

**Gebze Teknik Universitesi
Bilgisayar Muhendisligi**

CSE 470 - 2020

PROJE RAPORU

**LEMYE CEREN GUMUS
151044071**

GİRİS :

1. Problem Tanımları

1.1 AES şifreleme algoritmasının gerçekleştirilmesi ve şifreleme/deşifrelemede kullanılması(test verileri ile birlikte)

AES simetrik bir şifreleme algoritmasıdır yani hem şifreleme hem şifre çözme işlemleri için aynı anahtar kullanılır. AES için girdi ve çıktı matrisleri her zaman 128 bit olmak zorundadır ancak anahtar uzunluğu 128, 192 veya 256 bit olabilir.

S0	S4	S8	S12
S1	S5	S9	S13
S2	S6	S10	S14
S3	S7	S11	S15

Sekil ile gösterilen durum matrisinde her bir hücre 8 bit yer kaplamaktadır, 16 hücre bulunduğu için toplam 128 bitlik bir veriye karşılık düşer. Şifrelenecek mesaj ve anahtar durum matrisleri şeklinde düşünülerek üzerlerinde gerekli işlemler yapılır.

AES algoritması Bayt Değiştirme, Satır Kaydırma, Sütun Karıştırma ve Tur Anahtarı ile Toplama gibi adımların tekrar etmesi şeklinde düşünülebilir.

128 bit AES şifrelemesi için 10 tur sonunda şifrelenmiş mesaja ulaşılır.

1.1.A) Bayt Değiştirme

Bayt değiştirme işlemi durum matrisindeki baytların farklı baytlara dönüştürülmesi işlemidir. Bu dönüşüm daha önceden belli bir lookup table üzerinden yapılır. Örnek bir tablo aşağıda verilmiştir. Örneğin bayt değiştirme yapılacak olan değer 0x37 olsun, bayt değiştirme sonrası bu değer 0x9A (sbox'ın 0x37. değeri) değerine dönüşür.

Bayt degistirme matrisi :

```

BYTE_sbox[256] = {
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7,
0xab, 0x76,
0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4,
0x72, 0xc0,
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8,
0x31, 0x15,
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27,
0xb2, 0x75,
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3,
0x2f, 0x84,
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c,
0x58, 0xcf,
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c,
0x9f, 0xa8,
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff,
0xf3, 0xd2,
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d,
0x19, 0x73,
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e,
0x0b, 0xdb,
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95,
0xe4, 0x79,
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a,
0xae, 0x08,
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd,
0x8b, 0x8a,
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1,
0x1d, 0x9e,
0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
0x28, 0xdf,
0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54,
0xbb, 0x16};

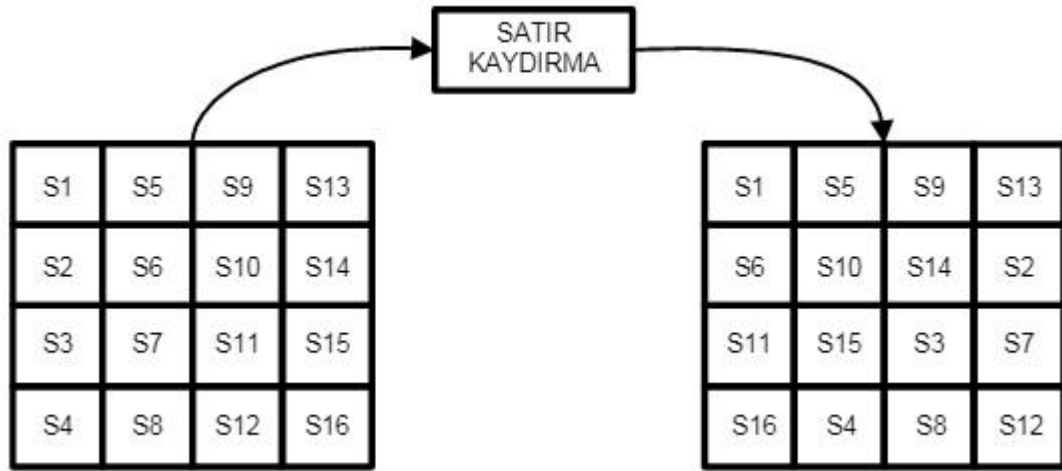
```

0x48	0xf6	0x24	0x5b		0x52	0x42	0x36	0x39
0xcc	0x1a	0xb7	0x34		0x4b	0xa2	0xa9	0x18
0x37	0xc9	0xd4	0x71		0x9a	0xdd	0x48	0xa3
0x1a	0x67	0x33	0x01		0xa2	0x85	0xc3	0x7c

Sekilde , Bayt deęiřtirme ncesi ve sonrası durum matrisi gosterilmiştir

1.1.B) Satır Kaydırma

Satır kaydırma iřlemi bayt deęiřtirme iřleminden sonra oluřan yeni durum zerine uygulanır. Durum matrisindeki satırların belli deęerde sola kaydırılması anlamına gelir.



				Kaydır				
0x48	0xf6	0x24	0x5b	0 bayt	0x48	0xf6	0x24	0x5b
0xcc	0x1a	0xb7	0x34	1 bayt	0x1a	0xb7	0x34	0xcc
0x37	0xc9	0xd4	0x71	2 bayt	0xd4	0x71	0x37	0xc9
0x1a	0x67	0x33	0x01	3 bayt	0x01	0x1a	0x67	0x33

Tablo 1.3.4.1 Satır kaydırma işlemi öncesi ve sonrası durum matrisi

1.1.C) Sütun Karıştırma

Sütun karıştırma işlemi satır karıştırma adımıyla oluşan durum matrisinin her sütununun ayrı ayrı belli bir matris ile çarpılması ve ortaya çıkan matrisin yeni sütun olarak kullanılmasıdır.

0x02	0x03	0x01	0x01
0x01	0x02	0x03	0x01
0x01	0x01	0x02	0x03
0x03	0x01	0x01	0x02

.

S0
S1
S2
S3

=

S0'
S1'
S2'
S3'

Tablo 1.3.5.1 Sütun Karıştırma

Bu çarpma işlemlerinden çıkan sonuç bazen 8 bitten büyük olabilir, böyle durumlarda çıkan sonuç indirgenmesi gerekir. Örneğin;

$$0xd4 * 0x02 = 11010100 * 00000010 = (x^7+x^6+x^4+x^2)*(x) = x^8+x^7+x^5+x^3$$

Görüldüğü gibi burada bulunan değer en az 9 bit ile ifade edilebilir. Bu nedenle sonucun 8. dereceden indirgenemez bir polinom($m(x)$) ile mod alınmalıdır. Bu polinomun böleni tektir ve sadece kendisidir.

$$m(x)=x^8+x^4+x^3+x+1= 100011011$$

Mod işlemi recursive olarak $m(x)$ fonksiyonunun modu alınacak sayının en yüksek dereceli basamağına kadar kaydırılması ve exorlanması ile gerçekleştirilir. Recursive işlemlerin son bulacağı şart ise çıkan sonucun 8 bit ile ifade edilebildiği noktadır.

$$0xd4 * 0x02 = x^8+x^7+x^5+x^3 = 110101000$$

$$\begin{array}{r} 110101000 \\ \oplus 100011011 \\ \hline 010110011 \end{array}$$

Bu örnekte mod işlemi tek iterasyonda tamamlandı ancak bazı örneklerde bu durum birkaç iterasyon sürebilir, ayrıca yeni sütundaki her bir bayt için ayrı ayrı 4 çarpma ve toplama işlemi olduğu için en kötü durumda her çarpmadan sonra mod işlemi uygulamak maliyeti artırmaktadır.

1.1.D) Tur Anahtarı ile Toplama

Her turun sonunda bulunan mesaj o anki tur anahtarı ile toplanır. Her turda fark bir anahtar kullanıldığı için tur sayısı kadar yeni anahtar gereklidir. Aşağıda 10 turluk bir AES 128 algoritmasında gerekli anahtar tablosunun bir kısmı gösterilmiştir.

0	1	2	3	4	5	6	7		42	43
0x61	0x74	0x6c	0x6b	0xf2	0x86	0xea	0x81	0xa3	0x6e
0x79	0x61	0x65	0x74	0x80	0xe1	0x84	0xf0	0x39	0x1b
0x73	0x74	0x63	0x69	0x8a	0xfe	0x9d	0xf4	0x4c	0x99
0x65	0x69	0x69	0x69	0x1a	0x73	0x1a	0x73	0x67	0xa8

4. sütun 2. anahtarın başlangıç noktasını ifade eder. 128 bitlik AES Algoritması için 11 adet farklı anahtara ihtiyaç vardır. Bunlardan ilki(belirlenen anahtar) en başta mesaj ile toplanır. Geriye kalan anahtarlar ilk anahtar yardımıyla üretilir ve her tur sonunda oluşan mesaj üzerine eklenir.

1.1.E) Tur Anahtarının Üretilmesi

Tur anahtarlarının üretilip sıralı bir şekilde bir bellek alanına yazılmasıyla oluşan diziye genişletilmiş anahtar dizisi denir. Genişletilmiş anahtar dizisi aşağıdaki algoritma ile bulunur.

N = Anahtardaki Kelime Sayısı

R = Tur Sayısı

K_i = İlk Anahtardaki Kelimeler, $i=0...(N-1)$

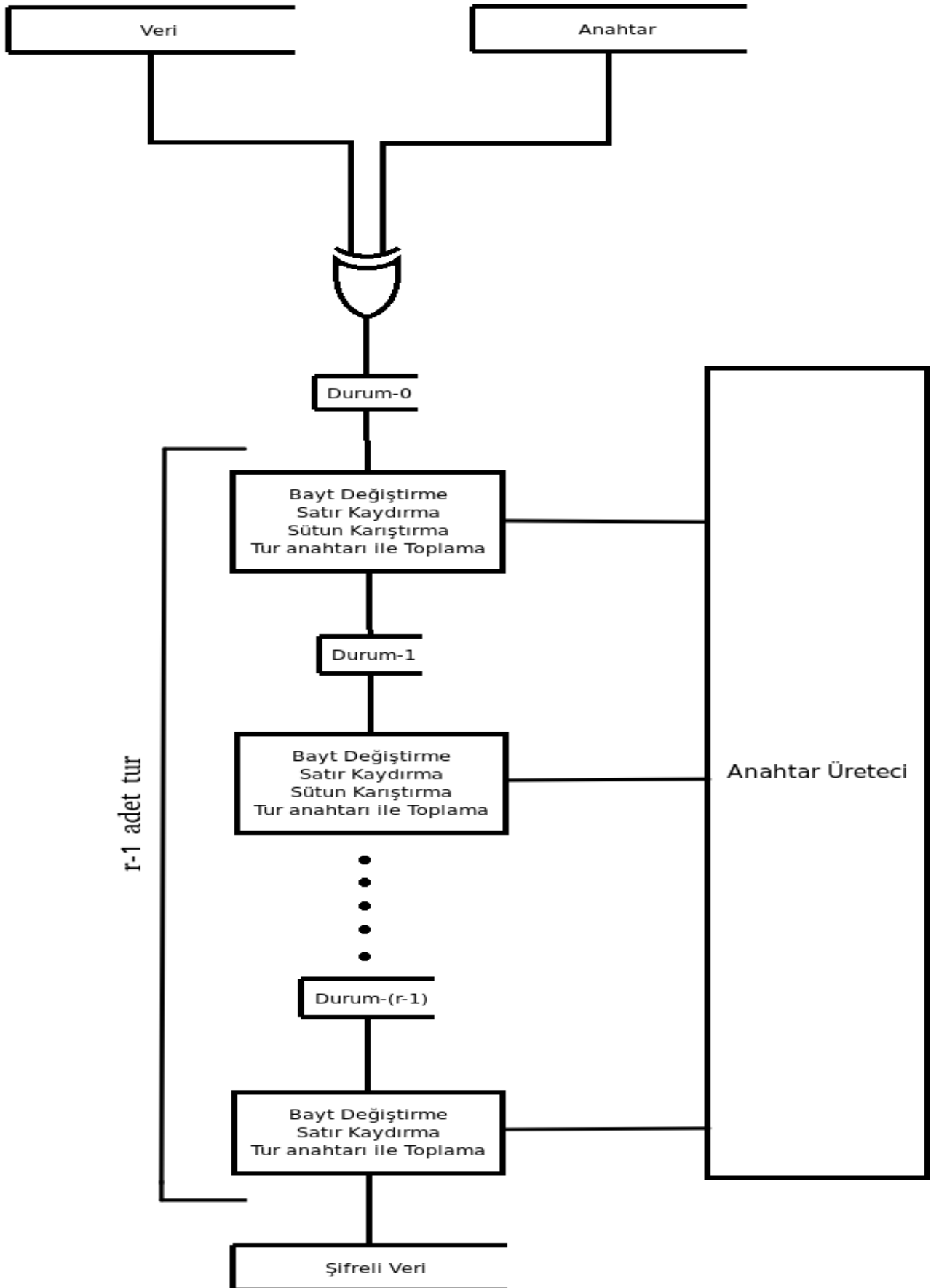
W_i = Genişletilmiş Anahtardaki Kelimeler, $i=0...4(R+1)-1$

$rcont = [0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a]$

$W_i = K_i$, Eğer $i < N$, $W_i = W_{i-N}$ Kaydır($SBox(W_{i-1})$) $rcon_i$, Eğer $i \geq N$ ve $i \neq 0 \pmod{N}$

$W_i = W_{i-N} SBox(W_{i-1})$, Eğer $i \geq N$, $N > 6$ ve $i \neq 0 \pmod{N}$

$W_i = W_{i-N} \oplus W_{i-1}$, diğer durumlar



Deşifreleme

Deşifreleme algoritmasının akışı yukarıdaki Şekildeki akış adımlarının terslerini içermektedir. Ters Bayt Değiştirme, Ters Satır Kaydırma, Ters Sütun Karıştırma ve Tur Anahtarıyla Toplama adımlarını içerir ancak Tur Anahtarıyla Toplama adımı genişletilmiş anahtarın içerisindeki son anahtardan başlayarak geriye doğru ilerler. Yani şifreleme için kullandığımız son anahtar deşifreleme için kullandığımız ilk anahtar olur.

Ters bayt değiştirme işlemi için bayt değiştirme işleminde olduğu gibi bir lookup table'dan faydalanılır. Bu tablodaki veriler bayt değiştirme tablosundaki verilerin tersleridir. Örneğin 0x00 değerinin bayt değiştirme tablosundaki karşılığı 0x63 iken, 0x63 değerinin ters bayt değiştirme adımındaki değeri 0x00'dır. Aşağıda örnek bir ters bayt değiştirme tablosu görülmektedir.

```
BYTE invSbox[256] = {
0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3,
0xd7, 0xfb,
0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde,
0xe9, 0xcb,
0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa,
0xc3, 0x4e,
0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b,
0xd1, 0x25,
0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65,
0xb6, 0x92,
0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d,
0x9d, 0x84,
0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3,
0x45, 0x06,
0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13,
0x8a, 0x6b,
0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4,
0xe6, 0x73,
0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75,
0xdf, 0x6e,
0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18,
0xbe, 0x1b,
0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd,
0x5a, 0xf4,
0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80,
0xec, 0x5f,
0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9,
0x9c, 0xef,
0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53,
0x99, 0x61,
0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21,
0x0c, 0x7d};
```


Ters satır kaydırma adımında, satır kaydırma adımında yapılan işlemler sağa kaydırılarak tekrarlanır.

				Kaydır				
0x48	0xf6	0x24	0x5b	0 bayt	0x48	0xf6	0x24	0x5b
0xcc	0x1a	0xb7	0x34	1 bayt	0x34	0xcc	0x1a	0xb7
0x37	0xc9	0xd4	0x71	2 bayt	0xd4	0x71	0x37	0xc9
0x1a	0x67	0x33	0x01	3 bayt	0x67	0x33	0x01	0x1a

Tablo 1.3.8.2 Ters Satır Kaydırma İşlemi

Ters sütun karıştırma adımında ise yine sütun karıştırma adımı benzer işlemler yapılır ancak bu kez aşağıdaki matris kullanılır.

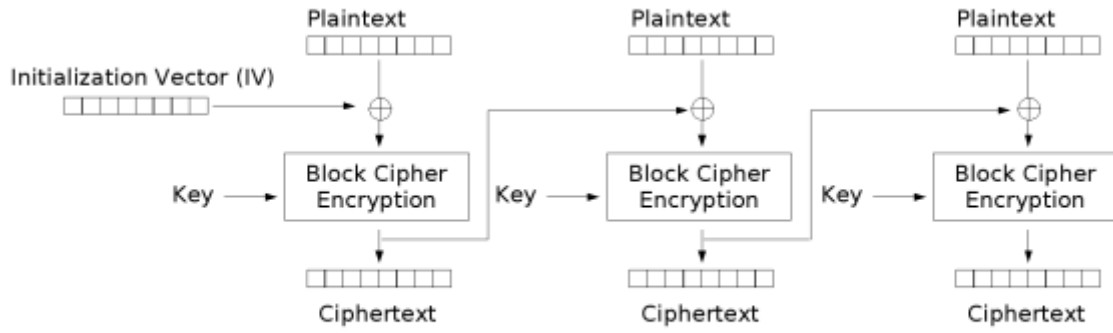
0x0e	0x0b	0x0d	0x09
0x09	0x0e	0x0b	0x0d
0x0d	0x09	0x0e	0x0b
0x0b	0x0d	0x09	0x0e

Tablo 1.3.8.3 Ters Sütun Karıştırma

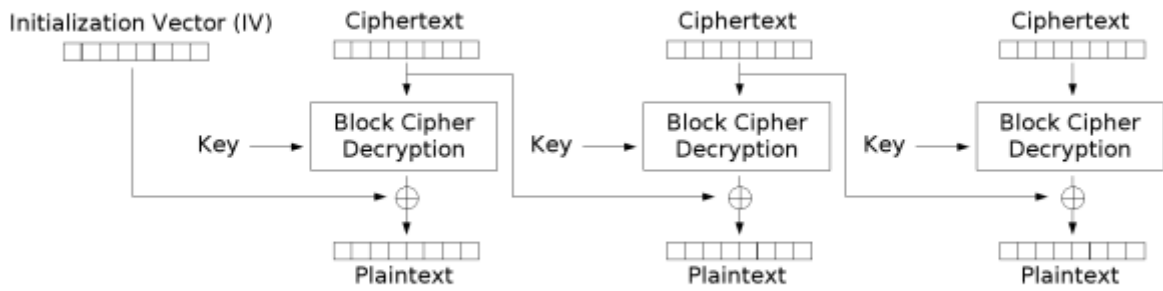
1.2. Gerçeklenen Şimetrik şifreleme algoritması kullanılarak CBC ve OFB modlarında çalışmayı gerçekleyip testlerini yapacak şekle getiriniz)

Şifre-bloku zincirleme (CBC)

CBC kipinde her açık metin bloku şifrelenmeden önce bir önceki kapalı metin bloku ile XORlanır. Bu sayede her kapalı metin bloku kendisinden önce gelen tüm açık metinlere bağımlı olmuş olur. Bir mesajın aynı anahtar altında tekrar şifrelendiğinin anlaşılabilmesi için ilk blokta ilklendirme vektörü (IV) kullanılmalıdır.



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

İlk blokun indeksine 1 dersek, CBC kipinin matematiksel ifadesi şu şekilde olur

$$C_i = E_K(P_i \oplus C_{i-1}), C_0 = IV$$

CBC kipi ile şifrelenmiş metnin deşifreleme işlemi de şu şekilde yapılır

$$P_i = D_K(C_i) \oplus C_{i-1}, C_0 = IV.$$

CBC kipi en yaygın olarak kullanılan kip olmuştur. Bu kipi en büyük dezavantajları blokların birbirine bağımlı olmalarından dolayı paralel olarak işlenememeleri ve tamamlama gerektirmesidir. Tamamlama sorununu çözmek için kapalı metin çalma yöntemi önerilmiştir.

Farklı bir IV ile deşifreleme yapmaya çalışmak ilk açık metin blokunun bozulmasına yol açarken diğer bloklar doğru şekilde deşifrelenecektir. Bunun sebebi bir açık metin blokunun kendisi ve önce gelen kapalı metin bloklarından çıkarılabilmesidir. Sonuç olarak deşifreleme işlemi paralelleştirilebilir. Bir kapalı metin bitindeki hata o bloktaki açık metni tamamen bozar ve sonraki bloktaki açık metinde de karşılık gelen bitlerde hataya yol açar, ancak sonraki bloklarda bir bozulma olmaz.

Çıktı geri bildirim(OFB)

Çıktı geri bildirim kipi bir blok şifreyi bir senkron akış şifre yapar. Anahtar akışı blokları oluşturur, bunlar daha sonra şifreli metin üretmek için şifresiz metin

bloklarıyla XORlanır. Diğer akış şifrelerinde olduğu gibi, şifreli metinde bir bit döndürmek şifresiz metinde aynı konumda döndürülmüş bit üretir. Bu özellik, şifrelemeden önce uygulandığında bile hata düzeltme kodlarının normal çalışmasına izin verir.

XOR operasyonunun simetri özelliğinden dolayı şifreleme ve şifre çözme aynıdır:

$$C_j = P_j \oplus O_j,$$

$$P_j = C_j \oplus O_j,$$

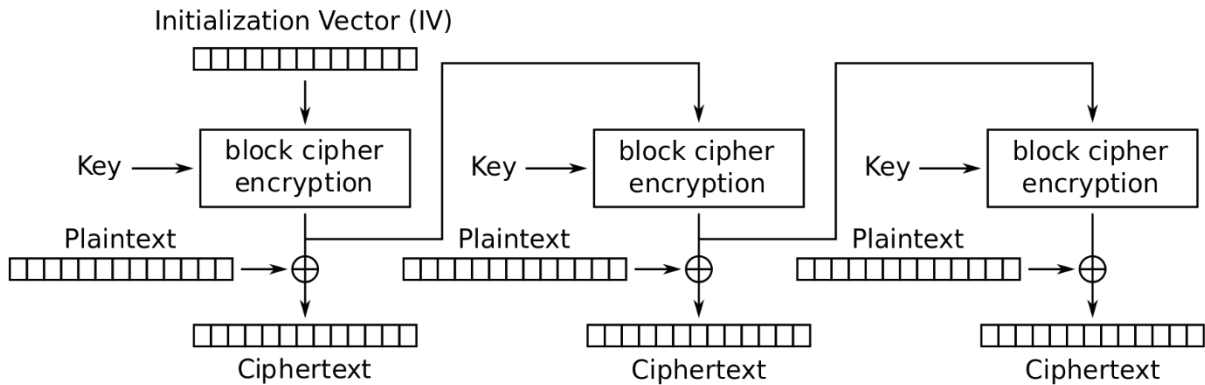
$$O_j = E_K(I_j),$$

$$I_j = O_{j-1},$$

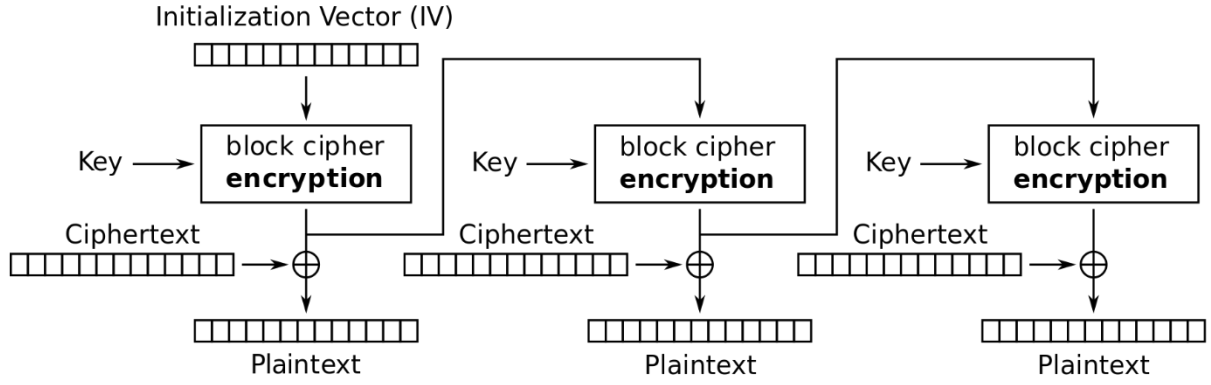
$$I_0 = IV.$$

Çıktı geri bildirim blok şifreleri operasyonları öncekilerin hepsine bağlıdır, bu yüzden paralel olarak çalıştırılmaz. Ancak, şifreli metin veya şifresiz metin sadece son XOR için kullanıldığından, blok şifreleme operasyonu işlemleri önceden gerçekleştirilebilir, son adımın şifreli veya şifresiz metin müsait olduğunda paralel çalıştırılabilmesine izin verilir.

CBC kipi ile girdi olarak sıfırlardan oluşan bir sabit dizi kullanarak OFB kipi anahtar akışı elde etmek mümkündür. Bu kullanışlı olabilir çünkü OFB kipi şifrelemesi için CBC kipi hızlı donanım implementasyonlarının kullanılmasına izin verir.



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

1.3. Herhangi bir doküman (.doc/.docx, .pdf, ppt, xls vs) üzerinde değişiklik yapıp yapılmadığını ve yapanın kimliğini anlamak için, özütünü alacak ve sadece işlem yapan kişinin bildiği bir anahtar ile şifreleyip dosyanın sonuna ekleyecek bir araç (b şıkkındaki gerçeklemeyi özüt fonk. Olarak kullanınız)

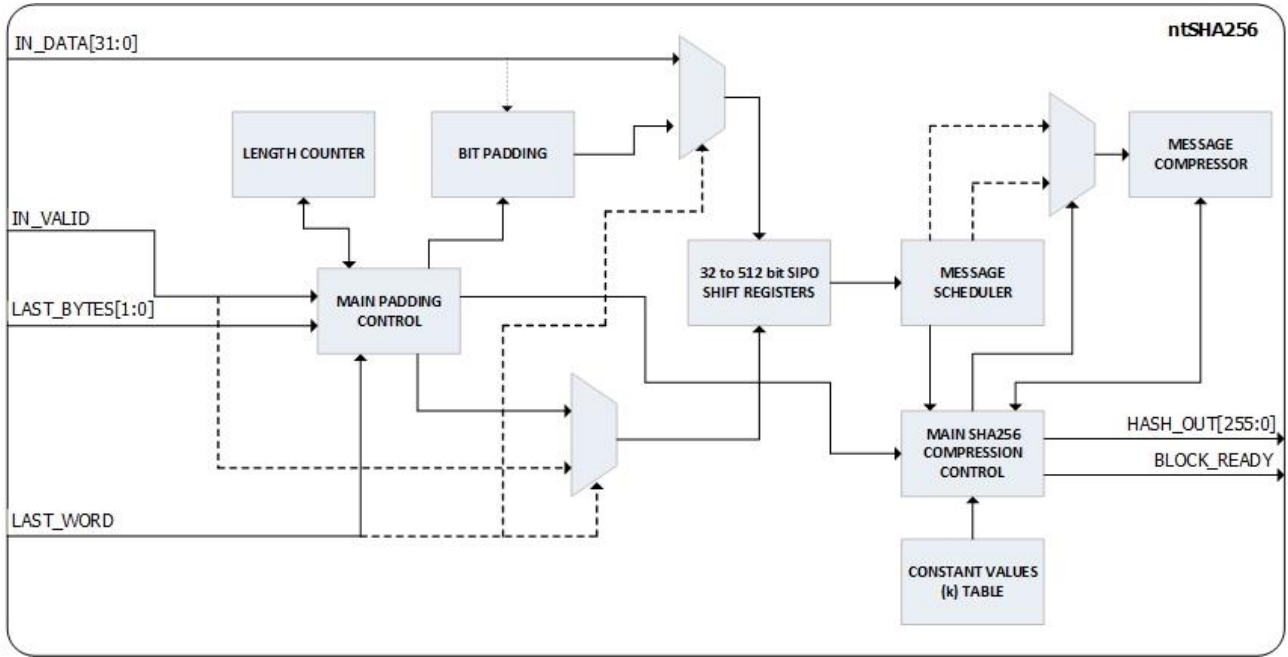
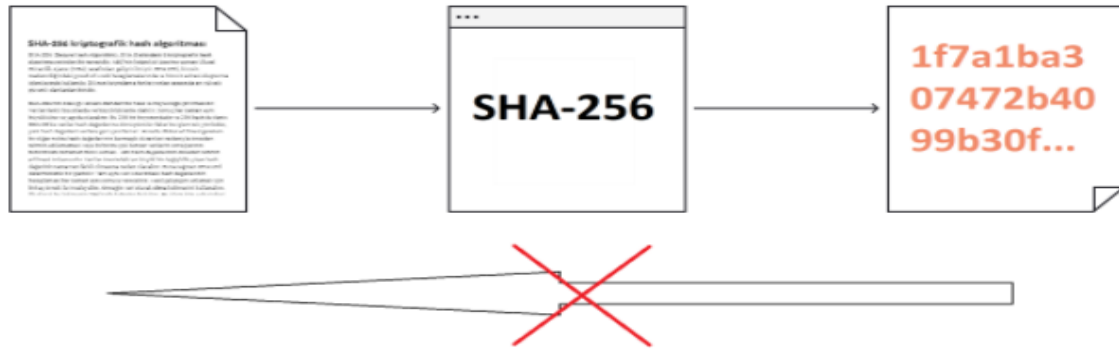
Girilen veriyi, sabit uzunlukta çıktıya dönüştüren matematiksel işleme hash denir. Bu işlemin amaçlarından biri, verinin gizlenmesidir. Söz gelimi, web sitelerine üye olurken yazılan şifreler hash'e dönüştürülerek veri tabanına yazılır. Bu sayede veri tabanını inceleyen kişi kullanıcının şifresini bilemez.

Bir diğer kullanım amacı ise verinin güvenli özetinin oluşturulmasıdır. Girdi verisi ne kadar uzun olursa olsun çıktı daima aynı uzunlukta olacağından, hash kodu özet amaçlı saklanabilir.

Bu kısımda hash yapmak için sha256 algoritması kullanılmıştır

SHA256 ALGORİTMASI:

SHA-256'nın özelliği verileri standart bir hale ve büyüklüğe çevirmesidir. Veriler farklı boyutlarda ve büyüklüklerde olabilir. Sonuç her zaman aynı büyüklükte ve yapıda olacaktır. Bu 256 bit (**32 byte - 64 hexadecimal**) boyutundadır ve 256 hash'dir. SHA-256'da veriler hash değerlerine dönüştürülür fakat bu işlem tek yönlüdür, yani hash değerleri verilere geri çevrilemez. Burada dikkat edilmesi gereken bir diğer nokta hash değerlerinin karmaşık düzenleri nedeniyle önceden tahmin edilememesi veya birbirine çok benzer verilerin sonuçlarının birbirinden tamamen farklı olması. Yani hash değerlerinin önceden tahmin edilmesi imkansızdır. Veriler üzerindeki en küçük bir değişiklik çıkan hash değerinin tamamen farklı olmasına neden olacaktır. Buna rağmen SHA-256 deterministik bir işlemdir. Yani aynı veri üzerindeki hash değerlerinin hesaplanması her zaman aynı sonucu verecektir.



Not :Dosyanin icerigi SHA-256 algoritmasi ile hashlenip(ozutu alinip) AES sifreleme algoritmasi ile OFB ve CBC modlarında sifrelenmistir. Sifreleme islemi kullanicidan alınan key bilgisiyle saglanmistir.Key kullanicici icin bir password olusturur.

1.4 Dosyanın bütünlüğünün değişip değişmediğinin kontrolü için, c'deki işlemleri yaparak ilk üretilen özüt değeri ile karşılaştıran doğrulama aracını gerçekleyerek örnek testleri gösteriniz

Part 3 – c de gerçekenlen işlemleri test edebilmek için belirli test case ler olusturulmustur. Sifreleme yapmadan önce , dosyanin hash (özütunu alip); Sifreleme yaptıktan sonra degisen dosya hash(özütü) ile karsilastirir. Bu sekilde data integrity (veri butunlugunun) degisip degismediginin kontrolunu saglar.

2 METHOD

2.1 Kullanım durumları

PART1_A :

Python main.py

PART2_B :

Python main.py

PART3_C :

Python main.py aes_mode file_path user_key

e.g : Python main.py cbc ceren.txt x011cbc

Python main.py ofb ceren.txt x011ofb

User_key : sadece kullanıcıların bildiği bir KEY

File_path : şifrelenmesi istenilen kullanıcının istediği ,kullandığı dosya

Aes_mode: şifrelenmek istenen mod e.g cbc,ofb

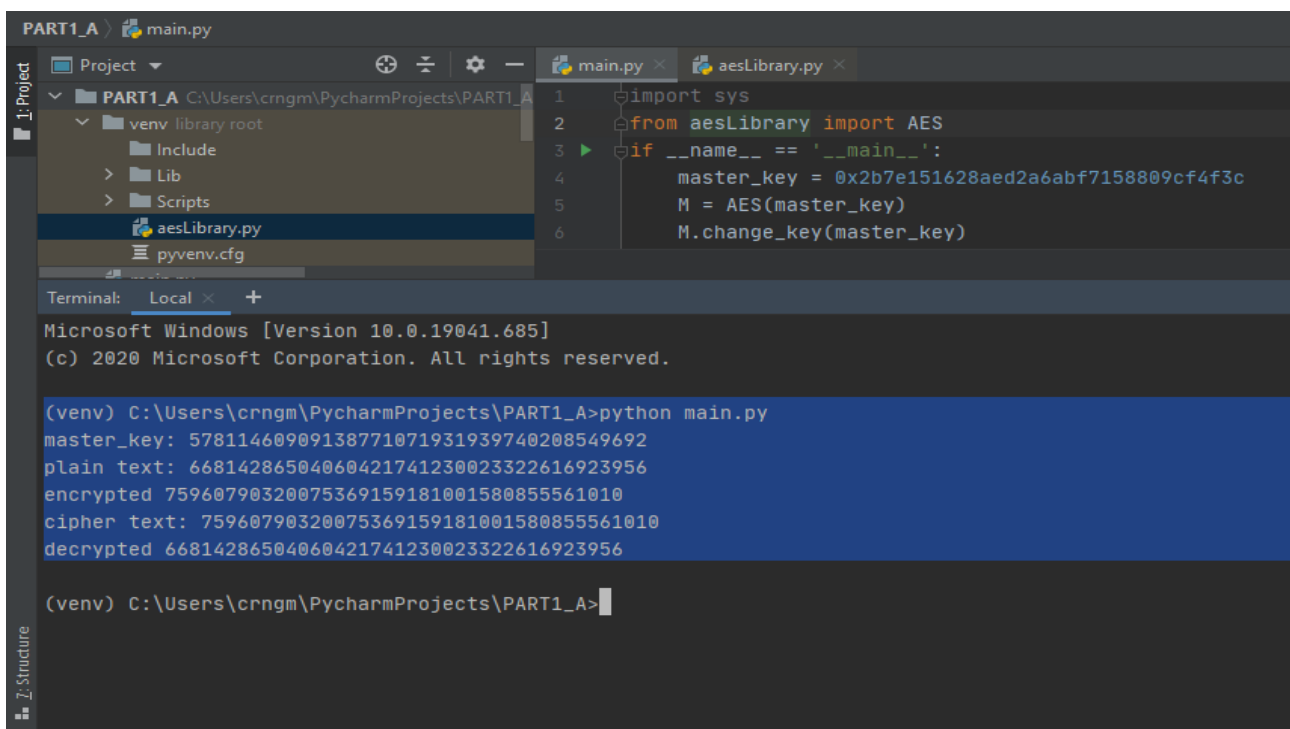
PART4_D :

Python main.py

PROJE SONUÇLARI:

PART1_A :

Main.py running result....



```
PART1_A main.py
Project
  PART1_A C:\Users\crngm\PycharmProjects\PART1_A
    venv library root
      Include
      Lib
      Scripts
    aesLibrary.py
    pyenv.cfg
Terminal: Local
Microsoft Windows [Version 10.0.19041.685]
(c) 2020 Microsoft Corporation. All rights reserved.

(venv) C:\Users\crngm\PycharmProjects\PART1_A>python main.py
master_key: 57811460909138771071931939740208549692
plain text: 66814286504060421741230023322616923956
encrypted 75960790320075369159181001580855561010
cipher text: 75960790320075369159181001580855561010
decrypted 66814286504060421741230023322616923956

(venv) C:\Users\crngm\PycharmProjects\PART1_A>
```

PART2_B :

Main.py running result....

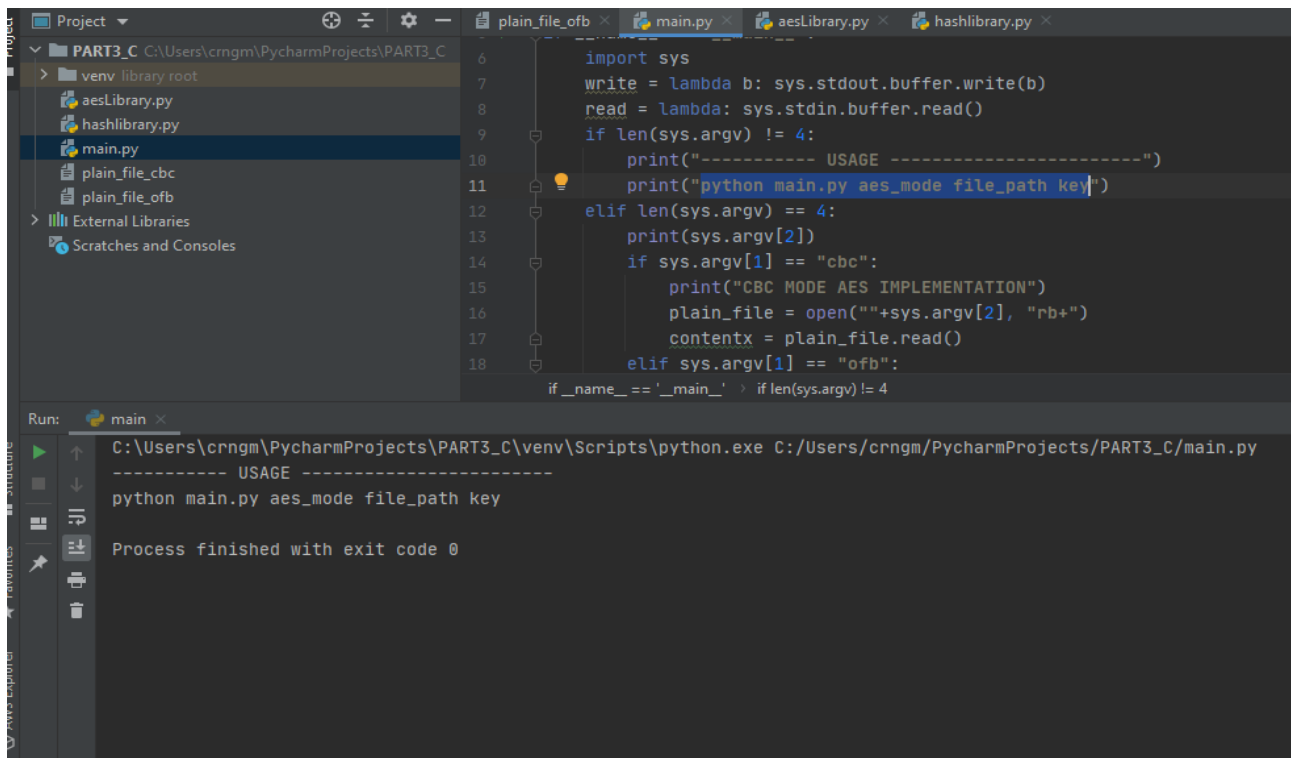
```
PART2_B > aesLibrary.py
Project
PART2_B C:\Users\crngm\PycharmProjects\PART2_B
  venv library root
    Include
    Lib
    Scripts
    pyvenv.cfg
    aesLibrary.py
    main.py
main.py x aesLibrary.py x
211
212     return b''.join(blocks)
213
214 def decryptOFB(self, ciphertext, iv):
215     print("DECRYPT OFB MODE")
216     assert len(iv) == 16
217
AES > decryptOFB()
Terminal: Local x +
(venv) C:\Users\crngm\PycharmProjects\PART2_B>python main.py
<class 'bytes'>
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
CBC MODE AES IMPLEMENTATION
CBC MESSAGE : Hello world
ENCRYPT CBC MODE
DECRYPT CBC MODE
cbc mode ciphertext_cbc: b'\xa9\x13\xe0<\xaf\x80\xa8gm\x7f\xe1\xafn\x07\x19\xd8'
cbc mode plain text : b'Hello world'

OFB MODE AES IMPLEMENTATION
OFB MESSAGE : Hello world
ENCRYPT OFB MODE
DECRYPT OFB MODE
ofb mode cipher text: b'\xa9(1b\x8dWb\xb0z\xd8q'
ofb mode plain text : b'Hello world'

(venv) C:\Users\crngm\PycharmProjects\PART2_B>
```

PART3_C :

Main.py running result....

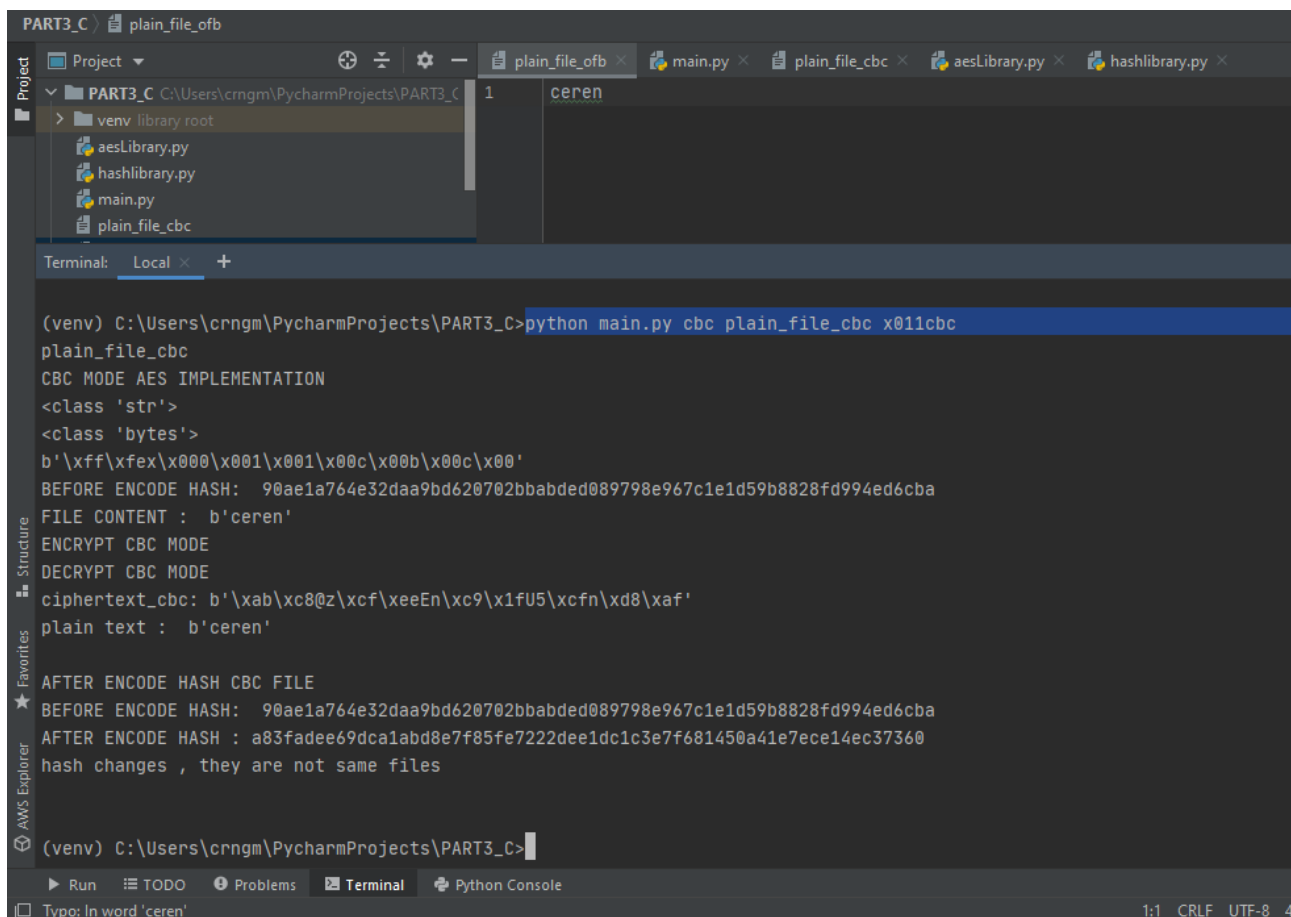


The screenshot shows the PyCharm IDE with the project 'PART3_C' open. The file explorer on the left shows the project structure, including 'main.py'. The main.py file is open in the editor, showing the following code:

```
import sys
write = lambda b: sys.stdout.buffer.write(b)
read = lambda: sys.stdin.buffer.read()
if len(sys.argv) != 4:
    print("----- USAGE -----")
    print("python main.py aes_mode file_path key")
elif len(sys.argv) == 4:
    print(sys.argv[2])
    if sys.argv[1] == "cbc":
        print("CBC MODE AES IMPLEMENTATION")
        plain_file = open(sys.argv[2], "rb+")
        contentx = plain_file.read()
    elif sys.argv[1] == "ofb":
        if __name__ == '__main__':
            if len(sys.argv) != 4:
```

The Run window at the bottom shows the execution of 'main.py' with the following output:

```
C:\Users\crngm\PycharmProjects\PART3_C\venv\Scripts\python.exe C:/Users/crngm/PycharmProjects/PART3_C/main.py
----- USAGE -----
python main.py aes_mode file_path key
Process finished with exit code 0
```



The screenshot shows the PyCharm IDE with the project 'PART3_C' open. The file explorer on the left shows the project structure, including 'main.py'. The main.py file is open in the editor, showing the following code:

```
1 ceren
```

The Terminal window at the bottom shows the execution of the command:

```
(venv) C:\Users\crngm\PycharmProjects\PART3_C>python main.py cbc plain_file_cbc x011cbc
```

The output of the command is as follows:

```
plain_file_cbc
CBC MODE AES IMPLEMENTATION
<class 'str'>
<class 'bytes'>
b'\xff\xfex\x00\x001\x001\x00c\x00b\x00c\x00'
BEFORE ENCODE HASH: 90ae1a764e32daa9bd620702bbabded089798e967c1e1d59b8828fd994ed6cba
FILE CONTENT : b'ceren'
ENCRYPT CBC MODE
DECRYPT CBC MODE
ciphertext_cbc: b'\xab\xc8@z\xcf\xeeEn\xc9\x1fU5\xcf\xfd8\xaf'
plain text : b'ceren'
AFTER ENCODE HASH CBC FILE
BEFORE ENCODE HASH: 90ae1a764e32daa9bd620702bbabded089798e967c1e1d59b8828fd994ed6cba
AFTER ENCODE HASH : a83fadee69dca1abd8e7f85fe7222dee1dc1c3e7f681450a41e7ece14ec37360
hash changes , they are not same files
(venv) C:\Users\crngm\PycharmProjects\PART3_C>
```


File Edit View Navigate Code Refactor Run Tools VCS Window Help PART3_C [C:\Users\crngm\PycharmProjects\PART3_C] - ...\plain_file_cbc

Project PART3_C C:\Users\crngm\PycharmProjects\PART3_C

- venv library root
- aesLibrary.py
- hashLibrary.py
- main.py
- plain_file_cbc
- plain_file_ofb
- External Libraries

plain_file_cbc

Visual layout of bidirectional text can depend on base direction (set in View menu) Choose direction Hide

File was loaded in the wrong encoding: 'UTF-8' Reload in 'windows-1252' Set project encoding to 'windows-1252'

```
1 cerenLH%EM[x011cbc
```

Terminal: Local +

```
plain_file_cbc
CBC MODE AES IMPLEMENTATION
<class 'str'>
<class 'bytes'>
b'\xff\xfe\x00\x01\x001\x00c\x00b\x00c\x00'
BEFORE ENCODE HASH: 90ae1a764e32daa9bd620702bbabded089798e967c1e1d59b8828fd994ed6cba
FILE CONTENT : b'ceren'
ENCRYPT CBC MODE
DECRYPT CBC MODE
ciphertext_cbc: b'\xab\xc0z\xcf\xeeEn\xc9\x1fU5\xcf\x08\xaf'
plain text : b'ceren'

AFTER ENCODE HASH CBC FILE
BEFORE ENCODE HASH: 90ae1a764e32daa9bd620702bbabded089798e967c1e1d59b8828fd994ed6cba
AFTER ENCODE HASH : a83fadee69dca1abd8e7f85fe7222dee1dc1c3e7f681450a41e7ece14ec37360
hash changes , they are not same files
```

PART3_C plain_file_cbc

Project PART3_C C:\Users\crngm\PycharmProjects\PART3_C

- venv library root
- aesLibrary.py
- hashLibrary.py

plain_file_ofb

```
1 cerenLH%EM[x011cbc
```

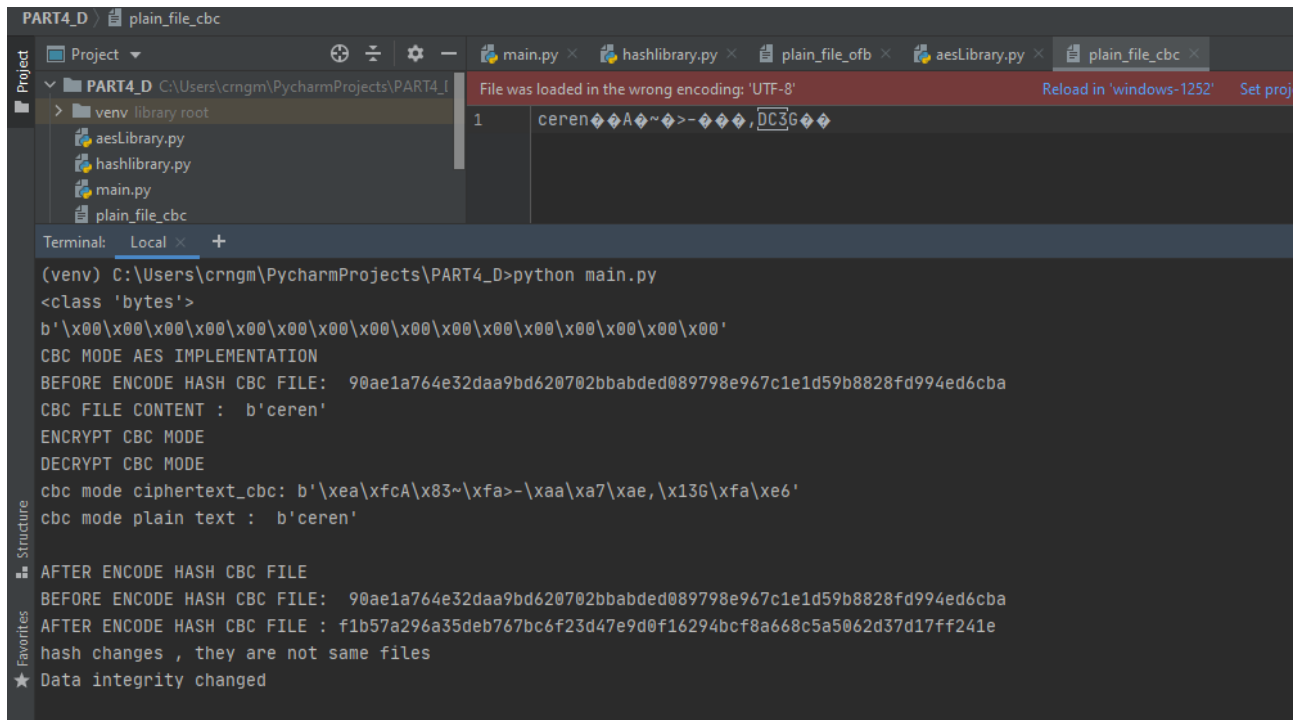
Terminal: Local +

```
(venv) C:\Users\crngm\PycharmProjects\PART3_C>python main.py ofb plain_file_ofb x011cbc
plain_file_ofb
OFB MODE AES IMPLEMENTATION
plain_file_ofb
<class 'str'>
<class 'bytes'>
b'\xff\xfe\x00\x01\x001\x00c\x00b\x00c\x00'
BEFORE ENCODE HASH: 90ae1a764e32daa9bd620702bbabded089798e967c1e1d59b8828fd994ed6cba
FILE CONTENT : b'ceren'
ENCRYPT OFB MODE
ENCRYPT OFB MODE
ciphertext_cbc: b'LH%\x19['
plain text : b'ceren'

AFTER ENCODE HASH CBC FILE
BEFORE ENCODE HASH: 90ae1a764e32daa9bd620702bbabded089798e967c1e1d59b8828fd994ed6cba
AFTER ENCODE HASH : b6fc85df51984fc9e8186a8d3943c9aa0b2b02550e384b946a4435885a2cdbcdb
hash changes , they are not same files
```

PART4_D :

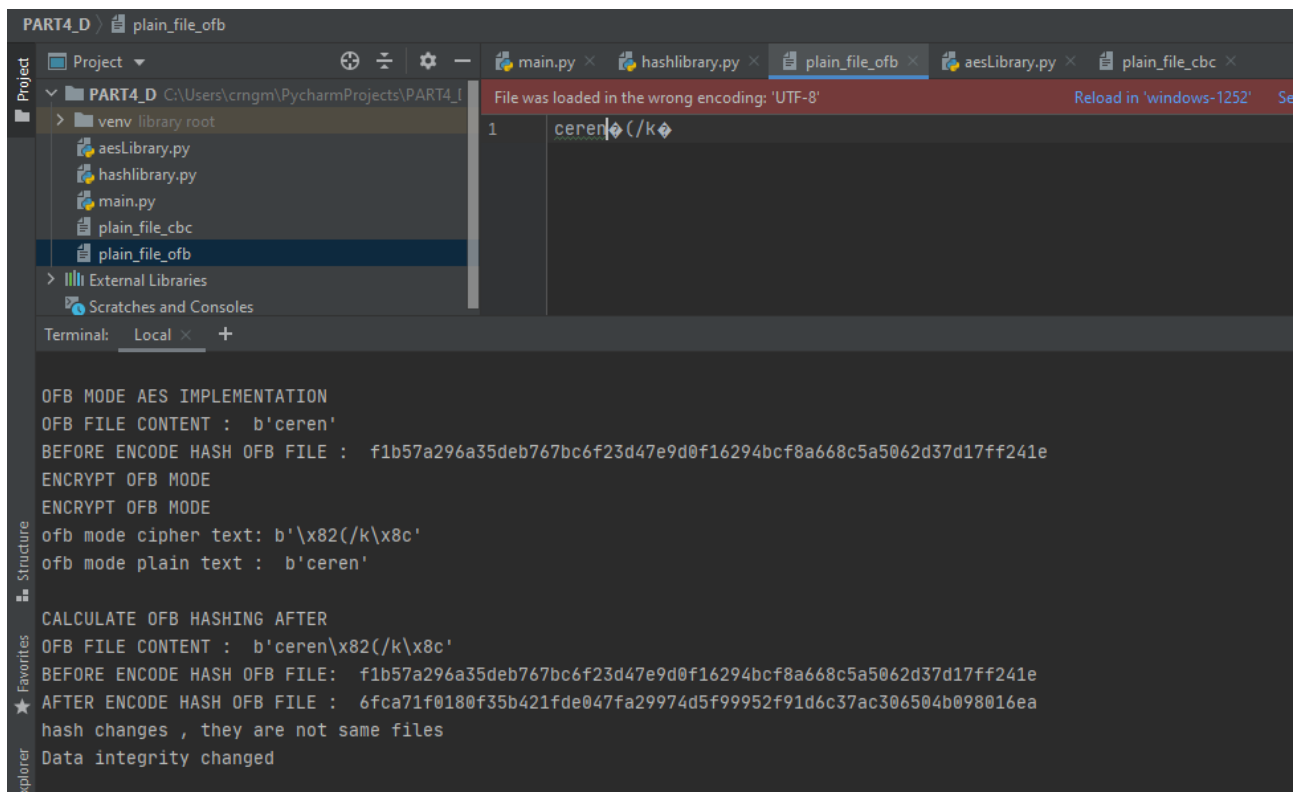
Main.py running result....



```
PART4_D > plain_file_cbc
File was loaded in the wrong encoding: 'UTF-8' Reload in 'windows-1252' Set proj
1 ceren

Terminal: Local x +
(venv) C:\Users\crngm\PycharmProjects\PART4_D>python main.py
<class 'bytes'>
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
CBC MODE AES IMPLEMENTATION
BEFORE ENCODE HASH CBC FILE: 90ae1a764e32daa9bd620702bbabded089798e967c1e1d59b8828fd994ed6cba
CBC FILE CONTENT : b'ceren'
ENCRYPT CBC MODE
DECRYPT CBC MODE
cbc mode ciphertext_cbc: b'\xea\xfa\x83~\xfa>-\xaa\xa7\xae,\x13G\xfa\xe6'
cbc mode plain text : b'ceren'

AFTER ENCODE HASH CBC FILE
BEFORE ENCODE HASH CBC FILE: 90ae1a764e32daa9bd620702bbabded089798e967c1e1d59b8828fd994ed6cba
AFTER ENCODE HASH CBC FILE : f1b57a296a35deb767bc6f23d47e9d0f16294bcf8a668c5a5062d37d17ff241e
hash changes , they are not same files
★ Data integrity changed
```



```
PART4_D > plain_file_ofb
File was loaded in the wrong encoding: 'UTF-8' Reload in 'windows-1252' Se
1 ceren(/k

Terminal: Local x +

OFB MODE AES IMPLEMENTATION
OFB FILE CONTENT : b'ceren'
BEFORE ENCODE HASH OFB FILE : f1b57a296a35deb767bc6f23d47e9d0f16294bcf8a668c5a5062d37d17ff241e
ENCRYPT OFB MODE
ENCRYPT OFB MODE
ofb mode cipher text: b'\x82(/k\x8c'
ofb mode plain text : b'ceren'

CALCULATE OFB HASHING AFTER
OFB FILE CONTENT : b'ceren\x82(/k\x8c'
BEFORE ENCODE HASH OFB FILE: f1b57a296a35deb767bc6f23d47e9d0f16294bcf8a668c5a5062d37d17ff241e
AFTER ENCODE HASH OFB FILE : 6fca71f0180f35b421fde047fa29974d5f99952f91d6c37ac306504b098016ea
hash changes , they are not same files
Data integrity changed
```