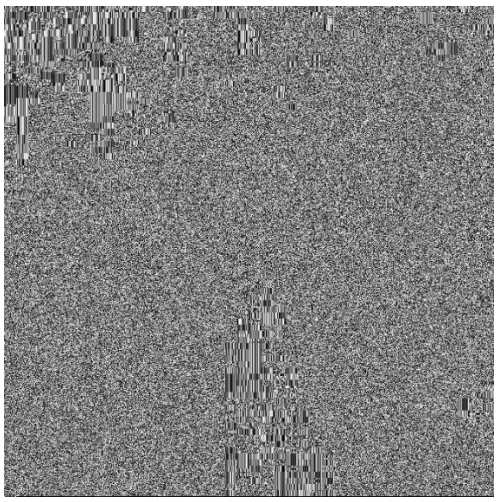
```
cv2.imshow("ECB image", imageEncrypted)

cv2.imwrite(r'C:\Users\hoori\Desktop\HW1\HW1_ dahesh_#8_ECB.jpg',
imageEncrypted)

cv2.waitKey()

cv2.destroyAllWindows()
```

و در اخر عکس به دست امده از رمزنگاری را در یک پوشه ریخته و تصویر ان را نمایش می دهیم.

CBC:



ECB:



سوال اضافه:

این حمله از نوع حملات متن رمز شده انتخابی یا Choosen ciphrtext attack می باشد.

جفت متن های رمز شده به نحوی انتخاب میشوند که xor متن های متناظر اصلیشان مقدار خاصی باشد. این حمله ابتدا بر روی DES با دور های کمتر از 16 انجام شد و یک سال بعد با 16 دور انجام شد.

در حمله تفاضلي به سیستم DES و یا سیستم هاي مشابه، جفت متن های رمز شده را به گونه ای انتخاب می کنند که xor متن های اصلی متناظرشان مقادیر خاصی داشته باشد. با بررسی تاثیر این مقادیر بر روی xor متن های رمز شده متناظر می توان در مورد احتمال وقوع کلید های مختلف بحث نمود. برای این کار دو متن ورودی با هم xor شده و متن های رمز شده متناظرشان نیز با هم xor می شوند. سپس با توجه به xor ورودی و خروجی و مقادیر این تفاضل ها در دورهای میانی می توان حدس هایی در مورد کلید زد.

پس اساس کار حمله تفاضلی بررسی xor جفت های ورودی و خروجی است. مراحل گسترش، جابجایی، xor با کلید دوره و xor نیمه چپ متن با خروجی تابع F تاثیری بر روی xor جفت ها ندارد. ولی وارد شدن یک xor خاص به توابع S به منزله xor خروجی خاص نمی تواند

باشد. ولی می توان برای هر xor ورودی ممکن، احتمال xor های خروجی مختلف را به دست اورد.

جدول زیر توزیع احتمال xor های ورودی و خروجی را برای تابع S1 نشان میدهد:

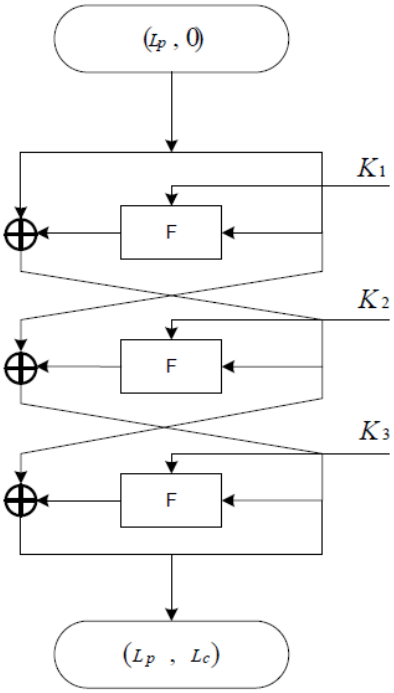
	خروجی XOR															
XOR ورودی	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x00	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x01	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
0x02	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
0x03	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
0x04	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
0x05	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
0x06	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
0x07	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
0x08	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
0x09	10	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
0x0A	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
0x0B	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
0x0C	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
0x0D	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
0x0E	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
0x0F	2	0	2	4	4	6	4	2	4	8	2	2	2	6	8	8
0x10	0	0	0	0	0	0	2	14	0	6	6	12	4	6	8	6
0x11	0	8	2	4	6	4	8	6	4	0	6	6	0	4	0	0
0x12	0	8	4	2	6	6	4	6	6	4	2	6	6	0	4	0
0x13	4	4	4	6	2	0	4	6	2	0	6	8	4	6	4	6
0x14	0	8	8	0	10	0	4	2	8	2	2	4	4	8	4	0
0x15	2	4	6	4	2	2	4	10	6	2	0	10	0	4	6	4
0x16	0	8	10	8	0	2	2	6	10	2	0	2	0	6	2	6
0x17	4	4	6	0	10	6	0	2	4	4	4	6	6	6	2	0
0x18	0	6	6	0	8	4	2	2	2	4	6	8	6	6	2	2
0x19	4	6	2	4	0	8	4	6	10	4	0	4	2	8	4	0
0x1A	0	6	4	0	4	6	6	2	6	4	2	0	4	4	6	8
0x1B	2	4	2	4	10	6	6	6	6	4	2	4	2	2	4	2
0x1C	0	10	10	6	6	0	0	2	6	4	0	0	2	4	4	0
0x1D	0	2	4	0	8	0	0	2	10	4	2	6	6	6	14	0
0x1E	10	2	6	0	14	2	10	4	4	4	10	8	2	2	6	2
0x1F	0	4	10	6	2	2	2	4	2	6	6	0	0	4	6	4
0x20	12	0	0	10	0	12	6	0	6	2	2	4	4	2	0	12
0x21	6	4	2	4	4	8	2	0	14	4	2	0	4	0	2	8
0x22	0	4	6	2	2	8	4	0	2	6	2	0	4	0	4	10
0x23	10	4	4	8	0	2	2	0	0	6	6	10	2	4	0	10
0x24	12	0	0	2	2	2	2	6	4	6	0	0	2	6	2	4
0x25	4	4	4	12	4	4	12	0	0	6	6	0	4	2	2	2
0x26	4	0	4	10	10	10	4	2	0	8	10	4	4	4	2	0
0x27	12	4	2	0	2	4	8	14	2	6	2	4	8	8	4	4
0x28	4	2	2	8	2	6	6	6	0	0	6	0	4	0	6	2
0x29	6	2	2	10	0	2	10	6	2	4	4	2	4	6	0	4
0x2A	6	2	4	6	0	2	4	6	2	0	6	6	2	6	2	2
0x2B	2	2	2	2	4	6	4	2	4	2	6	2	6	0	8	4
0x2C	0	2	2	4	0	2	8	10	2	2	4	8	8	4	2	6
0x2D	4	2	6	2	8	4	2	6	8	2	4	0	8	2	0	6
0x2E	4	6	2	2	8	2	2	2	6	4	0	2	12	2	6	4
0x2F	4	2	2	2	2	6	2	2	6	4	4	6	8	2	4	2

برای xor ورودی 64 حالت مختلف و برای xor خروجی 16 حالت مختلف وجود دارد. جدول بالا به این صورت تنظیم شده که برای هر حالت xor ورودی و خروجی جفت های ممکن از توابع S با هم xor شده و تعداد دفعاتی که این xor برابر با یکی از 16 حالت خروجی میشود در جدول ثبت می گردد مثلا برای xor ورودی که برابر 0X00 است، قطعا جفت های ورودی با هم برابر بوده اند، در نتیجه خروجی هر جفت نیز با هم برابر خواهند بود بنابراین برای تمام توابع S با احتمال $\frac{64}{64} = 1$ ، xor خروجی 0X00 خواهد بود.

برای درک بهتر حمله تفاضلی به DES این موضوع را با 3 دور توضیح می دهیم:

برای حمله به این سیستم از جفت متن های رمز شده ای استفاده میکنیم که نیمه راست xor متن اصلی متناظرشان برابر با صفر و xor نیمه چپ یک مقدار دلخواه باشد. xor نیمه چپ و راست متن رمز شده نیز مشخص است.

طبق توضیح هایی که دادیم و شکل زیر می توان به نتیجه های زیر رسید:



نتیجه ها:

Xor ورودی تابع F در دور اول برابر صفر و در نتیجه با احتمال یک، خروجی نیز صفر خواهد بود. ورودی تابع F در دور دوم یعنی $L_p \; xor \; 0_x = L_p$ است. ورودی تابع در دور سوم نیمه راست xor جفت متن رمز شده است. بنابراین خروجی تابع F در دور دوم برابر با $(0_x \; xor \; R_c)$ R_c خواهد بود. خروجی تابع F در دور سوم نیز از رابطه $L_p \; xor \; L_c$ به دست می اید.

اکنون فرض میکنیم که ورودی تابع S1 در دور سوم قبل از xor شدن با کلید برای یکی از متن های اصلی برابر با 0X01 و برای متن دیگر 0X35 باشد بنابراین xor ورودی تابع S1 برابر با 0X34 میشود.

فرض میکنیم xor خروجی تابع S1 ، $(L_c \; xor \; L_p)$ برابر با 0X0D باشد با توجه به جدول بالا، 8 حالت برای xor ورودی 0X34 و xor خروجی 0X0D وجود دارد. حالت های ممکن در ستون سمت چپ جدول برای " کلید های ممکن برای تبدل 0X34 به 0X0D توسط تابع S1 با ورودی های 0X01 و 0X35 " در پایین نشان داده شده است:

جفت ورودی تابع S		کلیدهای ممکن	
0x06	0x32	0x07	0x33
0x10	0x24	0x11	0x25
0x16	0x22	0x17	0x23
0x1C	0x28	0x1D	0x29

مثال: برای جفت ورودی 0X06 و 0X32 داریم:

0X32 xor 0X06 = 0X34

خروجی تابع S1 برای 0X06 برابر با 0X01 وبرای 0X32 برابر با 0X0C است پس:

0X01 xor 0X0C = 0X0D

برای به دست اوردن 6 بیت کلید مربوط به S1 کافی است xor ورودی S1 با 0X01 و 0X35 را حساب کنیم که میشود:

0X01 xor 0X06 = 0X07

0X01 xor 0X32 = 0X33

طبق جدول " کلید های ممکن برای تبدل 0X34 به 0X0D توسط تابع S1 با ورودی های 0X01 و 0X35 " مشخص است که هر سطر در ستون سمت چپ، دو جفت ورودی را نشان میدهد که دو کلید تولید می کنند. برای به دست اوردن مقدار واقعی کلید باید جفت های متعددی را به این صورت تست کنیم. کلیدی که بیشترین تکرار را داشته باشد با احتمال زیاد 6 بیت کلید S1 خواهد بود. برای به دست اودن بیت های دیگر کلید باید همین مراحل را برای سایر توابع S انجام دهیم.