**EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ**

**(YÜKSEK LİSANS TEZİ)**

# AESX$^{+}$, AESX, AES-128 VE AES-256'NIN PERFORMANS VE GÜVENLİK AÇISINDAN KARŞILAŞTIRILMASI

**Mehmet TÜFEKÇİ**

**Tez Danışmanı: Prof.Dr. Mehmet Emin DALKILIÇ**

**Uluslararası Bilgisayar Anabilim Dalı**

**Bilim Dalı Kodu: 619.02.04**
**Sunuş Tarihi: 11.01.2018**

**Bornova-İZMİR**
**2018**

Mehmet TÜFEKÇİ tarafından yüksek lisans tezi olarak sunulan "AESX$^{+}$, AESX, AES-128 ve AES-256'nın Performans ve Güvenlik Açısından Karşılaştırılması" başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 11.01.2018 tarihinde yapılan tez savunma sınavında aday oybirliği/~~oyçokluğu~~ ile başarılı bulunmuştur.

| **Jüri Üyeleri:** | | **İmza** |
|---|---|---|
| **Jüri Başkanı** | **: Prof.Dr. M.Emin DALKILIÇ** | ..... |
| **Raportör Üye** | **: Doç.Dr. Müge SAYIT** | ..... |
| **Üye** | **: Y.Doç.Dr. Gökhan DALKILIÇ** | ..... |

## EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

## ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi olarak sunduğum "AESX$^{+}$, AESX, AES-128 ve AES-256'nın Performans ve Güvenlik Açısından Karşılaştırılması" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

11 / 01 / 2018

Mehmet TÜFEKÇİ

# ÖZET

## AESX$^{+}$, AESX, AES-128 VE AES-256'NIN PERFORMANS VE GÜVENLİK AÇISINDAN KARŞILAŞTIRILMASI

TÜFEKÇİ, Mehmet

Yüksek Lisans Tezi, Uluslararası Bilgisayar Anabilim Dalı
Tez Danışmanı: Prof.Dr. Mehmet Emin DALKILIÇ
Ocak 2018, 83 sayfa

Bu tezde AES (Advanced Encryption Standard; Gelişmiş Şifreleme Standardı) algoritmasının farklı anahtar uzunluklarındaki versiyonları olan AES-128 ve AES-256'ya ek olarak anahtar beyazlatma kullanan DESX'den esinlenen AESX ve AESX'den anahtar seçiminde farklılaşan AESX+ adını verdiğimiz yeni bir modelin performans ve güvenlik açısından karşılaştırılması yapılarak birbirlerine olan üstünlükleri ve zayıflıkları incelenmiştir.

Yapılan performans testlerinde beklendiği üzere AES-256, AES-128'den yaklaşık yüzde 20 daha yavaş çalışmaktadır. Önerilen AESX ve AESX+ algoritmaları ise standart crypto kütüphaneleri kullanılarak yapılan implementasyonlarda AES-128'den yaklaşık %12 daha yavaş, AES-256'dan ise yaklaşık %8 daha hızlı çalışmıştır.

Güvenlik analizinde öncelikle önerilen Kilian ve Rogaway'in ünlü makalelerinde DESX için verilen model AESX ve AESX+ algoritmaları uyarlanarak anahtar arama saldırısına karşı AESX ve AESX+ algoritmalarının temel olarak AES-256 ile benzer güvenlik seviyesine sahip olduğu gösterilmiştir. Öte yandan, Phan ve Shamir'in İlişkili Anahtar Saldırısı için yapılan analizde ise AESX algoritmasının AES-128'den daha yüksek dirence sahip olduğu, güvenlik seviyesinin AES-256'ya yakın olduğu tespit edilmiştir.

**Anahtar sözcükler:** Şifreleme, AES, AESX, Anahtar arama saldırısı, ilişkili anahtar saldırısı.

# ABSTRACT

## COMPARISON OF AESX$^+$, AESX, AES-128 AND AES-256 IN TERMS OF PERFORMANCE AND SECURITY

TÜFEKÇİ, Mehmet

MSc in International Comp Ins.
Supervisor: Prof. Dr. Mehmet Emin DALKILIÇ
January 2018, 83 pages

In this thesis, in addition to the AES (Advanced Encryption Standard) algorithm's version of AES-128 and AES-256, which are on the different key lengths. AESX which is inspired from DESX and AESX$^+$ which differs from AESX in its key use have been compared in terms of performance and security and their advantages and disadvantages has been studied. AESX and AESX$^+$ employs two XOR functions one before and one after the AES encryption block, a technique known as "key whitening".

Performans tests show that, as expected, AES-256 (AES with 256 bit key) executes approximately 20% slower than AES-128 (AES with 128 bit key). The two construct offered in this thesis, AESX and AESX$^+$ when implemented using standart crypto libraries, execute approximately 20% slower than AES-128 while running 8% faster than AES-256.

In the security analysis we first adopted Kilian and Rogaway's DESX security model against exhaustive key search attack presented in their famous paper to AESX and AESX$^+$. We have showed that AESX and AESX$^+$ algorithms basically have similar security levels against exhaustive key search attack. On the other hand, in the analysis for Phan and Shamir's related key attack has shown that AESX$^+$ construct has a much higher resistance level against this attack as compared to AES-128 and AESX constructs. Analysis indicate that AESX$^+$'s security level against this attack is close to AES-256 algorithm.

**Keywords:** Encryption, AES, AESX, Key search attack, Related key attack.

# TEŞEKKÜR

Tez çalışmam süresince, yardımlarından ve katkılarından dolayı değerli hocam ve danışmanım Sayın Prof. Dr. Mehmet Emin DALKILIÇ'a ve yüksek lisans eğitimim boyunca bilgi teknolojileri ile ilgili bana büyük katkılarından dolayı Uluslararası Bilgisayar Enstitüsü öğretim görevlilerine sonsuz teşekkürlerimi sunarım.

Ayrıca, bana çalışmalarım esnasında destek olan ve güç veren eşim Mine ULUDAĞ TÜFEKÇİ'ye, bu süreçte beni motive eden kızım Defne Serra TÜFEKÇİ'ye, anneme ve babama teşekkürü bir borç bilirim.

# İÇİNDEKİLER

Sayfa

# İÇİNDEKİLER (devam)

Sayfa

# İÇİNDEKİLER (devam)

Sayfa

# ŞEKİLLER DİZİNİ

# ŞEKİLLER DİZİNİ (devam)

# ŞEKİLLER DİZİNİ (devam)

# ÇİZELGELER DİZİNİ

Çizelge Sayfa

# SİMGELER VE KISALTMALAR DİZİNİ

| Simgeler | Açıklama |
|---|---|
| $\oplus$ | Xor (Exclusive OR) |
| $\xleftarrow{R}$ | *Rastlantısal seçme işlemi* |
| $\mathcal{P}_n$ | *n-bit üzerine $(2^n)!$ permütasyon uzayının tamamı* |
| $\mathcal{B}_{k,n}$ | *n-bit blok, k-bit anahtar uzunluğunda blok şifreleme uzayı* |
| $\pi$ | *Rastsantısal permütasyon* |
| $Pr[A_1; A_2... : E]$ | *E olayının $A_1$, $A_2$, .. eylemlerinden sonra oluşma ihtimali* |
| $\Delta X$ | *Xa ve Xb bloklarının xor işlemine tabi tutulması* |

Kısaltmalar

| | |
|---|---|
| VHDL | Çok hızlı tümleşik devreler için donanım tanımlama dili |
| FGPA | Alanda programlanabilir kapı dizileri |
| RFC | Yorumlar için talep |
| XEX | Xor - Şifreleme - Xor |
| DES | Simetrik Şifreleme Standardı |
| AES | Gelişmiş Şifreleme Standardı |
| ECB | Elektronik Kod Defteri Modu |
| CBC | Şifreli Blok Zincirleme Modu |
| OFB | Çıktı Geri Beslemeli Modu |
| CFB | Şifre Geri Beslemeli Modu |
| CTR | Sayıcı Şifreleme Modu |

# BİRİNCİ BÖLÜM

## 1.GİRİŞ

Güvenlik, bilgi teknolojilerinde her zaman temel konulardan biri olmuştur ve şifreleme algoritmaları bilgi teknolojilerinde güvenliği sağlamanın en önemli araçlarından biridir. Şifreleme algoritmasının saldırılara karşı sağlamış olduğu direnç algoritmanın güvenliğini, algoritmanın şifreleme/çözme için geçen zamanı da performansını gösterir.

Teknoloji hızlı bir şekilde gelişmesi ile birlikte veri güvenliğinin hızlı ve etkili bir şekilde sağlanması giderek önem kazanmaktadır. Günümüz teknolojisinde ürünlerin güvenli olması yanında hızlı olması da beklenir. Bilgi teknolojilerinde ürünlerin hızlı ve güvenli olmasını sağlayan birden çok etken vardır. Bilgi güvenliğinde kullanılan algoritmaların hızlı ve güvenilir olması, algoritmada kullanılan anahtar uzunluğundan, algoritmada şifreleme için kullanılan çevrim sayısı gibi birçok algoritma özelliklerine göre değişmektedir.

AES şifreleme algoritması günümüzde hala güvenirliğini korumakta ve bilişim dünyasında güvenlik için kullanılmaktadır. Dolayısıyla AES şifreleme algoritmasının, performansının da hızlı olması gerekmektedir. Mevcut AES şifreleme algoritması 128-bit girdi bloğu, 128-bit, 192-bit ve 256-bit anahtar uzunluklarından biri ile şifreleme/çözme işlemi yapabilmektedir. Kullanılan anahtar uzunluklarına bağlı olarak şifreleme/çözme işlemi için kullanılan çevrim sayıları değişmektedir. Bu çevrimsel işlemin artması ile veri daha güvenli hale gelir. Fakat bu çevrimsel işlemlerin artmasıyla işlem sayısı ve bellek alanı artar. Bu performansı ve güvenliği doğrudan etkiler.

AESX ve önerilen model AESX$^{+}$ şifreleme algoritmaları, AES-128 şifreleme algoritması öncesinde ve sonrasında, 128-bit girdi ve çıktı bloklarını AES-128 algoritmasının anahtarından farklı bir 128-bitlik anahtar ile xor (Exclusive OR) işlemine tabi tutar. AESX ve önerilen model AESX$^{+}$ şifreleme algoritmalarında AES-128 şifreleme algoritmasında olduğu gibi 128-bitlik anahtar kullanıldığı için çevrim sayısı AES-256 şifreleme algoritmasına göre daha azdır. Dolayısıyla AESX ve önerilen model AESX$^{+}$ şifreleme algoritmalarının, AES-256 şifreleme algoritmasına göre daha hızlı

olması beklenir.

AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının AES-128 ve AES-256 ile performans karşılaştırmaları açık kaynak kripto kütüphaneleri kullanılarak yapılmıştır. Güvenlik analizleri ise anahtar- arama (kaba kuvvet) saldırısı ve ilişkili-anahtar saldırılarına karşı dirençleri analiz edilerek yapılmıştır.

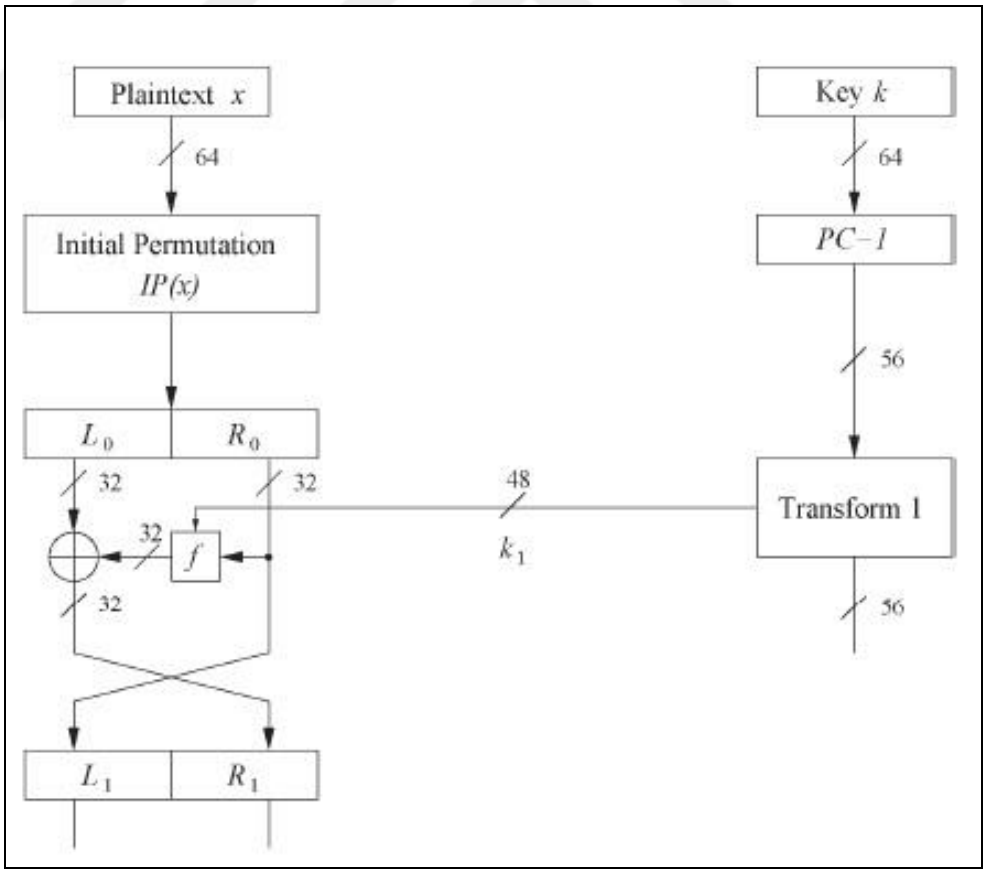
Bu çalışmada AESX ve önerilen model AESX$^+$ ile AES-128 ve AES-256 şifreleme algoritmaları somut kriterlere dayalı olarak karşılaştırılarak AESX$^+$ ve AESX şifreleme algoritmalarının, AES-128 ve AES-256 şifreleme algoritmalarına göre hız ve güvenlik açısından nerede olduğunun belirtilmesi hedeflenmektedir.

# İKİNCİ BÖLÜM

## 2.ÖNCEKİ ÇALIŞMALAR

DES şifreleme algoritması, 1970'li yıllarda geliştirilmiş olup, 1997'de resmileşmiş olan simetrik blok şifreleme algoritmasıdır. DES şifreleme algoritması 64-bit uzunluğunda blokları şifrelemektedir. 2000 yılında AES yerini almadan öncesinde kullanılmış olan en popüler blok şifreleme algoritmasıdır.

DES şifreleme algoritması şifreleme/çözme işlemini 16 turda tamamlamaktadır. Bu turlarda kullanılmak üzere 56-bitlik bir anahtar alır. Her bir turda 64-bitlik girdi bloğunu 32-bitlik iki parçaya böler ve bu parçalar yer değiştirir. Her bir yer değiştirmeden önce sağdaki parça tur anahtarı ile birlikte bir f fonksiyonuna girer ve soldaki parça ile xor işlemine tabi tutulur (Şekil 2.1). Bu işlem 16 tur boyunca devam ederek son olarak çıktı bloğunda şifreli veri elde edilir.



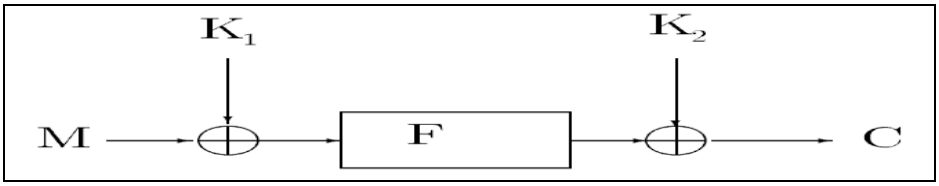

Şekil 2.1 Des şifreleme algoritmasının bir turu (Paar and P, 2010)

DES şifreleme algoritmasının 56-bitlik anahtar uzayının küçüklüğü algoritmanın duyurulduğu andan itibaren endişe verici olmuştur. Teknolojinin ilerlemesiyle birlikte 56-bitlik anahtarın yetersiz olduğu Wiener'ın (1994) tek bir düz metin, şifreli metin çiftinden anahtarı 4 saatten daha kısa bir sürede bulabilecek bir makine tahmininde bulunmasıyla ortaya çıkmıştır. 2000'li yılların başında DES şifreleme algoritmasının kırılmasıyla DES şifreleme algoritmasının güçlendirilmesi arayışları başlamıştır.

DES şifreleme algoritmasını güçlendirmek için üçlü DES (triple DES; DES-EDE3) şifreleme algoritması ortaya çıkarılmış ve DES-$EDE3_{k1.k2.k3}=DES_{k3}(DES_{k2}^{-1}(DES_{k1}(x)))$ şeklinde ifade edilmiştir. DES-EDE3 şifreleme algoritması 56 x 3 = 168-bitlik anahtar kullanmakta olup 64-bitlik girdi bloklarını şifreleyebilmektedir. DES-EDE3 şifreleme algoritmasının kullanılmasının bir nedeni üç anahtarı tek anahtara eşitlediğimizde DES şifreleme algoritması ile uygun bir şekilde çalışabilmesidir. Diğer bir nedeni ise ortadaki adam saldırılarına (meet in the middle attacks) karşı ikili DES şifreleme algoritmasına göre daha dirençli olmasıdır. Fakat DES-EDE3 şifreleme algoritması, DES şifreleme algoritmasına göre neredeyse üç kat daha yavaştır. Bu sebeple anahtar beyazlatma yöntemi ile daha dirençli ve daha az maliyetli bir DES türevi olabileceği ortaya çıkmıştır.

Blok şifrelemede, girdi bloğunun ve çıktı bloğunun xor işlemi ile güvenliğinin artırılmasına anahtar beyazlatma (key-whitening) adı verilir (Şekil 2.2). Böyle bir blok şifreleme yapısı ilk olarak Even and Mansour (1992) tarafından, $PX_{k1k2}(x) = k2 \oplus P(k1 \oplus x)$ şeklinde ifade edilmek üzere, yapılmıştır.

DES şifreleme algoritmasının anahtar beyazlatma tekniği ile yapılmış olan, DESX şifreleme algoritması Ron Rivest tarafından 1984 yılında icat edilmiştir. DESX, RSA Bilgi Güvenliği şirketinin ürünleri arasında uygulanmış ve bu ürünlerin dökümantasyonları arasında tanımlanmıştır. (Rogaway, 1996).




Şekil 2.2 Anahtar beyazlatma(Even and Mansour, 1992)

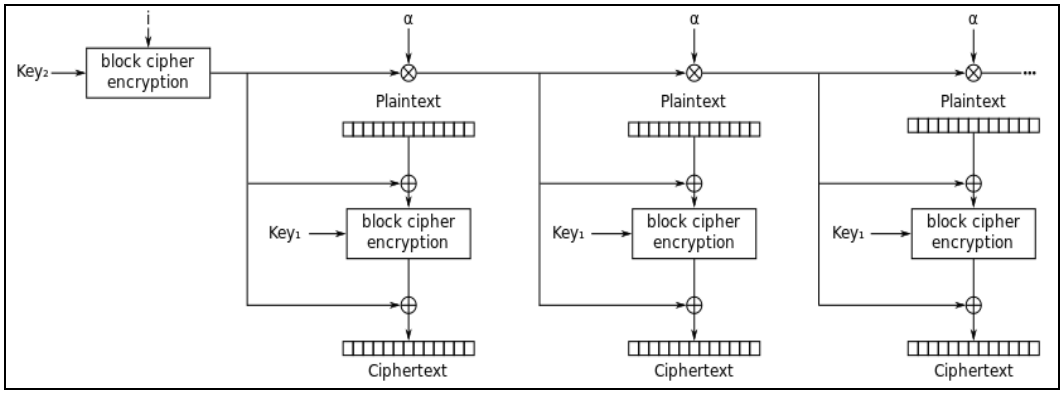
DESX şifreleme algoritması, $DESX_{k.k1.k2}(x) = k2 \oplus DES_k(k1 \oplus x)$ şeklinde tanımlanır. Burada $\oplus$ xor işlemini, k DES algoritmasında kullanılan anahtarı, k1 DES şifreleme öncesinde xor işlemi yapılan girdi bloğu için kullanılan anahtarı, k2 DES şifreleme sonrasında xor işlemi yapılan çıktı bloğu için kullanılan anahtarı, x ise algoritmaya girecek olan düz veriyi simgelemektedir. Görüldüğü üzere DESX şifreleme algoritması, DES şifreleme algoritmasına anahtar beyazlatma tekniği uygulanmış halidir. DES algortimasında kullanılan 56-bitlik anahtara ilave olarak iki adet 64-bitlik anahtar kullanılmaktadır. Böylelikle DESX şifreleme algoritmasında kullanılan anahtar uzunluğu 184 bit olmaktadır.

DESX şifreleme algoritması ile DES güvenliği anahtar arama (kaba kuvvet) saldırılarına karşı eklenen anahtar uzunluğu kadar daha dirençli hale geldiği ispatlanmıştır (Rogaway, 1996). Rogaway (1996) DES şifreleme algoritmasını bir kara kutu gibi düşünerek, anahtar arama saldırısı için saldırganın DES şifreleme algoritmasının iç özelliklerini kullanamayacağı bir saldırı modeli tasarlamıştır. Böylelikle DESX şifreleme algoritmasını genelleme yaparak bu saldırı modelinin tüm blok şifreleme algoritmalarında kullanılabileceğini göstermiştir. Killian ve Rogaway (1996), bu saldırının m adet bilinen düz veri ve $2^{118-\lg m}$ DES şifrelemeye ihtiyaç duyduğunu göstermişlerdir. Örneğin saldırganın elinde $2^{30}$ adet bilinen düz veri varsa, saldırıyı tamamlamak için, $2^{88}$ DES şifrelemesine ihtiyacı vardır.

DESX şifreleme algoritmasına karşı birçok saldırı türleri duyurulmuştur. Daemen (1992), $2^{32}$ seçilmiş düz veri ve $2^{88}$ DES şifrelemesine veya 2 bilinen düz veri ve $2^{120}$ DES şifrelemesine ihtiyaç duyan bir saldırı bildirmiştir. Biryukov ve Wagner $2^{32,5}$ bilinen düz veri, $2^{32,5}$ hafıza ve $2^{87,5}$ DES şifrelemesine ihtiyaç duyan saldırıyı bildirmişlerdir. Bu saldırılarda anahtar ilişkisi kurulmamıştır. Kelsey, Schneier ve Wagner (1997) DESX şifreleme algoritmasına $2^7$ adet ilişkili anahtar ve bilinen düz veri çifti ile $2^{56}$ DES şifrelemesine, Phan ve Schamir (2008) ise, $2^{3,5}$ adet ilişkili anahtar ve bilinen düz veri çifti ile $2^{56}$ DES şifrelemesine ihtiyaç duyan ilişkili anahtar saldırılarını duyurmuşlardır.

AES şifreleme algoritmasının anahtar uzunluklarının günümüz teknolojisine göre yeterli olması ve AES şifreleme algoritmasında bulunan anahtar ekleme katmanından dolayı, AES blok şifreleme algoritmasına uygulanmış olan anahtar beyazlatma tekniği pek bulunmamaktadır. AES

şifreleme algoritmasının disk şifrelemede kullanılan xor – şifreleme – xor (XEX) işlemi temelli bir modeli kullanılmaktadır (Şekil 2.3).




Şekil 2.3 XEX mod şifreleme (Wikipedia.org, 2017)

AES şifreleme algoritmasının, RFC 3566 olarak kabul edilmiş AES-XCBC-MAC-96 algoritması bulunmaktadır (Frankel and Herbert, 2003). Bu algoritmada anahtar beyazlatma tekniği uygulanmaktadır. AES şifreleme algoritmasının, AESX şifreleme algoritması da olmak üzere, çeşitli türleri VHDL dili ile FGPA içinde denemeleri yapılmıştır (Arrag S. et al., 2012).

# ÜÇÜNCÜ BÖLÜM

## 3.AES, AESX VE AESX$^{+}$

Yazının ilk icadından günümüze kadar gizli mesajları saklama yöntemleri var olmuştur. İlk şifreleme örnekleri eski Mısır'da M.Ö. 2000'li yıllara dayanmaktadır. Antik kriptografide harflerin değiştirilmesi ile yapılan Sezar Şifreleme gibi şifreleme yöntemleri daha çok popülerdir.

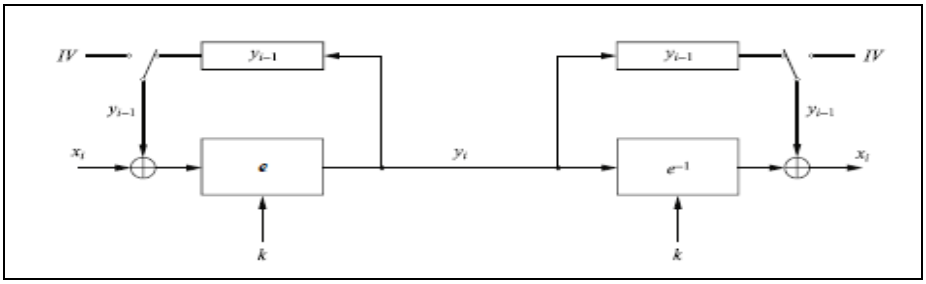
1976 yılında Diffie, Hellman ve Merkle'nin asimetrik kriptografiyi duyurmasına kadar kullanılan tüm şifreleme yöntemleri simetrik şifrelemedir. Simetrik şifrelemede bir anahtar bulunur ve aynı anahtar hem şifreleme işleminde hem de çözme işleminde kullanılır. Dolayısıyla simetrik şifrelemede hem şifreleyende hem de çözücü de aynı anahtar olması gereklidir. Modern kriptografide simetrik şifreleme algoritmalarının en önemli özelliği çok hızlı olmalarıdır. Simetrik şifreleme, asimetrik şifrelemeye göre çok daha hızlıdır. Bu sebeple günümüzde anahtar değişim işlemleri asimetrik şifreleme ile yapılırken şifreleme işlemleri simetrik şifreleme ile yapılmaktadır.

Simetrik şifreleme, blok şifreleme ve akım şifreleme olmak üzere ikiye ayrılmaktadır. Blok şifreleme veriyi bloklar halinde alıp şifreleme yapmak için, akım şifreleme ise veriyi birim (genellikle bitler halinde) olarak alıp şifreleme yapmak için kullanılmaktadır. Akım şifreleme cep telefonları gibi kaynakların kısıtlı olduğu ortamlarda kullanılmaktadır.

Simetrik şifreleme türlerinden blok şifreleme modern kriptografide kullanımı yaygın olan bir şifreleme türüdür. Blok şifrelemede şifrelenecek veri bloklar halinde şifrelenir ve çözme işleminde de bloklar halinde çözüldükten sonra birleştirilerek veri elde edilir. Ayrıca şifreleme işlemi yapılırken kullanılan bloklar arasındaki ilişkiye göre blok şifreleme modları vardır.

ECB (Electronic Code Book Mode; *Elektronik Kod Defteri Modu*) her bloğun birbirinden bağımsız olarak ayrı ayrı şifrelendiği moddur. Bloklar arasında bir ilişki olmadığı için uygulamada hızlıdır. Fakat belirli bir girdi bloğu için hep aynı sonucu üretmektedir.

CBC (Cipher Block Chaining Mode; *Şifreli Blok Zincirleme Modu*) şifreli verinin kendisinden önceki tüm bloklara dayalı olarak şifrelenmesi esasına dayalı olarak çalışır. Şifrelemede başlangıç vektörü sayesinde rastsallık sağlanır (Şekil 3.1).

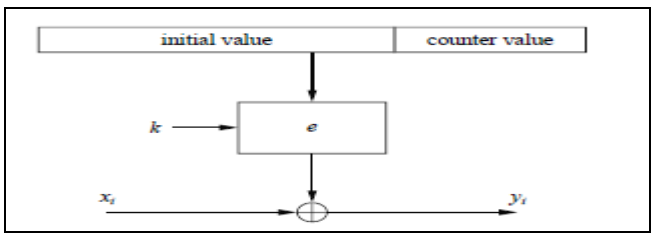



Şekil 3.1 CBC modu şifreleme/çözme (Paar and Pelzl, 2010)

OFB (Output Feedback Mode; *Çıktı Geri Beslemeli Modu*) blok şifrelemeden senkronize akım şifreleme elde etmek için kullanılmaktadır. Şifreli çıktı anahtar akım bitleri olarak kullanılarak şifrelenecek veri ile xor işlemine tabi tutulur.

CFB (Cipher Feedback Mode; *Şifre Geri Beslemeli Modu*) blok şifrelemeyi asenkronize akım şifreleme elde etmek için kullanır. OFB modundan farklı olarak xor işleminden sonra çıktı bitleri tekrar anahtar akım bitleri üretmek üzere kullanılmaktadır.

CTR (Counter Mode; *Sayıcı Şifreleme Modu*) OFB ve CFB modları gibi blok şifrelemeyi akım şifreleme gibi kullanır. Bloğa girdi olarak bir sayaç tarafından her defasında farklı değer gönderilerek yeni bir anahtar akım bitleri elde edilir (Şekil 3.2).




Şekil 3.2 CTR modu şifreleme/çözme (Paar and Pelzl, 2010)

Çalışmanın bu bölümünde popüler blok şifreleme algoritmalarından olan AES, AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının yapısı

ve özellikleri incelenecektir.

## 3.1 AES

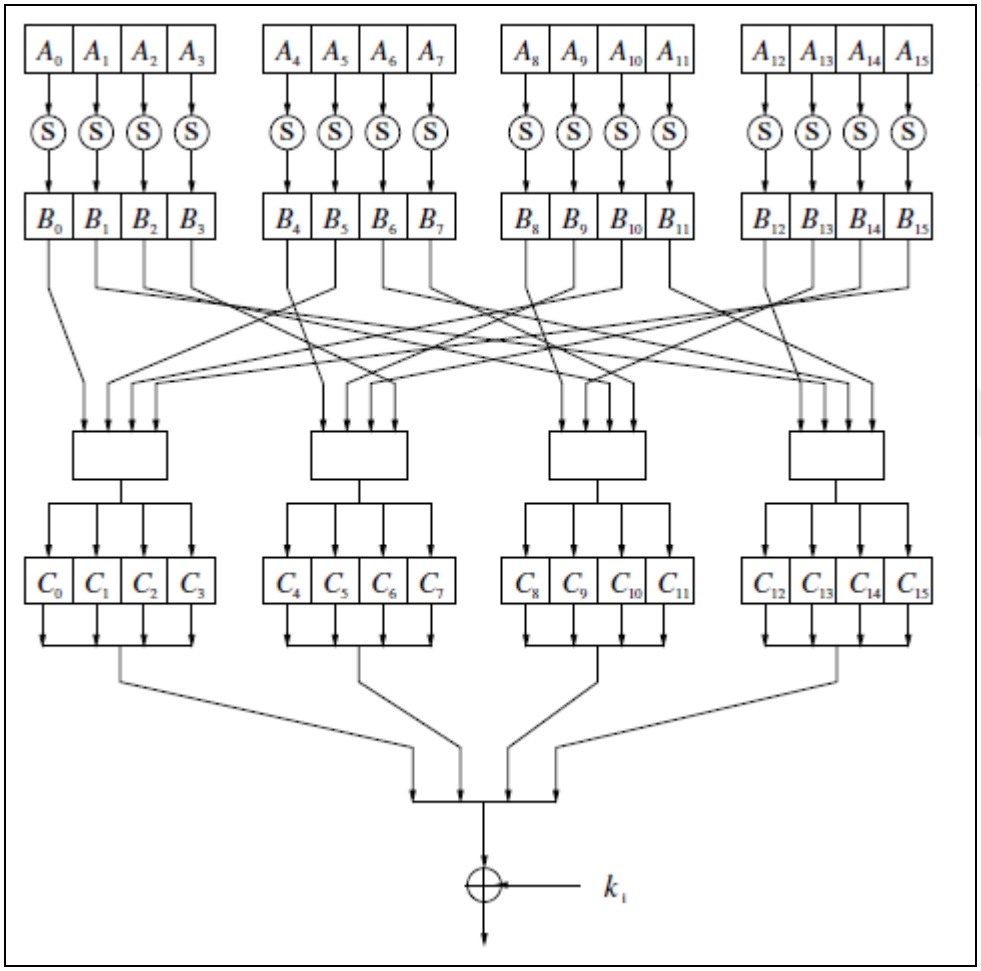
DES şifreleme algoritmasının bazı uygulamalarda yeteri kadar güvenlik sağlamadığı anlaşılınca, NIST Ocak 1997'de AES adında yeni bir blok şifreleme önerisinde bulunmuştur. Belirtilen öneride istenilen blok şifreleme algoritmasının; 128-bitlik blokları şifreleme/çözme işlemlerini yapabilmesi, 128-bit, 192-bit ve 256-bit anahtar uzunluklarını desteklemesi, yazılım ve donanımda etkili olabilmesi gibi yetenekleri olması istenmiştir. Yarışmaya dünya çapında 12 farklı ülke katılım göstermiştir. Ağustos 1998'de 15 aday şifreleme algoritmasının kabul edildiği, Ağustos 1999'da ise 5 finalist şifreleme algoritması açıklanmıştır. Bu algoritmalar IBM Şirketi tarafından hazırlanan Mars, RSA Laboratuvar tarafından hazırlanan RC6, Joan Daemen ve Vincent Rijmen tarafından hazırlanan Rijndael, Ross Andersson, Eli Biham ve Lars Knudsen tarafından hazırlanan Serpent, Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall ve Niels Ferguson tarafından hazırlanan Twofish şifreleme algoritmalarıdır. NIST Ekim 2000'de Rijndael şifreleme algoritmasını AES olarak seçtiğini ilan etmiştir.

AES şifreleme algoritması olarak Rijndael şifreleme algoritması seçilmesine rağmen ikisi arasında çok az farklılık bulunmaktadır. Rijndael şifreleme algoritması 128, 192 ve 256 bit uzunluklarında anahtar ve blok uzunluklarını desteklemektedir. Ancak AES şifreleme algoritması sadece 128 bit uzunluğundaki bloklara işlem yapmaktadır. Dolayısıyla Rijndael şifreleme algoritmasının sadece 128 bit blokları işleyen kısmı AES şifreleme algoritması olarak bilinmektedir. AES şifreleme algoritmasında şifreleme tur sayısı anahtar uzunluğuna göre değişmektedir. AES şifreleme algoritması 128 bit anahtar uzunluğunda 10 turda, 192 bit anahtar uzunluğunda 12 turda ve 256 bit anahtar uzunluğunda 14 turda şifreleme işlemini gerçekleştirmektedir.

AES şifreleme algoritmasında DES şifreleme algoritmasında bulunan Fiestel yapısı yoktur. Fiestel yapısında, şifrelenecek blok önce iki parçaya ayrılır ve sonra şifreleme işlemi yapılır. AES şifreleme algoritmasında ise şifrelenecek olan 128 bit bloğun tamamı tek bir turda şifrelenir. Bundan dolayı AES şifreleme algoritmasında, Fiestel yapısı kullanan şifreleme algoritmalarına göre daha az tur sayısı bulunmaktadır.

### 3.2 AES Şifreleme Yapısı

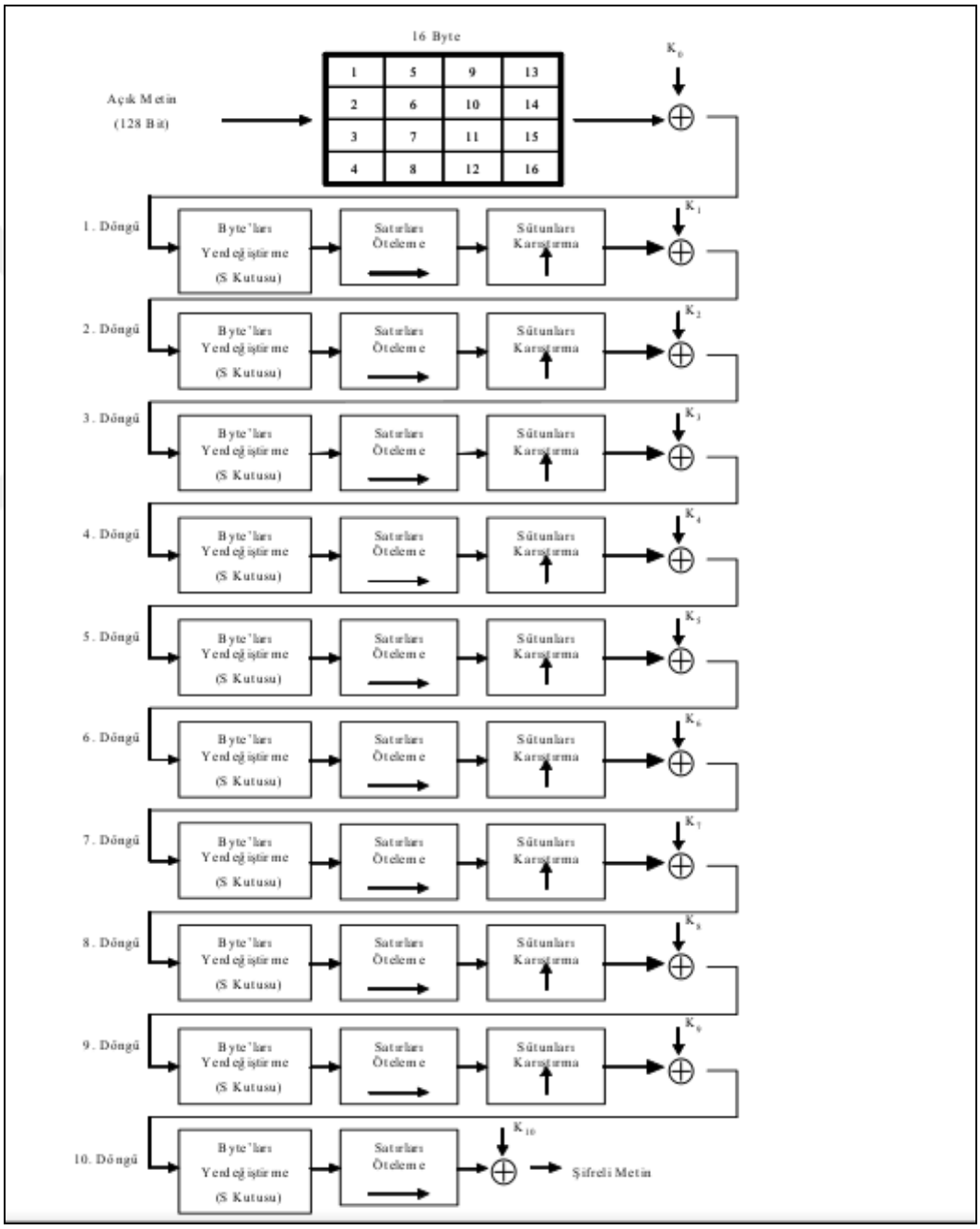
AES şifreleme algoritması bayt uyumlu şifreleme algoritmasıdır. AES şifreleme algoritmasında her bir tur dört işlemden geçer. Şekil 3.3 AES şifreleme algoritmasında bir turda yapılan işlemleri göstermektedir. İlk olarak 128 bitlik blok 4×4 bayt matrise dönüştürülür. Burada elde edilen 16 bayt girdi olan $A_0,\ldots,A_{15}$ değerleri S kutularından geçirilir. S kutularından çıkan 16 bayt $B_0,\ldots,B_{15}$ değerleri bayt olarak yer değiştirilerek, sütun karıştırma işleminden geçirilir. Burada elde edilen 16 bayt $C_0,\ldots,C_{15}$ değerleri o tur için üretilmiş olan anahtar ile xor işlemine tabi tutulur.




Şekil 3.3 AES şifreleme bir tur fonksiyonu (Paar and Pelzl, 2010)

Şekil 3.3'de görüldüğü üzere AES şifreleme algoritmasında işlemler matris üzerindeki satır ve sütunlarda yapılmaktadır. Anahtar baytları da anahtar uzunluğuna bağlı olarak 4 satır ve 128 bit anahtarda 4 sütun, 192 bit anahtarda 6 sütun ve 256 bit anahtarda 8 sütun matris olarak ayarlanmıştır.

Şekil 3.4'de 128 bit anahtar kullanılan tüm AES şifreleme algoritması gösterilmektedir. AES şifreleme algoritmasına gelen 128 bitlik blok ilk tur öncesinde 4×4 bayt matrise dönüştürülür ve $K_0$ anahtarı ile xor işlemine tabi tutulur. Her bir turda sırasıyla baytların yer değiştirmesi (Byte Substitution), satırların ötelenmesi (Shift Rows), sütunların karıştırılması (Mix Column) ve tur anahtarı ile xor (Add Round Key) işlemlerine tabi tutulur. Her bir turda bu işlemler tekrarlayarak devam eder. Ancak son turda sütunların karıştırılması yapılmamaktadır.




Şekil 3.4 Tüm AES algoritması (128 bit anahtar için) (Sakallı, 2006)

### 3.2.1 Bayt Değiştirme Katmanı

Bayt değiştirme katmanı Şekil 3.3'de görülen 16 adet paralel S kutularının bulunduğu katmandır. S kutularının tamamı aynıdır ve her birinin 8 bitlik girdi değerine karşılık 8 bitlik çıktı değeri bulunmaktadır. S kutuları AES şifreleme algoritması için doğrusal olmayan tek elemandır. Yani S(Ai) + S(Aj) ≠ S(Ai + Aj) eşitsizliği bulunmaktadır. S kutuları her bir bayt için bağımsız olarak çalışır ve tersine çevrilebilir olarak tasarlanmıştır. AES şifreleme algoritmasının S kutularının yer değiştirme tablosu Şekil 3.5'de görülmektedir.

| | | y | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| x | 0 | 63 | 7C | 77 | 7B | F2 | 6B | 6F | C5 | 30 | 01 | 67 | 2B | FE | D7 | AB | 76 |
| | 1 | CA | 82 | C9 | 7D | FA | 59 | 47 | F0 | AD | D4 | A2 | AF | 9C | A4 | 72 | C0 |
| | 2 | B7 | FD | 93 | 26 | 36 | 3F | F7 | CC | 34 | A5 | E5 | F1 | 71 | D8 | 31 | 15 |
| | 3 | 04 | C7 | 23 | C3 | 18 | 96 | 05 | 9A | 07 | 12 | 80 | E2 | EB | 27 | B2 | 75 |
| | 4 | 09 | 83 | 2C | 1A | 1B | 6E | 5A | A0 | 52 | 3B | D6 | B3 | 29 | E3 | 2F | 84 |
| | 5 | 53 | D1 | 00 | ED | 20 | FC | B1 | 5B | 6A | CB | BE | 39 | 4A | 4C | 58 | CF |
| | 6 | D0 | EF | AA | FB | 43 | 4D | 33 | 85 | 45 | F9 | 02 | 7F | 50 | 3C | 9F | A8 |
| | 7 | 51 | A3 | 40 | 8F | 92 | 9D | 38 | F5 | BC | B6 | DA | 21 | 10 | FF | F3 | D2 |
| | 8 | CD | 0C | 13 | EC | 5F | 97 | 44 | 17 | C4 | A7 | 7E | 3D | 64 | 5D | 19 | 73 |
| | 9 | 60 | 81 | 4F | DC | 22 | 2A | 90 | 88 | 46 | EE | B8 | 14 | DE | 5E | 0B | DB |
| | A | E0 | 32 | 3A | 0A | 49 | 06 | 24 | 5C | C2 | D3 | AC | 62 | 91 | 95 | E4 | 79 |
| | B | E7 | C8 | 37 | 6D | 8D | D5 | 4E | A9 | 6C | 56 | F4 | EA | 65 | 7A | AE | 08 |
| | C | BA | 78 | 25 | 2E | 1C | A6 | B4 | C6 | E8 | DD | 74 | 1F | 4B | BD | 8B | 8A |
| | D | 70 | 3E | B5 | 66 | 48 | 03 | F6 | 0E | 61 | 35 | 57 | B9 | 86 | C1 | 1D | 9E |
| | E | E1 | F8 | 98 | 11 | 69 | D9 | 8E | 94 | 9B | 1E | 87 | E9 | CE | 55 | 28 | DF |
| | F | 8C | A1 | 89 | 0D | BF | E6 | 42 | 68 | 41 | 99 | 2D | 0F | B0 | 54 | BB | 16 |

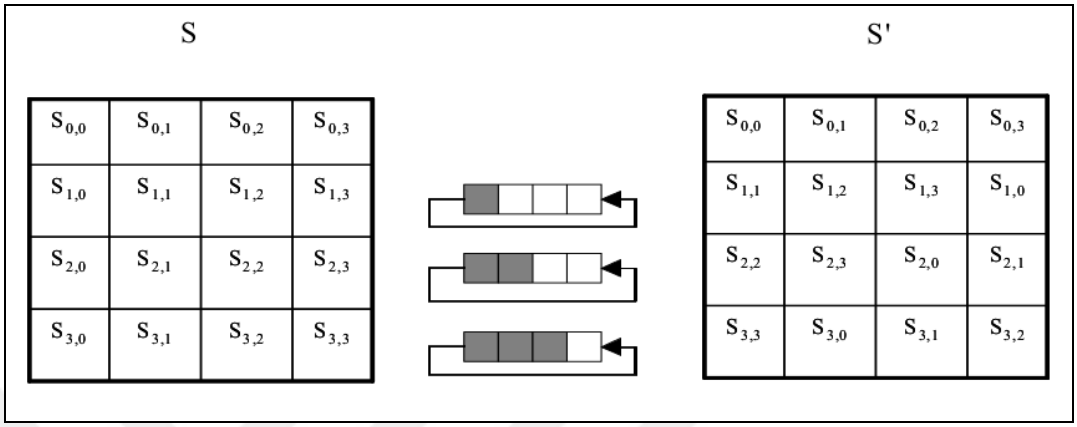
Şekil 3.5 AES S kutuları yer değiştirme tablosu(Paar and Pelzl, 2010)

S kutusuna girdi baytı olarak gelen $A_i$ değerinin A5 olduğunu varsayarsak, Şekil 3.6'daki tabloya göre değeri S(A5) = 06 olur. Aynı işlemi bitler şeklinde gösterecek olursak, S(1010 0101) = (0000 0110) olur.

### 3.2.2 Dağılma Katmanı

AES şifreleme algoritmasında dağılma katmanı satır kaydırma (ShiftRows) ve sütun karıştırma (MixColumn) işlemlerinden oluşur. Dağılma katmanının amacı girdi bitlerinin tamamının tüm durumlara yayılmasını sağlamaktır. Dağılma katmanı S kutularından farklı olarak doğrusaldır. Matriste bulunan A ve B değerleri için D(A) + D(B) = D(A + B) eşitliği elde edilmektedir.

Satır kaydırma işleminde, matriste bulunan ilk satır aynı kalacak şekilde, ikinci satır sola bir, üçüncü satır sola iki ve dördüncü satır sola üç kere kaydırılır (Şekil 3.6). Her satırda taşan bölümler satır başına eklenir. Satır kaydırma işleminin amacı AES şifreleme algoritmasının yayılma özelliğini artırmaktır.




Şekil 3.6 Satır kaydırma işlemi (Sakallı, 2006)

Sütun karıştırma işlemi, matriste bulunan her bir sütunu karıştırma yaparak doğrusal dönüşüm sağlar. Her bir girdi baytı, dört çıktı baytını etkilediği için sütun karıştırma işlemi AES şifreleme algoritmasında yayılma sağlayan en önemli işlemdir. Satır kaydırma ve sütun karıştırma işlemlerinin birlikte üç tur devam etmesi ile matriste bulunan tüm baytlar, girdi baytı olan 16 bayta dayalı olmaktadır.

Sütun karıştırma işleminde, her bir dört baytlık sütun vektör olarak kabul edilir ve sabit değerleri olan 4×4 bayt matrisle çarpılır (Şekil 3.7). Sabit matriste bulunan katsayılar ile yapılan tüm çarpma ve toplama işlemleri Galois alanında (GF $2^8$) yapılmaktadır.

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

Şekil 3.7 Sütun karıştırma matrisi (Paar and Pelzl, 2010)

Bir sonraki çıktı baytları olan $C_4$, $C_5$, $C_6$ ve $C_7$ için $B_4$, $B_9$, $B_{14}$, $B_3$ baytları aynı sabit matris ile çarpılarak ve böyle devam ederek tüm $C_0 \ldots C_{15}$ çıktı baytları elde edilir.

Sabit matriste bulunan onaltılık değer olan 01 (0000 0001), GF ($2^8$) polinom değeri 1, onaltılık değer olan 02 (0000 0010), GF ($2^8$) polinom değeri x ve onaltılık değer olan 03 (0000 0011), GF ($2^8$) polinom değeri x + 1 olarak gösterilir.

Galois alanında vektör matris çarpma işlemi, ilgili baytların xor işlemine tabi tutulması ile yapılabilmektedir. Çarpma işleminde 01 etkisiz olduğu için herhangi bir işlem yapılma ihtiyacı yoktur. Çarpma işleminde 02 ve 03 değerleri için x ve x + 1 polinom değerleri ile çarpma yapılır ve sonuç bayt değeri, indirgenemez polinom $P(x) = x^8 + x^4 + x^3 + x + 1$ değerine göre indirgenmektedir. Örnek verecek olursak onaltılık 02 (0000 0010) değerini GF ($2^8$) de onaltılık 87 (1000 0111) değeri ile çarpalım. İşlemin sonucu $x \cdot (x^7 + x^2 + x + 1) = x^8 + x^3 + x^2 + x$ çıkar. Sonuçta polinom derecesi 8 çıktığı için indirgenemez polinom $P(x) = x^8 + x^4 + x^3 + x + 1$ değeri ile xor işlemi (mod alma) yapılır. Eğer polinom derecesi 8'den küçük olsaydı indirgenemez polinom ile işlem yapmaya gerek kalmayacaktı. Sonuç olarak $x^8 + x^3 + x^2 + x$ (100001110) $\oplus$ $x^8 + x^4 + x^3 + x + 1$ (100011011) = 0001 0101 onaltılık 15 değeri elde edilir.

Galois alanında vektör matris çarpma işlemine göre Şekil 3.7'deki matrisi çarptığımızda aşağıda gösterildiği şekilde matris çarpımı elde edilir.

$$C_0 = (\{02\} \cdot B_0) \oplus (\{03\} \cdot B_5) \oplus B_{10} \oplus B_{15}$$

$$C_1 = B_0 \oplus (\{02\} \cdot B_5) \oplus (\{03\} \cdot B_{10}) \oplus B_{15}$$

$$C_2 = B_0 \oplus B_5 \oplus (\{02\} \cdot B_{10}) \oplus (\{03\} \cdot B_{15})$$

$$C_3 = (\{03\} \cdot B_0) \oplus B_5 \oplus B_{10} \oplus (\{03\} \cdot B_{15})$$

### 3.2.3 Anahtar Ekleme Katmanı

AES şifreleme algoritmasında anahtar ekleme katmanında 16 bayt matris ile 16 bayt tur anahtarı xor işlemine tabi tutulur (Bkz. Şekil 3.3).

Yapılan xor işlemi Galois alanında GF (2) ekleme işlemine eşittir.

### 3.2.4 Anahtar Üretme

AES şifreleme algoritmasında tur sayısı anahtar uzunluğuna göre belirlenir ve her bir turda kullanılacak olan alt anahtarlar bu anahtarlar kullanılarak üretilir. Anahtar, şifreleme işlemi öncesinde ve her bir tur sonunda kullanılmaktadır (Bkz. Şekil 3.4). Dolayısıyla AES şifreleme algoritmasında kullanılan tur sayısından bir fazla anahtara ihtiyaç bulunmaktadır (Çizelge 3.1).

Çizelge 3.1 AES alt anahtar sayıları

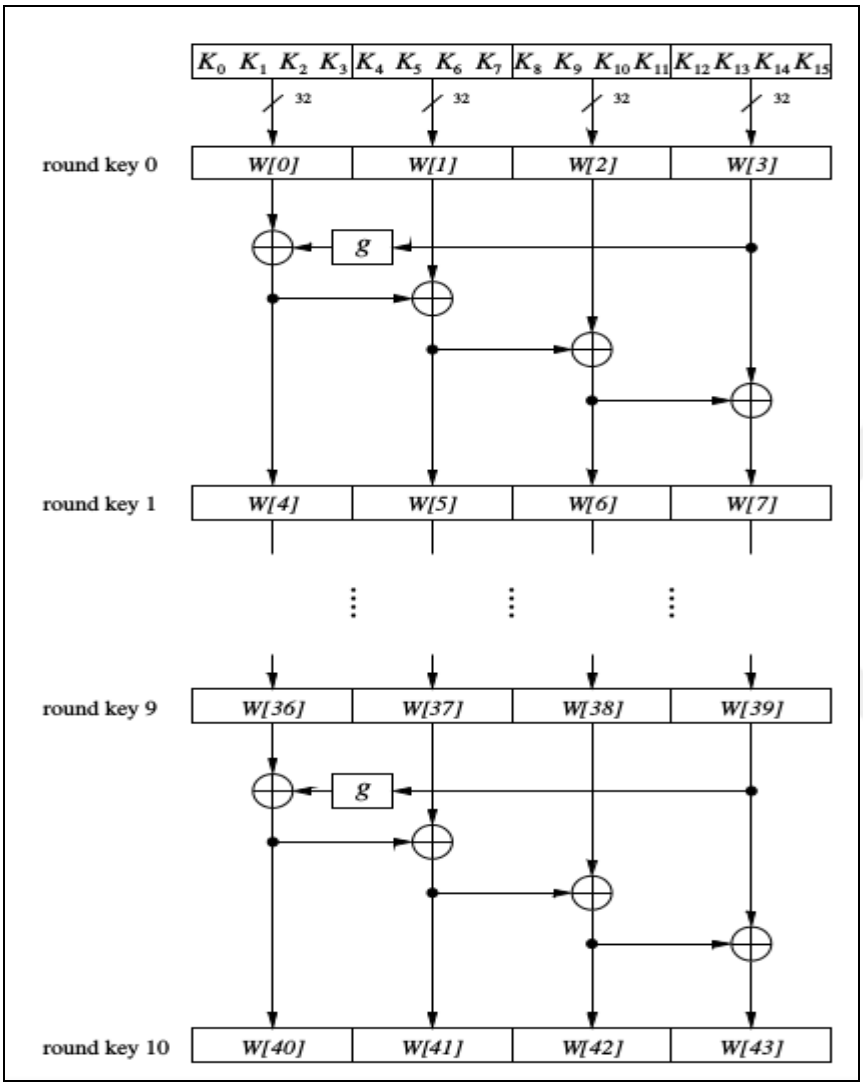
| Anahtar Uzunluğu (bit) | Alt anahtar sayısı |
|---|---|
| 128 | 11 |
| 192 | 13 |
| 256 | 15 |

AES şifreleme algoritmasında, başlangıç anahtarı hariç, alt anahtarların üretimi kendisinden önceki anahtara bağlıdır. Herhangi bir alt anahtar üretilmeden önce kendisinden önceki alt anahtar bilinmelidir. Anahtar üretim işlemi kelime (word) uyumludur ve bir kelime 32 bittir. Alt anahtarlar kelimeleri içeren W anahtar üretme dizinde tutulmaktadır. Anahtar uzunluklarına göre farklı anahtar üretme yöntemleri bulunmaktadır.

128 bit anahtar uzunluğuna göre alt anahtar üretme işlemi Şekil 3.8'de gösterilmektedir. Anahtar üretim dizisi 11 alt anahtarı W[0], … , W[43] elemanları ile birlikte tutmaktadır. Orijinal AES anahtarının baytları $K_0$, … , $K_{15}$ elemanları ile gösterilmektedir.

İlk tur alt anahtarı k0 orijinal AES anahtarıdır ve bu değer W dizisinin ilk dört elemanı içine kopyalanır. Dizinin diğer elemanlarının hesaplanması için; W[4i] = W[4(i - 1)] + g(W[4i - 1]) eşitliği kullanılır. W[4i] Şekil 3.8'de

görüldüğü üzere alt anahtarın en sol kısmıdır ve i değerleri i = 1, …, 10 şeklindedir. Alt anahtarın geriye kalan 3 kelimelik kısmı W[4i + j] = W[4i + j - 1] + W[4(i - 1) + j] formülü ile hesaplanmaktadır ve j = 1,2,3 değerleridir.




Şekil 3.8 128 bit anahtar uzunluğu için AES alt anahtar üretimi (Paar and Pelzl, 2010)

Anahtar üretmede kullanılan g() fonksiyonu doğrusal değildir ve 4 bayt girdi ve çıktısı bulunmaktadır (Şekil 3.9). g() fonksiyonu dört girdi baytını öncelikle yer değiştirir, bayt şeklinde S kutularından geçirir ve tur katsayısını (round coefficient, RC) ekler. Tur katsayısı Galois alanının GF(28) 8 bitlik bir elemanıdır. Şekil 3.9'da görüldüğü üzere g() fonksiyonunda sadece en soldaki bayt değerine eklenmektedir. Tur katsayısı her turda aşağıdaki şekilde değişmektedir;

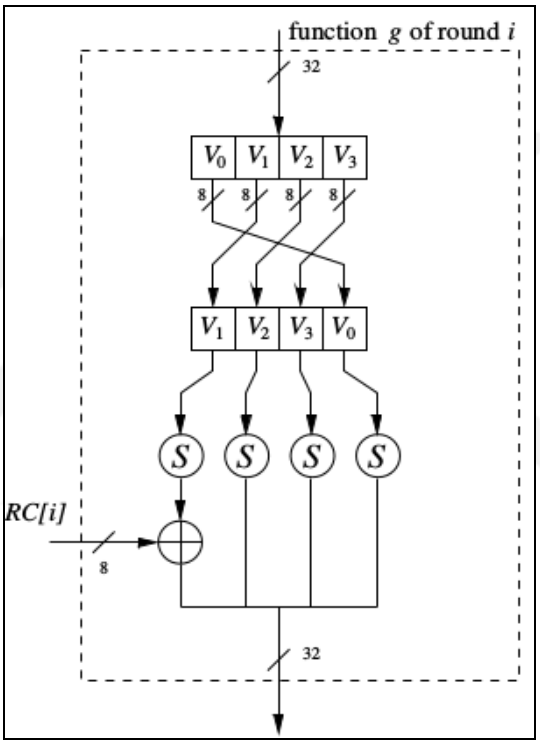
$RC[1] = x^0 = (0000\ 0001)_2$,

$RC[2] = x^1 = (0000\ 0010)_2$,

$RC[3] = x^2 = (0000\ 0100)_2$,

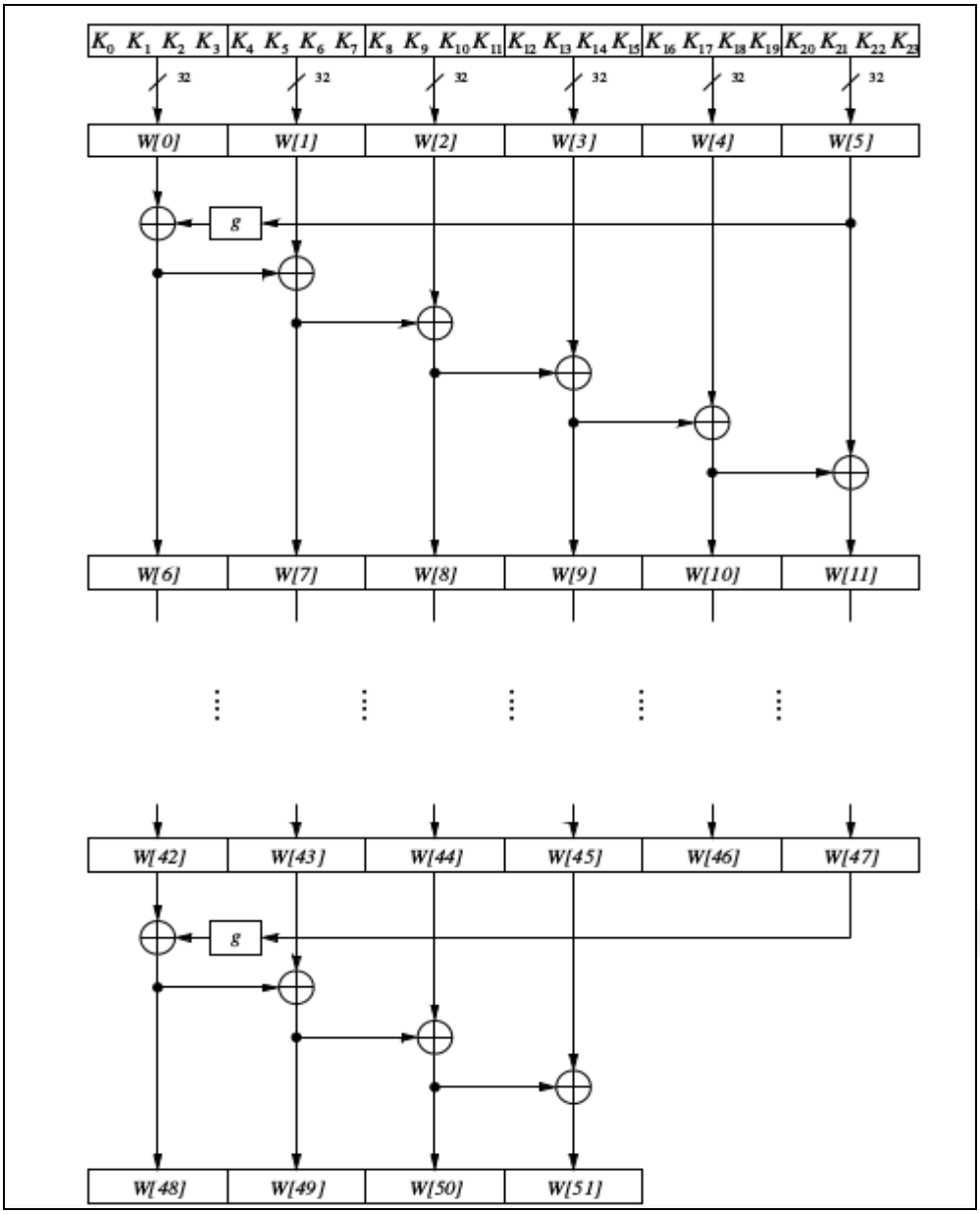
⋮

$RC[10] = x^9 = (0011\ 0110)_2$.



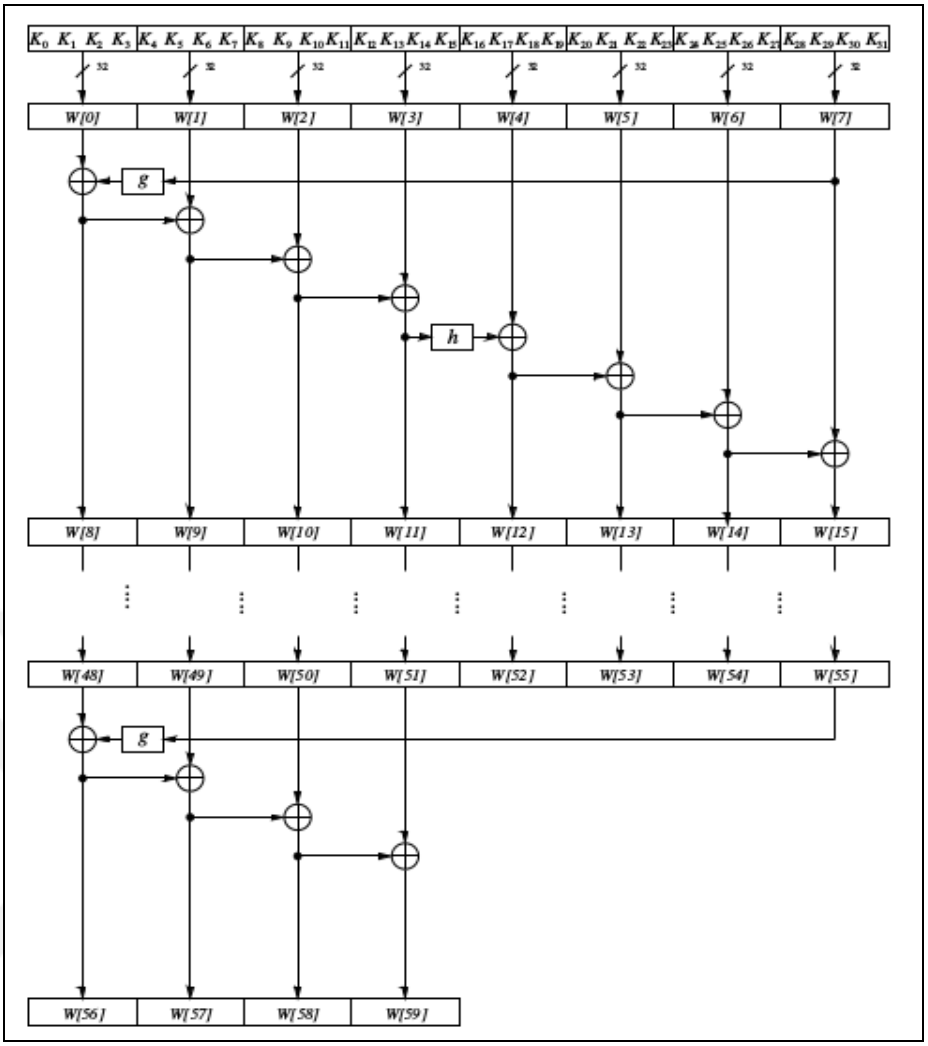

Şekil 3.9 g fonksiyonu (Paar and Pelzl, 2010)

AES şifreleme algoritması için 192 bit anahtar uzunluğunda 12 tur bulunmaktadır ve dolayısıyla her biri 128 bit uzunluğunda 13 adet alt anahtara ve W[0], …, W[51] dizi elemanları ile tutulmuş olan 52 adet kelimeye ihtiyaç bulunmaktadır. 192 bit anahtar için anahtar üretme işleminde sekiz adet tekrarlama bulunmaktadır (Şekil 3.10). Her bir tekrarlamada W dizisinde 6 adet kelime üretilir. AES turlarında kullanılan alt anahtarların elemanları W dizisinin sıralı dört elemanlarından oluşur. Örneğin ilk tur anahtarı dizinin W[0], W[1], W[2], W[3] elemanlarından, ikinci tur anahtarı W[4], W[5], W[6], W[7] elemanlarından oluşur. Sekiz adet tekrarlamada g() fonksiyonu için tur katsayısı RC[1], …, RC[8] hesaplamalarına ihtiyaç bulunmaktadır.



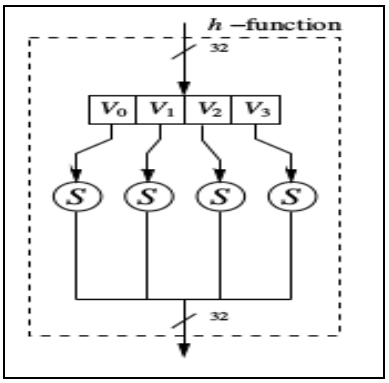

Şekil 3.10 192 bit anahtar uzunluğu için AES alt anahtar üretimi (Paar and Pelzl, 2010)

AES şifreleme algoritması için 256 bit anahtar uzunluğunda 14 tur bulunmaktadır ve dolayısıyla her biri 128 bit uzunluğunda 15 adet alt anahtara ve W[0], …, W[59] dizi elemanları ile tutulmuş olan 60 adet kelimeye ihtiyaç bulunmaktadır. 256 bit anahtar için anahtar üretme işleminde yedi adet tekrarlama bulunmaktadır ve her bir tekrarlamada alt anahtarlar için sekiz kelime hesaplanmaktadır (Şekil 3.11). AES turlarında kullanılan alt anahtarların elemanları W dizisinin sıralı dört elemanlarından oluşur (ilk tur için W[0], W[1], W[2], W[3]). Yedi adet tekrarlamada g() fonksiyonu için tur katsayısı RC[1], …, RC[7] hesaplamalarına ihtiyaç bulunmaktadır.




Şekil 3.11 256 bit anahtar uzunluğu için AES alt anahtar üretimi (Paar and Pelzl, 2010)

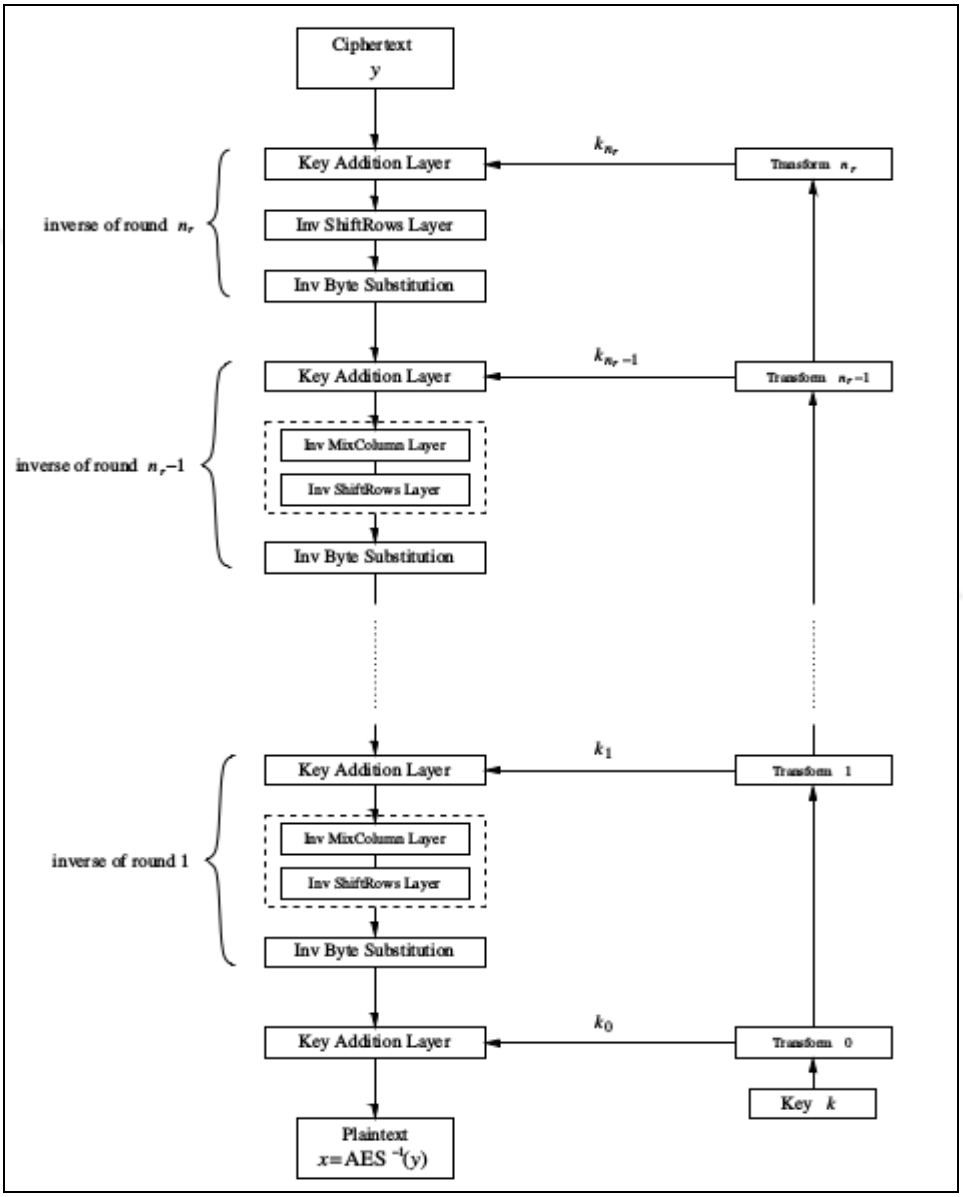
Bu anahtar üretiminde 4 bayt girdi ve çıktısı bulunan h() fonksiyonu bulunmaktadır (Şekil 3.12). Fonksiyon tüm girdi baytlarını s kutularından geçirmektedir.




Şekil 3.12 h fonksiyonu (Paar and Pelzl, 2010)

### 3.3 AES Çözme İşlemi

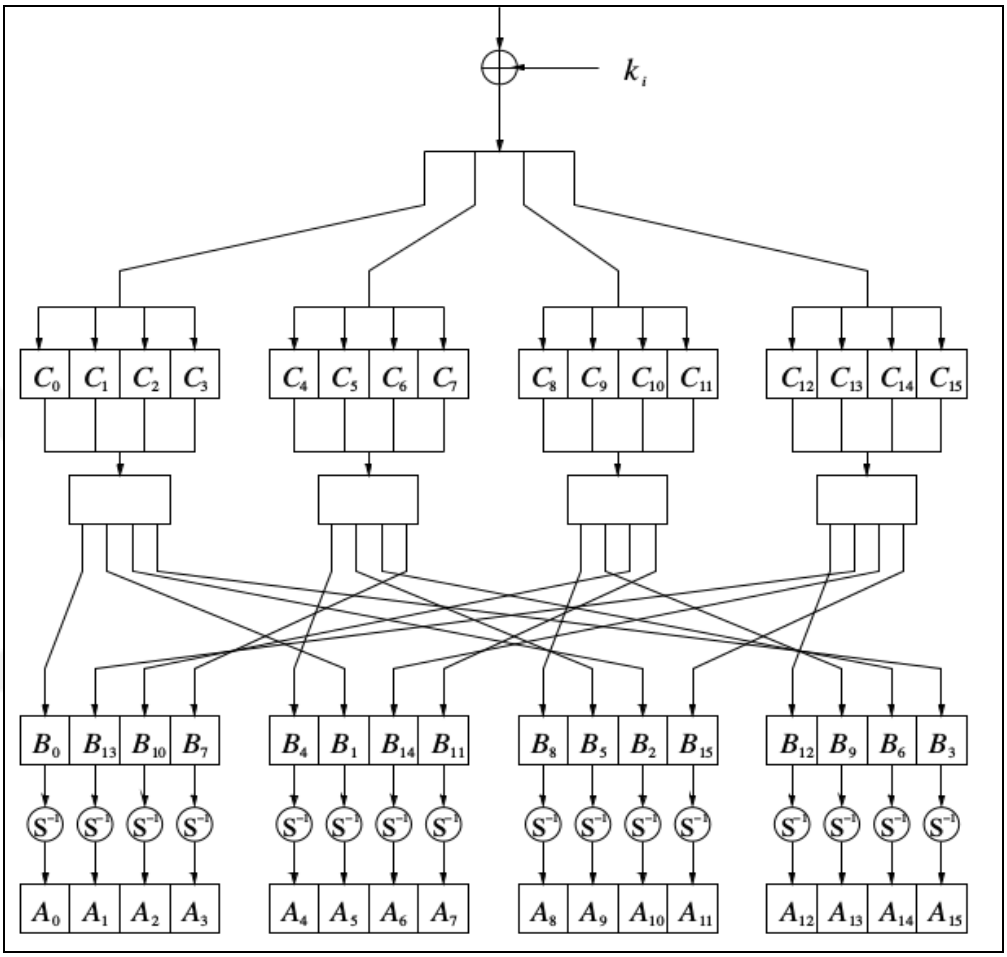
AES şifreleme algoritmasında şifre çözme işlemi için tüm katmanları ters sırada yapmak gereklidir (Şekil 3.13). Ayrıca katmanlarda kullanılan işlemler de tersleri ile değiştirilmelidir. Çözme işleminde kullanılacak olan alt anahtarları da ters çevirebilmek için, anahtar üretme işlemi tersten yapılmalıdır.




Şekil 3.13 AES şifre çözme blok diyagramı (Paar and Pelzl, 2010)

AES şifreleme algoritmasında son turda sütun karıştırma işlemi olmadığı için şifre çözme işleminin ilk turunda sütun karıştırma işleminin tersi bulunmamaktadır. Ancak diğer tüm çözme turlarında, şifreleme

işleminde kullanılan katmanların hepsi bulunmaktadır. Şekil 3.14'de genel bir AES şifre çözme turu gösterilmektedir. Xor işlemi, kendisinin tersi olduğu için şifreleme kullanılan anahtar ekleme işleminin aynısı, şifre çözme işleminde de kullanılmaktadır.



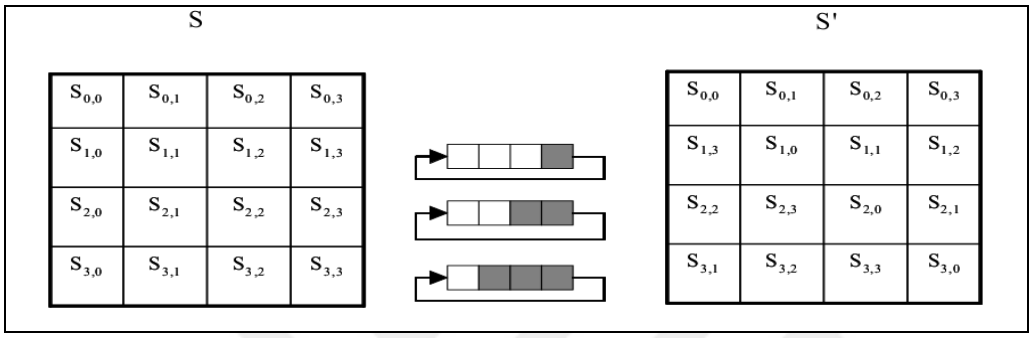

Şekil 3.14 AES çözme bir tur fonksiyonu (Paar and Pelzl, 2010)

Şifreli veri 4×4 bayt matris yapılıp, üzerine alt anahtar eklendikten sonra ters sütun karıştırma işlemi yapılmaktadır. Sütun karıştırma işlemini tersten yapabilmek için, sütun karıştırmada kullanılan matris ters çevrilmelidir. Şifreli verinin 4 baytlık sütunu ters matris ile çarpılmaktadır. Matris içinde sabitler bulunmaktadır ve çarpma ve ekleme işlemleri Galois alanında GF($2^8$) yapılmaktadır. Şifreli verinin her bir sütunu sabit matris ile çarpılarak yeni B0, …, B15 matrisi elde edilir.

Satırları kaydırma işlemini tersten yapabilmek için, durum matrisinde bulunan satırları, şifrelemede kullanılan yönün ters yönünde kaydırılmalıdır

(Şekil 3.15). Burada şifreleme işleminde olduğu gibi yine ilk satırda bir değişiklik olmamaktadır.

AES şifre çözme işleminde bayt değiştirme işlemini tersten yapabilmek için S kutularının tersi kullanılmalıdır. S kutularını ters yapabilmek için $Ai = S^{-1}(Bi) = S^{-1}(S(Ai))$ eşitliğini sağlamak gereklidir. Ters S kutularının şifrelemede olduğu gibi onaltılık değerler için hesaplama tablosu bulunmaktadır.




Şekil 3.15 Ters satır kaydırma işlemi (Sakallı, 2006)

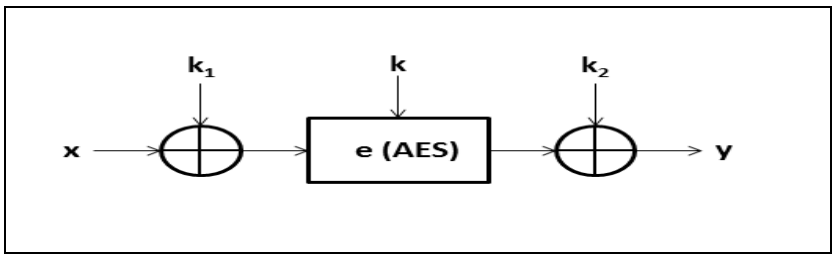
AES şifre çözme işleminde ilk turda son alt anahtara ihtiyaç duyulduğu için anahtar kullanımı Şekil 3.13'de gösterilen şekilde yapılmalıdır. AES şifre çözme işleminde tüm anahtar üretme işlemi önceden yapılmalı ve tur sayısına göre üretilmiş olan alt anahtarlar saklanmalıdır. Bu durum çözme işleminin, şifrelemeye göre küçük bir farkla gecikmesine neden olmaktadır.

## 3.4 AESX Yapısı

Basit bir teknik olan anahtar beyazlatma tekniği ile blok şifreleme algoritmaları anahtar arama stratejilerine karşı daha dirençli bir hale getirilebilmektedir (Bkz. Şekil 2.2). Anahtar arama stratejileri şifreleme algoritmalarını bir kara kutu gibi düşünerek, şifreleme algoritmalarının iç yapısını ve kriptografik özelliklerini göz ardı ederler. AES şifreleme algoritmasında kullanılan anahtar ekleme katmanı algoritmanın bir iç özelliği olduğu için anahtar arama stratejileri tarafından ihmal edilmektedir.

AESX şifreleme algoritmasında k anahtarına ek olarak, AES şifreleme algoritmasının öncesinde düz veriyi, sonrasında şifreli veriyi xor maskeleme

işlemi yapmak için k1 ve k2 anahtarları kullanılmaktadır (Şekil 3.16).

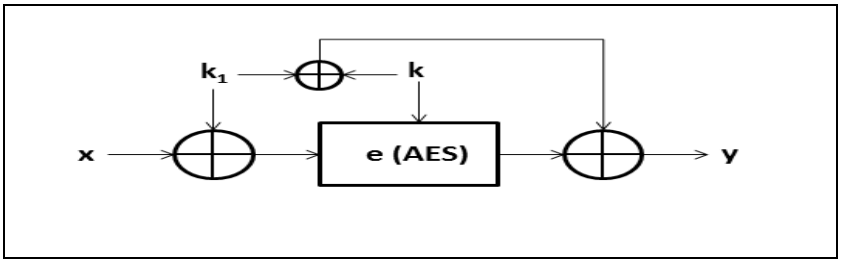



Şekil 3.16 AESX şifreleme algoritması

Burada y şifreli verisini elde etmek için $y = e_{k,k1,k2}(x) = e_k(x \oplus k1) \oplus k2$ şifreleme işlemi, x düz verisini elde etmek için $x = e^{-1}_{k,k1,k2}(x) = e^{-1}_k(x \oplus k1) \oplus k2$ çözme işlemi kullanılır. Blok şifreleme algoritmalarında kullanılan anahtar beyazlatma işleminde k1 ve k2 anahtarları girdi ve çıktı bloğu xor maskeleme işlemi yaptıkları için, uzunluklarının blok uzunluğuna eşit olması gereklidir. AES şifreleme algoritmasında blok uzunluğu 128 bit olduğu için, k1 ve k2 anahtarlarının uzunlukları 128 bittir.

## 3.5 AESX$^+$ Yapısı

Önerilen model AESX$^+$ şifreleme algoritmasında, AESX şifreleme algoritmasından farklı olarak k2 anahtarı yerine k ve k1 anahtarlarının xor işlemine tabi tutulması ile oluşturulmuş bir anahtar kullanılmaktadır (Şekil 3.17).




Şekil 3.17 AESX$^+$ şifreleme algoritması

AESX$^+$ şifreleme algoritmasında ise y şifreli verisini elde etmek için $y = e_{k,k1}(x) = e_k(x \oplus k1) \oplus (k \oplus k1)$ şifreleme işlemi, x düz verisini elde etmek için $x = e^{-1}_{k,k1}(x) = e^{-1}_k(x \oplus k1) \oplus (k \oplus k1)$ çözme işlemi kullanılır. Önerilen model AESX$^+$ şifreleme algoritması ile k2 anahtarının üretilmesine ihtiyaç duyulmamaktadır.

# DÖRDÜNCÜ BÖLÜM

## 4.PERFORMANS KARŞILAŞTIRMASI

Bilgi teknolojileri ortaya çıktığı ilk zamanlardan beri sürekli büyüme ve geliştirme göstermektedir. Bilgi teknolojileri hangi alanda gelişme gösterirse göstersin beklenen en büyük özellik her zaman daha hızlı olmasıdır. Ortaya çıkan her yeni teknolojinin bir önceki teknolojiden daha hızlı olması beklenir.

Günümüzde bilgi teknolojilerinde ürünlerin performans özelliklerinin iyi olması giderek daha fazla önem kazanmaktadır. Birbirleriyle uyumlu olan sistemlerde her bir ürünün hızı, sistemin genel hızını etkilemekte ve performansı değiştirmektedir. Dolayısıyla bilgiye erişimde ürünlerin performansına göre ciddi farklılıklar gözlenebilmektedir.

Çalışmanın bu bölümünde AESX$^{+}$, AESX, AES-128 ve AES-256'nın literatürde bulunan hız ölçümleri ile açık kaynak kripto kütüphanesinde uygulaması yapılarak elde edilmiş olan hız ölçümleri kullanılarak performans karşılaştırılması yapılacaktır.

### 4.1 AES-128 ve AES-256'nın Mevcut Performans Testleri

AES-128 şifreleme algoritması şifreleme işlemini 10 turda gerçekleştirir. AES-256 şifreleme algoritması ise şifreleme işlemini 14 turda gerçekleştirir. AES-128 şifreleme algoritmasının performansı AES-256'ya göre %40 civarında daha fazladır.

AES-128, AES-256 ve diğer şifreleme algoritmalarının şifreleme/çözme işlemleri özel yazılım ve cihazlarla yapılmaktadır. Bunun yanında bilimsel çalışmalar ve özel kullanımlar için açık kaynak kripto kütüphaneleri bulunmaktadır. Bu kütüphaneler c, c++, java, python vb. diğer dillerle yazılmış olup kullanılmış olan kodlar herkes için erişilebilir durumdadır.

Crypto++ açık kaynak kütüphanesi, c++ diliyle, Dr Wei Dai tarafından yazılmış ve ilk olarak 1995 yılında yayınlanmıştır. Kütüphane, 32-bitlik ve 64-bitlik mimaride kullanımı yaygın olan işletim sistemlerinde çalışmaktadır.

Crypto++ kütüphanesi API (Uygulama programlama ara yüzü) ile birçok kriptografik şemayı ve şifreleme modunu içermektedir. Crypto++ kütüphanesinde AES şifreleme algoritmasının farklı anahtar uzunluklarında ve şifreleme modlarında bulunan versiyonlarının performans değerlendirmeleri elde edilebilmektedir (Şekil 4.1).

**Crypto++ 5.6.5 Benchmarks**

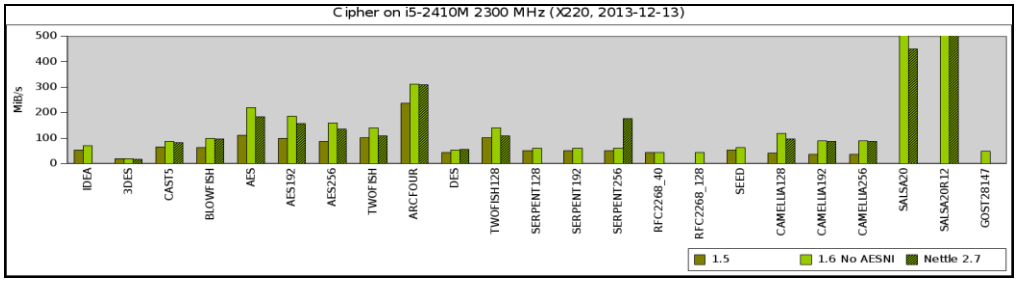
Here are speed benchmarks for some commonly used cryptographic algorithms.

The host OS is Fedora release 25 (x86_64). The host CPU is a 6th generation Skylake, frequency is 3.14713e+09 Hz.

| Algorithm | MiB/Second | Cycles Per Byte | Microseconds to Setup Key and IV | Cycles to Setup Key and IV |
|---|---|---|---|---|
| AES/CTR (128-bit key) | 3735 | 0.8 | 0.292 | 917 |
| AES/CTR (192-bit key) | 3174 | 0.9 | 0.282 | 888 |
| AES/CTR (256-bit key) | 2758 | 1.1 | 0.306 | 964 |
| AES/CBC (128-bit key) | 879 | 3.4 | 0.227 | 715 |
| AES/CBC (192-bit key) | 753 | 4.0 | 0.216 | 679 |
| AES/CBC (256-bit key) | 659 | 4.6 | 0.242 | 761 |
| AES/OFB (128-bit key) | 827 | 3.6 | 0.288 | 906 |
| AES/CFB (128-bit key) | 858 | 3.5 | 0.307 | 966 |
| AES/ECB (128-bit key) | 4140 | 0.7 | 0.081 | 254 |

Şekil 4.1 Crypto++ 5.6.5 testleri (Cryptopp.com, 2016)

Libgcrypt açık kaynak kripto kütüphanesi, c diliyle, Werner Koch tarafından ve Alman Hükümeti tarafından desteklenen GNUPG (GNU Privacy Guard) projesinin ayrı bir modülü olarak geliştirilmiştir. Farklı işletim sistemlerinde çalışma özelliğine sahip olup birçok şifreleme algoritmasını içermektedir. Libgcrypt kütüphanesinin AES şifreleme algoritmasını da içeren performans değerlendirmeleri bulunmaktadır (Şekil 4.2).




Şekil 4.2 Libgcrypt testleri (Gnupg.org, 2017)

Openssl açık kaynak kripto kütüphanesi, 1998 yılında, c dili ve assembly ile OpenSSL projesi tarafından bilgisayar ağlarında güvenli iletişimi sağlamak maksadıyla geliştirilmiştir. Openssl kütüphanesi SSL ve TLS protokollerini içermekte olup, çoğunlukla Unix ve benzeri işletim sistemlerine uygundur. Farklı işletim sistemlerinde ve cihazlarda denenmiş olan Openssl kütüphanesinin, farklı anahtar uzunluklarındaki AES şifreleme

algoritmasını içeren performans değerlendirmeleri bulunmaktadır (Şekil 4.3).

**New Benchmark Table**

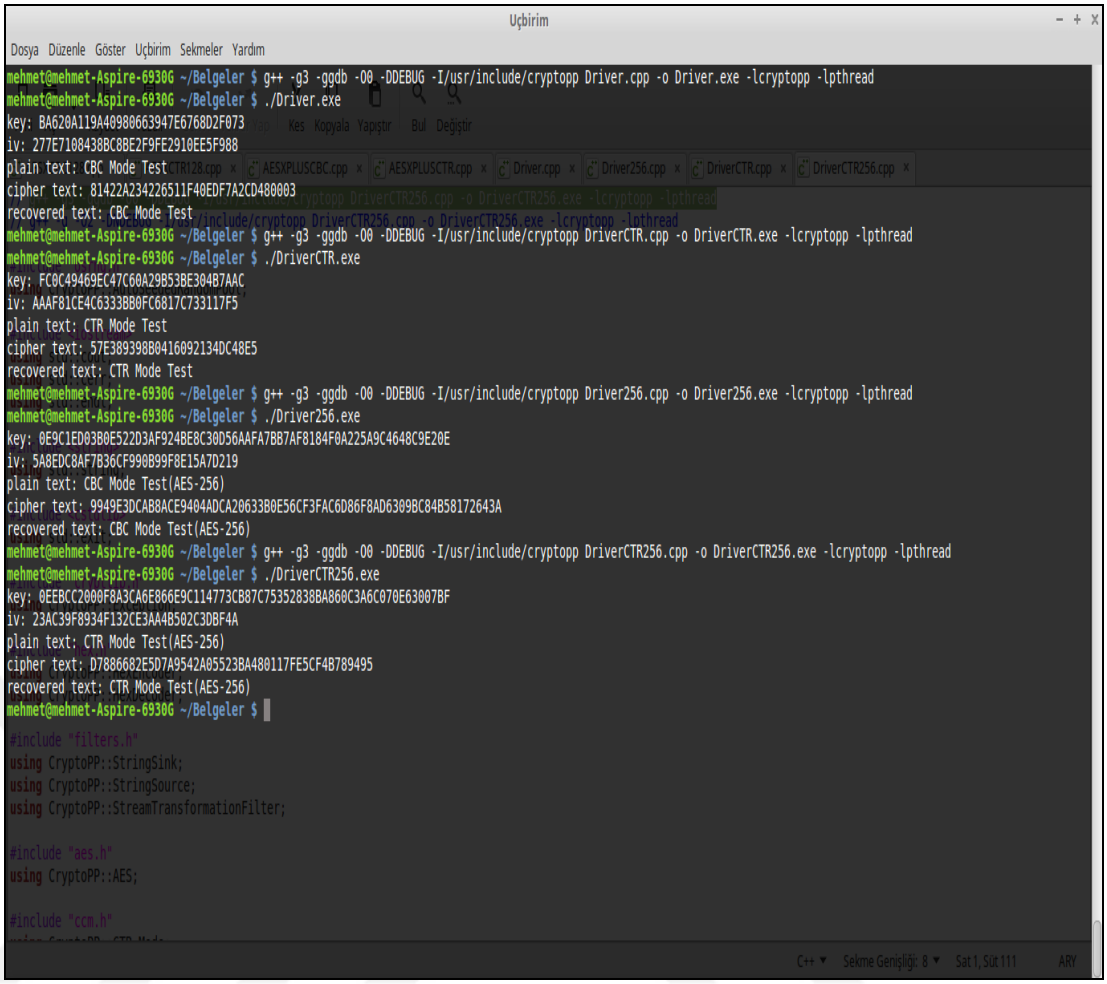
| OS | SoC | Device | CPU | BogoMIPS | OpenSSL Version | AES-128 | AES-192 | AES-256 |
|---|---|---|---|---|---|---|---|---|
| Backfire 10.03.1 | bcm6348 | 96348GW-11 | Broadcom BCM6348 V0.7 | 254.97 | 0.9.8x | 2814790 | 2455550 | 2124200 |
| r48532 | Ralink RT5350 id:1 rev:3 | A5-V11 | MIPS 24KEc V4.12 | 239.61 | 1.0.2g | 5052420 | 4348250 | 3824300 |
| r39183 GCC 4.8-linaro | Ralink RT3052 | ALPHA ASL-26555 | MIPS 24KEc V4.12 | 255.59 | 1.0.1e | 4931580 | 4239020 | 3737600 |
| r46163 (15.05-rc3) | BCM4708A0 | ASUS RT-AC68U | ARMv7 Processor rev 0 (v7l) | 1594.16 | 1.0.2d | 14129500 | 12396500 | 11477620 |
| r46767 (15.05 for BCM5301X) | BCM4708A0 | ASUS RT-AC68U | ARMv7 Processor rev 0 (v7l) | 1594.16 | 1.0.2d | 14084130 | 12310560 | 12341430 |
| r38297 | BCM5358 | Alcatel RG200E-CA | MIPS 74Kv V4.9 | 249.34 | 1.0.1e | 8890180 | 7710380 | 6783480 |
| r42625 | Geode(TM) Integrated Processor by AMD PCS | Alix2d13 | 500 MHz AMD Geode LX800 | 996.04 | 1.0.2a | 4092210 | 3543870 | 3138990 |
| Chaos Calmer (Linino) | Atheros AR9342 rev 3 | Arduino Tian | MIPS 74Kc V4.12 | 266.64 | 1.0.2d | 9596090 | 8250100 | 7143110 |
| Attitude Adjustment (Linino) | Atheros AR9330 rev 1 | Arduino Yun | MIPS 24Kc V7.4 | 265.42 | 1.0.1h | 5078280 | 4368490 | 3836380 |

Şekil 4.3 OpenSSL new benchmark table (Wiki.openwrt.org, 2017)

## 4.2 Crypto++ Kullanılarak Yapılan Performans Testleri

Crypto++ kütüphanesi simetrik blok şifreleri, genel şifreleme modlarını, akım şifreleme, ortak-anahtar kriptografi, Rijndael ve diğer AES aday şifreleme algoritmaları, şemalar, tek yönlü karma fonksiyonları, mesaj asıllama kodlarını içermektedir. Kütüphane için yazılmış olan API kaynakçası ile kullanmak istenilen sınıfların özellikleri belirtilmekte ve buradan indirilip kullanılabilmektedir. Bu şekilde AES kodları indirilerek derlenmiş olup, ekran görüntüleri Şekil 4.4'de gösterilmiştir.

Crypto++ kütüphanesi kullanılarak yapılmış olan birçok performans testleri bulunmaktadır. Donta (2007) Crypto++ kütüphanesi kullanarak DES, DESX, 3DES ve AES adayları olan Rijndael ve Blowfish şifreleme algoritmalarının performans karşılaştırmalarını yapmıştır (Şekil 4.5).




Şekil 4.4 AES şifreleme/çözme ekran görüntüsü

| *Algorithm* | *Megabytes(2^20 bytes) Processed* | *Time Taken* | *MB/ Second* |
|---|---|---|---|
| Blowfish | 256 | 3.976 | 64.386 |
| Rijndael (128-bit key) | 256 | 4.196 | 61.01 |
| Rijndael (192-bit key) | 256 | 4.817 | 53.145 |
| Rijndael (256-bit key) | 256 | 5.308 | 48.229 |
| DES | 128 | 5.998 | 21.34 |
| (3DES)DES-XEX3 | 128 | 6.159 | 20.783 |
| (3DES)DES-EDE3 | 64 | 6.499 | 9.848 |

Şekil 4.5 Crypto++ kullanılarak yapılan karşılaştırma sonuçları (Donta, 2007)

### 4.2.1 Performans Testinin Yapılışı

Performans testleri Intel Core 2 Duo 2.67 Ghz işlemcili, 4 GB bellekli, Linux Mint 17.3 Rosa Xfce İşletim sistemi üzerinde, Crypto++ 5.6.4 kütüphanesi kullanılarak yapılmıştır. AESX ve AESX$^{+}$ da kullanılan xor işlemi için kütüphanede bulunan xorbuf fonksiyonu kullanılmıştır (Şekil 4.6). Bu fonksiyon gelen veri 1 bayttan daha büyük olduğunda xor işlemini daha hızlı yapabilmektedir.

```
void xorbuf(byte *buf, const byte *mask, size_t count)
{
    CRYPTOPP_ASSERT(buf != NULLPTR);
    CRYPTOPP_ASSERT(mask != NULLPTR);
    CRYPTOPP_ASSERT(count > 0);

    size_t i=0;
    if (IsAligned<word32>(buf) && IsAligned<word32>(mask))
    {
        if (!CRYPTOPP_BOOL_SLOW_WORD64 && IsAligned<word64>(buf) && IsAligned<word64>(mask))
        {
            for (i=0; i<count/8; i++)
                ((word64*)(void*)buf)[i] ^= ((word64*)(void*)mask)[i];
            count -= 8*i;
            if (!count)
                return;
            buf += 8*i;
            mask += 8*i;
        }

        for (i=0; i<count/4; i++)
            ((word32*)(void*)buf)[i] ^= ((word32*)(void*)mask)[i];
        count -= 4*i;
        if (!count)
            return;
        buf += 4*i;
        mask += 4*i;
    }

    for (i=0; i<count; i++)
        buf[i] ^= mask[i];
}
```

Şekil 4.6 Xorbuf fonksiyonu

Performans testleri için AES modları olarak CBC ve CTR kullanılmıştır. Performans testlerinde sadece şifreleme işlemi ele alınmıştır. Test verisi olarak Crypto++ kütüphanesinde bulunan rijndael.dat verisi kullanılmıştır. Şifreleme sürelerinin hesaplanmasında daha ayırt edici büyük sayılar elde etmek maksadıyla şifreleme işlemi bir for döngüsü içinde 200000 tekrar ile yapılmıştır (Şekil 4.7).

```
        startTime = clock();

for ( int i = 0; i < 200000; i++ )
    {
        prewhiten = XOR(plain, encodedkey1);

                StringSource s(prewhiten, true,
                        new StreamTransformationFilter(e,
                                new StringSink(cipher)
                        ) // StreamTransformationFilter
                ); // StringSource


        postwhiten = XOR(encoded, encodedkey2);

     } // end-for
        finishTime = clock();

        double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;


        std::cout << "Encryption execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;

        std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;

        std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;

        double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;

        std::cout << "Encryption execution time MİB/S: " << data_rate_MiBps << " MİB/S." << std::endl;
```

Şekil 4.7 Performans ölçen kod parçası

### 4.2.2 Performans Testinin Sonuçları

Performans test sonuçlarını incelemeden önce, AES şifreleme algoritmasında anahtar beyazlatma işlemi için kullandığımız fonksiyonun aynısını kullanan DES ve DESX şifreleme algoritmasının Crypto++ 5.6.4 kütüphanesindeki karşılaştırmalı değerlendirme sonuçları Şekil 4.8'de gösterilmiştir. Bu sonuçlara göre elde ettiğimiz sonuçların doğruluğunu yorumlayabilmemiz mümkün olacaktır.

| Algorithm | MiB/Second | Cycles Per Byte | Microseconds to Setup Key and IV | Cycles to Setup Key and IV |
|---|---|---|---|---|
| DES/CTR (64-bit key) | 49 | 50.2 | 6.396 | 16630 |
| DES-XEX3/CTR (192-bit key) | 45 | 55.4 | 6.738 | 17519 |

Şekil 4.8 DES ve DESX Crypto++ 5.6.4 karşılaştırmalı değerlendirme sonuçları

Performans testleri için CBC ve CTR modlarındaki AESX$^{+}$, AESX, AES-128 ve AES-256 şifreleme algoritmaları işletim sisteminde GCC (GNU Compiler Collection) derleyicisi ile çalıştırılmıştır. AES-128 ve AES-256 şifreleme algoritmalarının şifreleme ölçümleri Çizelge 4.1'de gösterilmiştir.

Çizelge 4.1 AES-128 ve AES-256 şifreleme hızı ölçümleri

| | Şifreleme döngüsü zamanı (Mikrosaniye) | Anahtar ve IV kurulumu süresi | Düz Metin uzunluğu (Bayt) | Şifreleme döngüsü uygulama zamanı (MİB/Saniye) |
|---|---|---|---|---|
| AES/CBC (128-bit) | 2409,69 | 0,191 | 1020 | 80,7363 |
| AES/CBC (256-bit) | 2835,27 | 0,220 | 1020 | 68,6176 |
| AES/CTR (128-bit) | 2187,67 | 0,234 | 1020 | 88,9302 |
| AES/CTR (256-bit) | 2670,78 | 0,176 | 1020 | 72,8436 |

Çizelge 4.1'de görüldüğü üzere AES-128 şifreleme algoritmasının şifreleme hızı AES-256 şifreleme algoritmasına göre yaklaşık %20 daha hızlıdır. Bu sonuç Crypto++ kütüphanesi test sonuçlarında mevcut olan yaklaşık %40 oranı ile tutarlı değildir. Buna Crypto++ kütüphanesinin API'de mevcut olan sınıfların kullandığı komut zinciri (pipelining) neden olmaktadır.

$AESX^{+}$ ve AESX şifreleme algoritmalarının şifreleme ölçümleri Çizelge 4.2'de gösterilmiştir.

Çizelge 4.2 AES-$X^+$ ve AES-X şifreleme hızı ölçümleri

| | Şifreleme döngüsü zamanı (Mikrosaniye) | Anahtar ve IV kurulumu süresi | Düz Metin uzunluğu (Bayt) | Şifreleme döngüsü uygulama zamanı (MİB/Saniye) |
|---|---|---|---|---|
| AES-X/CBC (128-bit) | 2702,68 | 0,211 | 1020 | 71,9838 |
| AES-$X^+$/CBC (128-bit) | 2703,23 | 0,204 | 1020 | 71,9692 |
| AES-X/CTR (128-bit) | 2444,74 | 0,207 | 1020 | 79,5788 |
| AES-$X^+$/CTR (128-bit) | 2425,57 | 0,179 | 1020 | 79,3248 |

Çizelge 4.2'de görüldüğü gibi AESX$^+$/CBC ve AESX/CBC şifreleme algoritmaları AES/CBC(256-bit) şifreleme algoritmasına göre yaklaşık %5 oranında, AESX$^+$/CTR ve AESX/CTR şifreleme algoritmaları AES/CTR(256-bit) şifreleme algoritmasına göre yaklaşık %10 oranında daha hızlıdır.

Crypto++ kütüphanesi Intel işlemcilerin daha hızlı şifreleme/çözme yapmasını sağlayan AES-NI özelliğini desteklemektedir. Performans testlerinde bu özellik kullanılarak elde edilen sonuçlarda AES için CTR ve CBC modlarında yaklaşık yüzde 2 oranında fark elde edilmiştir.

Elde edilen sonuçlara göre, DES ve AES şifreleme algoritmalarına DESX ve AESX (AESX$^+$) şifreleme algoritmalarının getirdiği ek yük Çizelge 4.3'de gösterilmiştir. Ek yük oranları hem CTR modu, hem de CBC modu için incelendiğinde, AES-128 şifreleme algoritmasına, AES-X ve AES-$X^+$

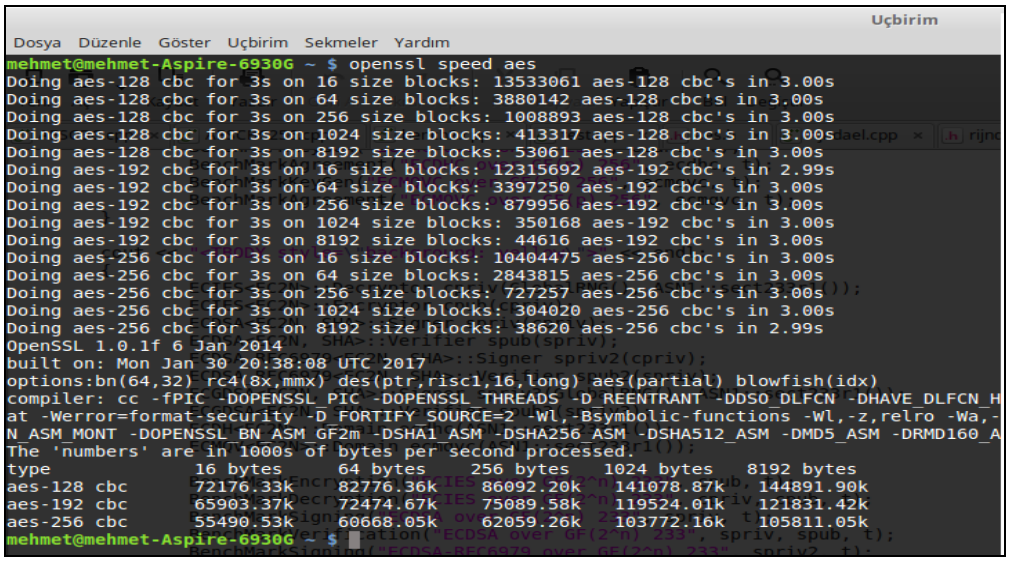
şifreleme algoritmaları, AES-256 şifreleme algoritmasına göre daha az ek yük getirmişlerdir.

Çizelge 4.3 DES, AES, DESX, AESX ve AESX$^{+}$ ek yük oranları

| Karşılaştırma Yapılan Algoritmalar | Oran (MİB/S) | Ek Yük (Overhead) ≈ |
|---|---|---|
| DES/CTR(64-bit) & DES-XEX3/CTR (192-bit) | 49 / 45 | % 8 |
| AES/CBC (128-bit) & AES/CBC (256-bit) | 80.73 / 68.61 | % 17.66 |
| AES/CBC (128-bit) & AES-X/CBC (AES-X$^{+}$/CBC) | 80.73 / 71.98 | % 12 |
| AES/CTR (128-bit) & AES/CTR (256-bit) | 88.93 / 72.84 | % 22 |
| AES/CTR(128-bit) & AES-X/CTR (AES-X$^{+}$/CTR) | 88.93 / 79.57 | % 11,76 |

## 4.3 Diğer Açık Kaynak Kütüphanelerin Test Sonuçları

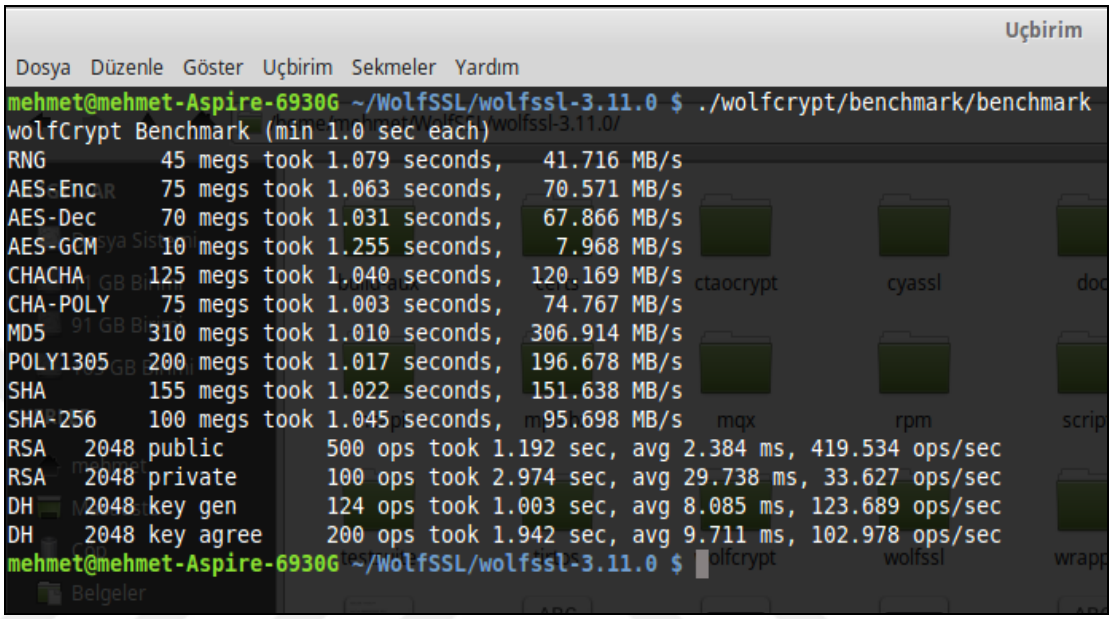
OpenSSL kripto kütüphanesi indirildikten sonra, komut satırına "openssl speed aes" yazdıktan sonra Şekil 4.9'de ekran çıktısı verilmektedir.




Şekil 4.9 OpenSSL AES test sonuçları

Şekil 4.9'de görüldüğü üzere AES-128 şifreleme algoritması 16 bayt veri için AES-256 şifreleme algoritmasına göre %30 oranında daha hızlı, 1024 bayt veri için %35 oranında daha hızlıdır.

Wolfcrypt kripto kütüphanesi indirildikten sonra, komut satırına "benchmark" yazdıktan sonra Şekil 4.10'daki ekran çıktısı elde edilmektedir.




Şekil 4.10 Wolfcrypt test sonuçları

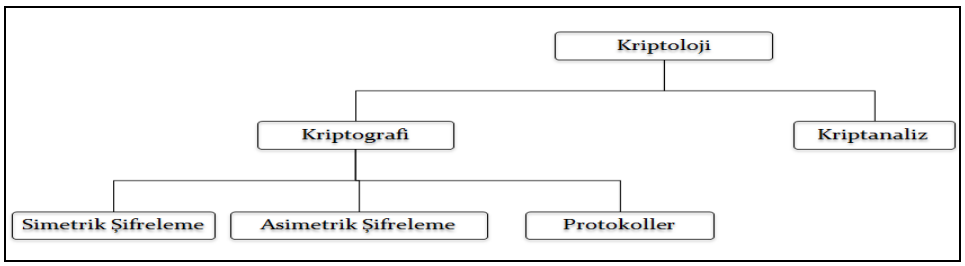
Şekil 4.10'da AES-128 için şifreleme/çözme test sonuçlarını içermektedir.

# BEŞİNCİ BÖLÜM

## 5.GÜVENLİK ANALİZİ

Güvenlik, herhangi bir zarardan korunma ve bu zarara karşı gösterilebilen direnç derecesidir. Bilgi güvenliği ise, verinin bulunduğu ortamda korunması ve doğru kişiye bozulmadan ulaştırılması demektir. Bilgi güvenliğinin amacı verinin gizliliğini, bütünlüğünü ve kullanılabilir olduğunu sağlamaktır. Bilgi güvenliğinde gizlilik, verinin yetkisiz erişime karşı korunması, bütünlük, verinin tutarlı, doğru ve eksiksiz olması ve kullanılabilirlik ise veriye zamanında güvenli erişim ve verinin kullanımı anlamına gelmektedir. Kriptografi ise bu kavramları sağlamak için kullanılan matematiksel yöntemlerdir.

Kriptoloji, kriptografiyi ve kriptanalizi içeren şifre bilimidir (Şekil 5.1). Kriptoloji, verinin kişiler ve kurumlar arası iletilmesinden, kripto sistemlerin oluşturulması ve bu sistemlerdeki güvenlik boşlukları gibi alanlarla alakalıdır. Kriptografi, gönderilen ve ya alınan mesajın gizliliğini, mesajın bütünlüğünü, gönderici ve alıcı için kimlik doğrulamayı ve mesaj oluşturan ya da gönderen için inkâr edilemezlik özelliklerini sağlar.

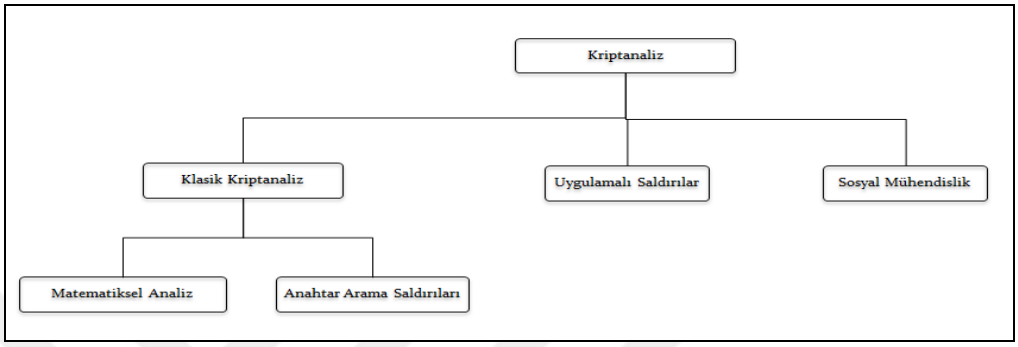



Şekil 5.1 Kriptoloji biliminin alanları (Paar and Pelzl, 2010)

Kriptanaliz ise bir kripto sistemi çözme ve ya analiz etmedir. Kriptanalizde kriptografinin aksine, güvenli veriyi kırma işlemi üzerine çalışılır. Bir kripto sistemin kırılması etik dışı gibi gözükse de bir kripto sistemin güvenli olup olmadığını anlamamızın tek yöntemi kriptanalizdir. Çoğu araştırmacı ve akademisyen olan kriptanalistler, gelişen teknolojiyi de kullanarak kripto sistemlerin zayıf yönlerini ortaya koyarak, yeni ve daha güçlü kripto sistemlerin gelişmesine katkıda bulunurlar.

Kriptanaliz yöntemleri Şekil 5.2'de gösterilmiştir. Klasik kriptanaliz, y

şifreli metinden x düz metnini elde etmeyi ya da y şifreli metinden k anahtarını elde etmeyi amaçlamaktadır. Uygulama saldırıları, saldırganın fiziksel olarak kripto sisteme ulaşabildiği, yan-kanal analizi gibi, gizli anahtar üreten işlemcinin tükettiği elektrik hesaplaması ile gizli anahtarı ele geçirmeyi hedefleyen saldırılar örnek verilebilir. Sosyal mühendislik saldırıları, insanın içinde bulunduğu, rüşvet, kandırma, şantaj gibi gizli anahtarı ele geçirmeyi hedefleyen saldırılardır.




Şekil 5.2 Kriptanaliz alanları (Paar and Pelzl, 2010)

Çalışmanın bu bölümünde AESX$^{+}$, AESX, AES-128 ve AES-256'nın güvenlikleri analiz edilerek birbirlerine karşı zayıflıkları ve üstünlükleri ortaya konulacaktır.

## 5.1 Anahtar Beyazlatma Tekniğine Yapılan Saldırı Türleri

Anahtar beyazlatma tekniği, daha önce belirtildiği gibi, blok şifrelemede girdi ve çıktı bloğunun, yeni bir anahtar ile xor işleminden geçirilerek güvenliğinin artırılması tekniğidir (Bkz. Şekil 2.2). Bu tekniğin kullanıldığı en çok bilinen şifreleme algoritması ise DESX şifreleme algoritmasıdır.

DESX şifreleme algoritması, Ron Rivest tarafından, 1984 yılında, $DESX_{k.k1.k2}(x) = k2 \oplus DES_k(k1 \oplus x)$ şeklinde tanımlanmıştır. Görüldüğü üzere DESX şifreleme algoritması, AESX ve AESX$^{+}$ şifreleme algoritmalarının yapısı ile benzerlik göstermektedir. Dolayısıyla AESX ve AESX$^{+}$ şifreleme algoritmalarının güvenliklerini analiz edebilmek için öncelikle DESX şifreleme algoritmasına yapılmış olan bilinen en iyi saldırıları incelemek gereklidir. DESX şifreleme algoritmasına yapılmış olan bilinen en iyi saldırlar:

Kodkitabı Saldırısı; adından da anlaşılacağı üzere saldırgan tarafından şifreli veri ile düz veriyi ilişkilendirebileceği bir kod kitabı tablosu oluşturarak yaptığı bir saldırı türüdür. DESX şifreleme algoritması için, bu saldırıda saldırgan $2^{64}$ adet düz veri - şifreli veri çiftine ihtiyacı bulunmaktadır. Her ne kadar bu saldırıda $2^{64}$ adet hesaplama ve hafıza gerekli olsa da, bu saldırıda anahtarı elde etmeden şifreli veriyi çözmek mümkündür (Nachef et al., 2017). Ancak AESX ve AESX$^{+}$ şifreleme algoritmalarının 128 bitlik blok uzunluğu göz önüne alındığında bu saldırı türü son derece elverişsizdir.

Lineer Kriptanaliz; bir bilinen düz veri saldırısıdır. 1993 yılında Matsui tarafından duyurulan saldırı DES ve DESX şifreleme algoritmalarına karşı uygulanmıştır. Lineer kriptanaliz, düz veri ile şifreli veri arasındaki doğrusal yapıyı kullanır. DESX şifreleme algoritması için bu saldırıda saldırganın $2^{60}$ adet düz veri – şifreli veri çiftine, $2^{60}$ hesaplama ve küçük bir hafızaya ihtiyacı bulunmaktadır. Burada saldırgana toplam düz veri – şifreli veri çiftinin 1/16'sı yeterli olmaktadır (Nachef et al., 2017). Anahtar arama saldırısı bir lineer kriptanalizdir ve blok şifrelemeye karşı kullanılan en yaygın saldırı türüdür. AESX ve AESX$^{+}$ şifreleme algoritmalarının güvenlik analizleri için kullanılması mümkündür.

Daeman'ın Saldırısı; bir seçilmiş şifreli veri saldırısıdır. Daemen 1991'de $2^{n/2}$ karmaşıklığında bir anahtar ele geçirme saldırısı duyurmuştur (Advances in Cryptology, 2012). Bu saldırıda öncelikle fark ($\Delta$) seçilmektedir. $\ell$ değerleri için $\Delta F_0 = F_0(a) \oplus F_0(a \oplus \Delta)$ hesaplanır ve L listesine ($\Delta F_0$, a) çiftleri kaydedilir. $P' = P \oplus \Delta$ ile birlikte P düz verisi seçilir ve şifreli veri C ve C' sorulur. $\Delta C = C \oplus C'$ hesaplaması yapılır ve $\Delta C$'nin a elde etmek için L listesinde olup olmadığı kontrol edilir. Eğer $\Delta C$ L listesinin içinde varsa anahtar için bir aday bulunmuş demektir ve $K_0 = a \oplus P$ ve $K_1 = F_0(a) \oplus C$ hesaplanır. Eğer $\Delta C$ L listesinin içinde yoksa P düz verisi tekrar seçilerek işlemlere devam edilir. Burada en uygun $\ell$ değeri $2^{n/2}$ seçilerek saldırının karmaşıklığı $2^{n/2}$ çıkmaktadır.

Daeman'ın Saldırısı DESX şifreleme algoritmasında $2^{32}$ düz veri – şifreli veri çifti ve $2^{88}$ hesaplama ile anahtar ele geçirilebilmektedir. Daeman saldırısı DESX şifreleme algoritmasında, ortadaki DES algoritmasını Even-Mansour yapısındaki F fonksiyonu olarak görür. Dolayısıyla $2^{32}$ düz veri – şifreli veri çiftine ek olarak, $2^{32}$ DESX şifreleme algoritması için, $2^{56}$ DES şifreleme algoritması için olmak üzere $2^{88}$ hesaplamaya ihtiyaç duyulmaktadır

(Nachef et al., 2017). Bu saldırı türünün de AESX ve AESX$^+$ şifreleme algoritmalarının güvenlik analizleri için kullanılması mümkündür.

Kısmi Deşifreleme Saldırısı; bu saldırıda sadece blok boyutu yer almaktadır. Eğer 64 bitlik bloğun $2^{32}$'sinden daha fazlasını aynı anahtar ile şifreliyorsak, doğumgünü paradoksu ile, saldırgan bilinen metin saldırısı içinde bir 64 bitlik blok düz veri ile veya sadece şifreli veri saldırısı ile 2 düz veri bloğunu xor işlemi yaparak, yüksek ihtimalle çözecektir. Daha genel bir ifadeyle, A'nın tamsayı ve 1'den büyük olduğu yerde, eğer A.$2^{32}$ blokları aynı anahtar ile şifrelenmişse, saldırgan iyi bir olasılık ile $A^2$ adet düz veri bloğunu bilinen düz veri saldırısında veya $A^2$ adet düz veri bloğunu sadece şifreli veri saldırısı içinde xor işlemi yaparak bulabilir (Nachef et al., 2017). Bu saldırı türü AESX ve AESX$^+$ şifreleme algoritmaları için düşünüldüğünde saldırının gerçekleşebilmesi için 128 bitlik bloğun $2^{64}$'ünden daha fazlasını aynı anahtar ile şifrelenmesi gerekmektedir.

Biham Tipi Saldırı; bu saldırıda ise aynı A düz veri bloğunun farklı anahtarlarla şifrelendiğini varsayıyoruz. Saldırgan elinde mevcut olan anahtarla şifrelenmiş olan A'nın tablosunu oluşturur. Eğer A bu anahtarlardan biriyle şifrelenirse tabloya bakarak anahtarı ele geçirmek mümkün olacaktır. DESX için örnek verecek olursak saldırganın $2^{28}$ anahtarlar içinde bulunan bir anahtarı, $2^{92}$ hesaplama ve $2^{92}$ hafıza ile bulması olasıdır (Nachef et al., 2017). Bu saldırının yüksek hafıza ihtiyacından dolayı gerçekleştirilmesi çok zordur. Aynı şekilde AESX ve AESX$^+$ şifreleme algoritmalarının anahtar büyüklükleri düşünüldüğünde saldırının bu algoritmalara uygulanması elverişli değildir.

İlişkili anahtar saldırısı; Biryukov ve Wagner, 2000 yılında, DESX şifreleme algoritmasına Gelişmiş Kayma Saldırısı (Advanced Slide Attack) isimli, ilişkili anahtarlarla birlikte $2^{32}$ veri, $2^{88}$ hesaplama ve $2^{32}$ hafıza ile DESX anahtarını ele geçirebilen saldırıyı duyurmuşlardır. Bu saldırının özellikleri Daemen'ın saldırısına benzemektedir ancak bu saldırıda seçilmiş düz veri yerine anahtarlar arasındaki ilişki kullanılmaktadır (Nachef et al., 2017). İlişkili anahtar saldırısının DESX şifreleme algoritması için duyurulmuş olan başka türleri de bulunmaktadır. İlişkili anahtar saldırısının, AESX ve AESX$^+$ şifreleme algoritmalarının güvenlik analizleri için kullanımı uygundur.

İlişkili anahtar ayırıcı (Related-key distinguisher); DES şifreleme algoritmasında, eğer düz veride, anahtarda ve şifreli veride tüm 0'ları 1 ile ve tüm 1'leri 0 ile değiştirdiğimizde, orjinal şifreli veriyi elde edebiliyoruz (DES tamamlama özelliği). Bu DESX kara kutusunun, sadece q = 2 veri, 2 hesaplama ile ve bir ilişkili anahtar saldırısıyla birlikte 64 bitlik ideal bir şifreleme kara kutusundan ayırt edilmesine olanak tanır. Ancak burada saldırgan DESX şifreleme algoritmasının anahtarını değilde, sadece ilişkili anahtar ayırıcıyı elde etmektedir. Bundan dolayı bu saldırı türü güvenlik açısından önemli olarak gözükmemektedir (Nachef et al., 2017).

Sonuç olarak, DESX şifreleme algoritmasına yapılmış olan en önemli saldırı Daemen'ın saldırısıdır. Her ne kadar Daemen saldırısı 3 farklı anahtar kullanan 3DES şifreleme algoritmasının, DESX şifreleme algoritmasına göre kullanılmasının daha güvenli olduğunu gösterse de, DESX şifreleme algoritması basitliği ve hızlılığı ile daha etkili güvenlik sağlamaktadır (Nachef et al., 2017).

## 5.2 Anahtar Arama Saldırısı

Anahtar arama saldırısı bir diğer adıyla kaba kuvvet saldırısı, saldırganın doğru parolayı bulmak için birçok muhtemel parolayı denemesidir. Bu saldırıda saldırgan, doğru anahtarı buluncaya kadar sürekli olarak tüm olası anahtarları kontrol eder. Burada saldırganın en az bir şifreli metin ve düz metin çiftini ele geçirdiğini varsayarsak, saldırganın yapması gereken tüm olası anahtarlarla şifreli metni çözmeyi deneyip eldeki düz metin ile aynı olan deşifre edilmiş düz metine ulaşmak olacaktır. Saldırgan doğru çifti bulmuşsa doğru anahtarı da bulmuş demektir.

Anahtar arama saldırısı, şifrelenmiş veriyi çözme girişimi için kullanılabilecek bir kripto analitik saldırı türüdür. Eğer bir kripto sistemin faydalanılabilecek başka bir zayıf yönü yoksa bu saldırı yöntemi kullanılabilmektedir. Örneğin eski bir şifreleme yöntemi olan harfleri değiştirme şifreleme tekniğini ele alalım. Burada A harfinin yerine 29 harfli alfabemizden başka bir harf seçtiğimizi düşünelim. Geriye B harfi için 28 seçenek kalır ve tüm harfler için hesaplama yaptığımızda; anahtar uzayı= 29. 28…3.2.1=$2^{102}$ olarak karşımıza çıkar. Bu anahtar uzayında başarılı bir anahtar arama saldırısı yapmak günümüz teknolojisiyle pek mümkün değildir. Ancak bu durumda harflerin kullanım sıklıkları analizine göre bir saldırı

yapılarak anahtar daha kolay bir şekilde ele geçirilebilir.

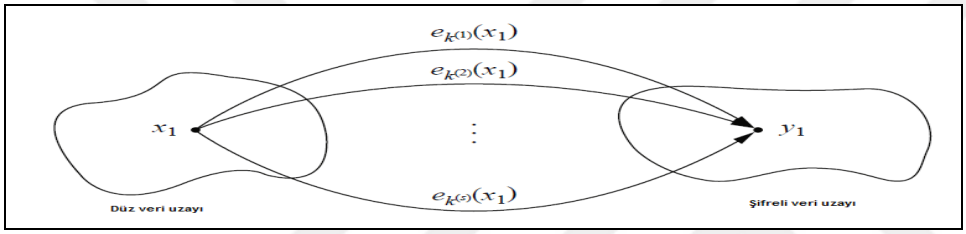
Anahtar arama saldırıları simetrik şifreleme için her zaman kullanılabilecek bir saldırıdır. Bir kripto sistemin şifrelediği veri için kullanmış olduğu muhtemel anahtar sayısı çok fazla ise ve bu anahtarların hepsini test etmek modern bilgisayarlarla çok uzun zaman alıyor ise bu kripto sistem anahtar arama saldırısına göre güvenlidir. Çizelge 5.1'de farklı anahtar uzunluklarındaki simetrik şifreleme için anahtar arama saldırılarının tahmini zamanları gösterilmektedir.

Çizelge 5.1 Simetrik şifrelemeye karşı tahmini anahtar arama saldırısı süreleri (Paar and Pelzl, 2010)

| Anahtar Uzunluğu | Anahtar Uzayı | Güvenlik Sağlama Süresi |
|---|---|---|
| 64 | $2^{64}$ | Kısa Dönem (Birkaç gün) |
| 128 | $2^{128}$ | Uzun Dönem (Kuantum bilgisayarların olmadığını varsayarak onlarca yıl) |
| 256 | $2^{256}$ | Uzun Dönem (Kuantum bilgisayarlar olduğu varsayılarak onlarca yıl) |

Teknoloji çok hızlı bir şekilde gelişmekte ve gelecek teknolojisi ile ilgili kesin bir tahmin yapmak gittikçe zorlaşmaktadır. Kısa dönem yapılacak tahminler için Moore yasası idealdir. Moore yasasına göre, mevcut teknoloji gücü her 18 ayda bir ikiye katlanmaktadır. Örneğin, günümüzde 1000 lira değerinde olan bir teknoloji 3 yıl sonra 250 lira değerinde olacaktır. Moore yasası aynı zamanda üstel bir fonksiyondur. Günümüzde bilgi teknolojisini ikiye katlamak için x birime ihtiyacımız var ise 15 yıl sonra aynı para ile $2^x$ bilgi teknolojisini ikiye katlayabiliriz. Örnek verecek olursak bir şifreleme algoritmasını kırmak için sahip olacağımız teknoloji 100000 lira ise 15 yıl sonra aynı teknoloji 100 lira değerinde olacaktır (Paar and Pelzl, 2010).

Anahtar arama saldırısı bazı durumlarda yanlış sonuç verebilir. Şifreleme için kullanılmış olan anahtar uzayına ve veri uzayına bağlı olarak, bulunan bazı anahtarlar şifrelemede kullanılmamış olabilir. Örneğin 64 bitlik bir şifreleme bloğunun 80 bitlik bir anahtar uzayı ile şifrelendiğini varsayalım. Bu durumda düz veriyi $2^{80}$ adet anahtar ile şifrelediğimizde, elimizde $2^{80}$ adet şifreli veri olacaktır. Ancak blok uzunluğu 64 bit olduğu için şifreli veri sadece $2^{64}$ adet olacaktır. Dolayısıyla $2^{80} / 2^{64} = 2^{16}$ adet anahtar Şekil 5.3'de görüldüğü üzere $e_k(x_1) = y_1$ işlemini yapacaktır. Yani şifreleme için kullanmış olduğumuz anahtar uzayı, şifrelemeye giren veri bloklarının genişliğinden daha büyükse bazı anahtarlar aynı şifreli veriyi gösterecektir. Burada anahtar arama saldırısı hala yapılabilmektedir ancak sadece elde bulundurulan düz veri - şifreli veri çifti sayısı artırılmalıdır.




Şekil 5.3 Büyük anahtar uzayının aynı şifreli veriyi göstermesi (Paar and Pelzl, 2010)

### 5.2.1 AESX ve AESX$^+$ Anahtar Arama Analizi

Anahtar arama stratejileri, bir şifreleme algoritmasının matematiksel veya kriptanalitik özelliklerini önemsememektedir. Dolayısıyla genel bir anahtar arama saldırısında, bir kripto sistemin iç yapısı kullanılmayacağı için saldırı yapılacak kripto sistem bir kara kutu olarak düşünülebilir (Rogaway, 1996). Bundan dolayı anahtar arama saldırısının güvenlik analizini yapabilmek maksadıyla Killian ve Rogaway'in (2000) DESX şifreleme algoritmasının anahtar arama saldırı analizini yapmak için kullanmış oldukları modeli AESX ve AESX$^+$ şifreleme algoritmaları için de kullanabiliriz.

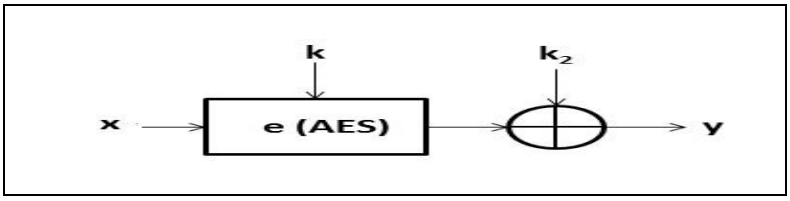
Bu modelde bir blok şifreleme için K anahtar uzunluğu, n ise blok uzunluğu olarak kabul edilmektedir. İdeal bir blok şifreleme olan $F : \{0,1\}^K \times \{0,1\}^n \rightarrow \{0,1\}^n$ şeklinde gösterilmektedir. Bu model için bir anahtar arama saldırı algoritmasına iki kâhin verilmiştir: birincisi (k,x) girdisine, F(k,x) değerini döndürür, diğeri ise (k,y) girdisine $F^{-1}(k,y)$ değerini döndürmektedir. Burada $F^{-1}(k,y)$, x noktasını gösterir, böylece F(k,x) = y olur (Kilian and Rogaway, 2000).

Genel bir anahtar arama saldırganı, F'ye olan tek erişimi $F/F^{-1}$ kâhinleri aracılığıyla, sınırsız zaman kullanarak, kriptanalitik hesaplamalar yapabilir. Saldırganın başarı oranını analiz etmek için, bu hesaplamaları yapmak için kullandığı $F/F^{-1}$ kahinlerine yaptığı erişim sayısı kullanılır.

Kilian ve Rogaway (2000) anahtar arama saldırısını DESX şifreleme algoritmasına uygulamak için DESX yapısını genelleştirmişlerdir. Dolayısıyla saldırı ve saldırı analizi anahtar beyazlatma tekniği kullanan tüm blok şifreleme algoritmaları için kullanılabilmektedir. Bunu yapmak için herhangi bir F blok şifreleme algoritması için, $FX : \{0,1\}^{K+2n} \times \{0,1\}^{n} \rightarrow \{0,1\}^{n}$ olacak şekilde $FX(k.k1.k2) = k2 \oplus F(k,k1 \oplus x)$ şeklinde tanımlanmıştır. Burada FX için anahtar K = k.k1.k2 şeklinde gösterilmektedir.

Kilian ve Rogaway (2000) FX fonksiyonun anahtar aramaya karşı dayanıklılığını araştırmak için "FX-veya-π" oyunu ile bir test yapmaktadırlar. Bu oyunda öncelikle F ve $F^{-1}$ için kâhinler içeren bir saldırgan A belirlenmektedir. Saldırgan A'ya rastgele seçilmiş olan "şifreleme kâhini" E verilir. Burada E kahini, 0.5 olasılık ile $K \in \{0,1\}^{K+2n}$ rastgele seçerek $E(x) = FX_K(x)$ üretir veya yine 0.5 olasılık ile rastgele bir permütasyon $\pi:\{0,1\}^{n} \rightarrow \{0,1\}^{n}$ seçerek $E(x) = \pi(x)$ üretir. Saldırgan A'nın görevi, E kâhinin vermiş olduğu E(x)'in hangi yoldan üretilmiş olduğunu doğru tahmin etmektir. Eğer saldırganın doğru tahmin etme olasılığı 0.5'den önemli derecede büyükse, oyunu kazanır. "FX-veya-π" oyununu kazanmak için gerekli olan kaynaklar, F'i kırmak için gereken kaynaklardan büyükse, FX yapımının daha güvenli olduğunu söyleyebiliriz.

Bu modelde, girişte ve çıkışta xor işleminin önemini AESW analizi ile anlayabiliriz. k1 anahtarını 0 olarak düşünürsek $AESW_{k.k2}(x) = k2 \oplus AES_k(x)$ şeklinde olur (Şekil 5.4). AESW algoritmasına anahtar arama saldırısı için, saldırgana, herhangi bir x değeri için k ve $AESW_{k.k2}(x)$ verilirse, kolaylıkla $k2 = AESWk,k2(x) \oplus AES_k(x)$ hesaplamasını yaparak anahtarı ele geçirebilir. Aynı şekilde k2 anahtarı değeri 0 olarak düşünüldüğünde k1 anahtarı aynı yöntemle kolaylıkla elde edilmektedir. Sonuç olarak AESW genel anahtar arama saldırısına karşı AES şifreleme algoritmasına göre daha dirençli değildir. AESX şifreleme algoritmasını anahtar arama saldırılarına daha dirençli hale getirmek için mutlaka girişte ve çıkışta xor işleminden geçirmek gereklidir.




Şekil 5.4 AESW modeli

Anahtar arama saldırgan algoritması A, "FX veya π" oyununda şifreleme kâhini E'ye yeterli sayıda sorgu yapabilmesi için, $F/F^{-1}$ kâhinlerine aşırı sayıda sorgu sormalı ve bu nedenle çok uzun süre çalışmalıdır. Saldırgan A'nın bu çalışmasını hesaplamak için, saldırganın elde edebileceği ‹x, FXK (x)› çiftlerinin sayısını m ile sınırlandırılır. Saldırganın, $F/F^{-1}$ kâhinlerine ise en çok t sorgu sorabildiğini varsayılır. Saldırgan A'nın FX veya π oyununu kazanması rastgele tahminine göre avantajı en çok $mt \cdot 2^{-K-n+1}$ olarak hesaplanır. Başka bir deyişle, saldırganın avantajı en fazla $t \cdot 2^{-K-n+1+\lg m}$ 'dir, bu nedenle FX'in etkili anahtar uzunluğu anahtar arama saldırısına göre en az K + n – 1 – lg m bittir (Kilian and Rogaway, 2000).

Bu sonucu daha iyi anlamak için "FX veya π" oyununu daha detaylı olarak incelemek gereklidir. "FX veya π" oyununda bulunan anahtar arama saldırganı, üç kâhin, E, F ve $F^{-1}$ erişimine sahip bir algoritma A'dır. Böylece saldırgan A, E(P), $F_k(x)$ veya $F_k^{-1}(y)$ şeklinde sorgular yapabilmektedir. Burada (m, t) genel anahtar arama saldırganı, E kâhinine m sorgularını ve F ve $F^{-1}$ kâhinlerine toplamda t sorguyu yapan bir anahtar arama saldırganıdır. Saldırgan A'nın sorgularının bir parçası olarak k değerini F ve $F^{-1}$ kâhinlerine sağlamaktadır. Oyunda A'nın E, F ve $F^{-1}$ kâhinleri ile etkileşime geçtiğini $A^{E, F, F-1}$ ile gösterilmektedir.

Oyuna başlamadan önce k-bit anahtarları ve n-bit blokları olan rastgele bir blok şifre F seçilmekte ve $F_k \xleftarrow{R} \mathcal{P}_n$ şeklinde gösterilmektedir. Burada $x \xleftarrow{R} S$ ifadesi, x'in S içerisinden rastgele olarak seçilmesini ve $\mathcal{P}_n$ ise n-bit üzerine $(2^n)!$ permütasyon uzayının tamamını göstermektedir. Girdi x üzerine, şifreleme kâhini E, ya rastgele bir (K+2n)-bit anahtar K için $FX_K(x)$ hesaplar ya da rastgele permütasyon $\pi \xleftarrow{R} \mathcal{P}_n$ için π(x) hesaplar. Bundan sonra saldırgan A'dan hangi tür şifreleme yapıldığına dair cevap beklenmektedir. Saldırgan A'nın verdiği cevaplara göre durumlar belirlenmiştir. 1 durumu saldırganın FX ile şifrelenmiş veri ile π arasındaki ayrımı çok iyi yapabildiğini; 0 durumu saldırganın verdiği cevaplar ile ayrım yapamadığını;

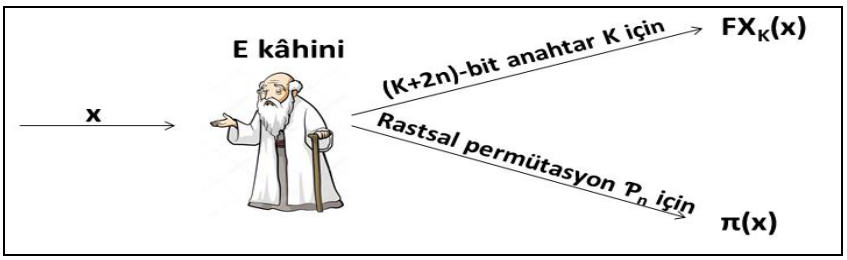
-1 durumu ise saldırganın sürekli doğru ve sahte şifreleme tahminlerini birbirine karıştırdığını göstermektedir (Rogaway, 1996).

k,n ≥ 0 ve tamsayı ve ε ≥ 0 ve gerçel bir sayı olmak üzere,

$$Adv_A = Pr[F \xleftarrow{R} \mathcal{B}_{k,n;} K \xleftarrow{R} \{0,1\}^{K+2n} : A^{FXK,\,F,\,F-1} = 1\,] -$$

$Pr[F \xleftarrow{R} \mathcal{B}_{k,n;}\, \pi \xleftarrow{R} \mathcal{P}_n : A^{\pi,\,F,\,F-1} = 1] \geq \varepsilon$ ise A saldırganın FX şifreleme algoritmasını k, n parametreleri ile ε kırdığı anlaşılır. Burada $Pr[A_1; A_2;\ldots: E]$ ifadesi, E olayının $A_1$, $A_2$, …… eylemlerinden sonra oluşma ihtimalini göstermektedir. Her $k \in \{0,1\}^K$, $F(k,\cdot) \in \mathcal{P}_n$ için $F : \{0,1\}^K \times \{0,1\}^n \rightarrow \{0,1\}^n$ bir blok şifrelemedir. $\mathcal{B}_{k,n}$ ise yukarıda belirtilmiş olan k ve n parametreleri için tüm blok şifreleme uzayını göstermektedir. $F \in \mathcal{B}_{k,n}$ için $FX \in \mathcal{B}_{k+2n,n}$, yani $FX(K,x) = k2\ k_2 \oplus F_k(k_1 \oplus x)$ blok şifreleme olarak tanımlanmıştır. Burada kullanılan anahtar K = k.k1.k2 olarak, |k| = k ve |k1| = |k2| = n olarak verilmiştir (Kilian and Rogaway, 2000). Dolayısıyla anahtar arama analizinde kullanılmış olan FX blok şifreleme modelini AESX ve $AESX^+$ için kullanabilmemiz mümkün olmaktadır. Çünkü k anahtarımız 128-bit uzunluğunda, k1 ve k2 anahtarlarımız ise n-bit blok uzunluğuna eşit olarak 128-bit uzunluğundadır.

A'nın oynayabileceği iki farklı oyun bulunmaktadır. Oyunların daha iyi anlaşılabilmesi için Şekil 5.5'de oyun modeli gösterilmektedir. Birinci oyun saldırganın rastlantısal permütasyon için oynadığı R oyundur (Game R) ve avantaj $P_R = Pr[A^{\pi,F,F-1} = 1]$ şeklinde ifade edilir. R Oyunun adımları Şekil 5.6'de gösterilmektedir. Burada kullanılan $\overline{\text{Range}}$ ve $\overline{\text{Dom}}$ ifadeleri, $\{0,1\}^m$ 'nin bir alt kümesinden $\{0,1\}^n$ 'nin bir alt kümesine kısmen tanımlanmış olan F fonksiyon için, Dom(F), F'nin alanını ve Range(F), F aralığını tanımlamak için kullanılmaktadır. $\overline{\text{Dom}}(F) = \{0,1\}^m$ - Dom(F) olarak ve $\overline{\text{Range}}(F) = \{0,1\}^n$ - Range(F) olarak ifade edilmektedir.




Şekil 5.5 FX-veya-π oyun modeli

İkinci oyun, saldırganın FX için oynadığı X oyundur (Game X) ve avantaj $P_X = Pr[A^{FXk, F, F-1} = 1]$ şeklinde ifade edilir. X Oyunun adımları da Şekil 5.6'da gösterilmektedir. Şekil 5.6'da görüldüğü üzere Oyun X'de, Oyun R gibi hareket edilmektedir. Ancak Oyun X'de farklı olarak FX-cevapları ile $F/F^{-1}$-cevapları arasında bir tutarsızlık oluşmaktadır ve X oyuncusu bu tutarsızlıkların oluşup oluşmadığını kontrol etmektedir. Oluşturulmaya çalışılan bir tutarsızlık bulursa tutarlılığı zorlamak için cevabını değiştirir. Oyun X bunu yapmaya çalıştığında "bad" bayrağını kurmaktadır.

**Oyun R**

Başlangıçta, F ve E'nin tanımsız olmasına izin ver. "Bad" bayrağı başlangıçta kurulmadı.

Rastgele $k^* \xleftarrow{R} \{0,1\}^k$, $k_1^*, k_2^* \xleftarrow{R} \{0,1\}^n$ seç. Sonra saldırganın yaptığı her soruyu aşağıdaki gibi cevapla:

(E) Kâhine E(P) sorgusu:

1. $\overline{\text{Range}(E)}$ 'den üniform olarak $C \in \{0,1\}^n$ 'yi seçin.
2. Eğer $F_{k^*}(P \oplus k_{1^*})$ tanımlandıysa "bad" bayrağı kurulur.
   Eğer $F_{k^*}^{-1}(C \oplus k_{2^*})$ tanımlandıysa "bad" bayrağı kurulur.
3. E(P) = C tanımlandıysa C'yi döndür.

(F) Kâhine $F_k(x)$ sorgusu:

1. $\overline{\text{Range}(F_k)}$ 'den üniform olarak $y \in \{0,1\}^n$ 'yi seçin.
2. Eğer $k = k^*$ ve $E(x \oplus k_{1^*})$ tanımlandıysa "bad" bayrağı kurulur.
   Eğer $k = k^*$ ve $E^{-1}(y \oplus k_{2^*})$ tanımlandıysa "bad" bayrağı kurulur.
3. $F_k(x) = y$ tanımlandıysa y'yi döndür.

($F^{-1}$) Kâhine $F_k^{-1}(y)$ sorgusu:

1. $\overline{\text{Dom}(F_k)}$ 'den üniform olarak $x \in \{0,1\}^n$ 'yi seçin.
2. Eğer $k = k^*$ ve $E^{-1}(y \oplus k_{2^*})$ tanımlandıysa "bad" bayrağı kurulur.
   Eğer $k = k^*$ ve $E(x \oplus k_{1^*})$ tanımlandıysa "bad" bayrağı kurulur.
3. $F_k(x) = y$ tanımlandıysa x'i döndür.

**Oyun X**

Başlangıçta, F ve E'nin tanımsız olmasına izin ver. "Bad" bayrağı başlangıçta kurulmadı.

Rastgele $k^* \xleftarrow{R} \{0,1\}^k$, $k_1^*, k_2^* \xleftarrow{R} \{0,1\}^n$ seç. Sonra saldırganın yaptığı her soruyu aşağıdaki gibi cevapla:

(E) Kâhine E(P) sorgusu:

1. $\overline{\text{Range}(E)}$ 'den üniform olarak $C \in \{0,1\}^n$ 'yi seçin.
2. Eğer $F_{k^*}(P \oplus k_{1^*})$ tanımlandıysa $C \leftarrow F_{k^*}(P \oplus k_{1^*}) \oplus k_{2^*}$ yapılır ve "bad" bayrağı kurulur.
   Değilse eğer $F_{k^*}^{-1}(C \oplus k_{2^*})$ tanımlandıysa "bad" bayrağı kurulur ve 1. Adıma gidilir.
3. E(P) = C tanımlandıysa C'yi döndür.

(F) Kâhine $F_k(x)$ sorgusu:

1. $\overline{\text{Range}(F_k)}$ 'den üniform olarak $y \in \{0,1\}^n$ 'yi seçin.
2. Eğer $k = k^*$ ve $E(x \oplus k_{1^*})$ tanımlandıysa $y \leftarrow E(P \oplus k_{1^*}) \oplus k_{2^*}$ yapılır ve "bad" bayrağı kurulur.
   Değilse eğer $k = k^*$ ve $E^{-1}(y \oplus k_{2^*})$ tanımlandıysa "bad" bayrağı kurulur ve 1. Adıma gidilir.
3. $F_k(x) = y$ tanımlandıysa y'yi döndür.

($F^{-1}$) Kâhine $F_k^{-1}(y)$ sorgusu:

1. $\overline{\text{Dom}(F_k)}$ 'den üniform olarak $x \in \{0,1\}^n$ 'yi seçin.
2. Eğer $k = k^*$ ve $E^{-1}(y \oplus k_{2^*})$ tanımlandıysa $x \leftarrow E^{-1}(y \oplus k_{2^*}) \oplus k_{1^*}$ yapılır ve "bad" bayrağı kurulur.
   Değilse eğer $k = k^*$ ve $E(x \oplus k_{1^*})$ tanımlandıysa "bad" bayrağı kurulur ve 1. Adıma gidilir.
3. $F_k(x) = y$ tanımlandıysa x'i döndür.

Şekil 5.6 Oyun R ve X (Kilian and Rogaway, 2000)

Anahtar arama saldırısını analiz edebilmek için, saldırganın tahmin ve sorgu karmaşıklığını bir üst değer ile bağlamamız gereklidir. Bağlayacağımız bu değer aynı zamanda A'nın avantajını ortaya koyacaktır. Her iki oyunda da ortak nokta "bad" bayrağının kurulması olduğu için saldırganın avantajı $Pr_R[BAD]$ ile yani $Adv_A \leq Pr_R[BAD]$ olarak bağlanabilmektedir. Kilian ve Rogaway (2000) bu ifadeyi sağlayabilmek için aşağıdaki işlemleri kullanmışlardır;

$$Adv_A = P_X - P_R$$

$$= Pr_X[A^{E,F,F^{-1}} = 1] - Pr_R[A^{E,F,F^{-1}} = 1]$$

$$= Pr_X[A = 1|\overline{BAD}]\, Pr_X[\overline{BAD}] + Pr_X[A = 1|BAD]\, Pr_X[BAD] -$$

$$Pr_R[A = 1|\overline{BAD}]\, Pr_R[\overline{BAD}] + Pr_R[A = 1|BAD]\, Pr_R[BAD]$$

$$= Pr_R[BAD]\,(Pr_X[A = 1|BAD] - Pr_R[A = 1|BAD])$$

$$\leq Pr_R[BAD]$$

Killian ve Rogaway (1996) R oyununu değiştirerek üçüncü bir oyun (Game R') tasarlamışlardır. R' oyununun adımları Şekil 5.7'de gösterilmektedir.

Başlangıçta, F ve E'nin tanımsız olmasına izin verin. Saldırganın yaptığı her soruyu şu şekilde cevaplayın:

(E) Kâhine E(P) sorgusu:

1. $\overline{Range}(E)$ 'den üniform olarak C'yi seçin.
2. E(P) = C tanımlandıysa C'yi döndür.

(F) Kâhine $F_k(x)$ sorgusu:

1. $\overline{Range}(F_k)$ 'den üniform olarak 'yi seçin.
2. $F_k(x) = y$ tanımlandıysa y'yi döndür.

($F^{-1}$) Kâhine $F_k^{-1}(y)$ sorgusu:

1. $\overline{Dom}(F_k)$ 'den üniform olarak x'i seçin.
2. $F_k(x) = y$ tanımlandıysa x'i döndür.

Tüm sorgular cevapladıktan sonra:
Bad bayrak başlangıçta kurulmadı.
$k^* \xleftarrow{R} \{0,1\}^k$, $k1^*, k2^* \xleftarrow{R} \{0,1\}^n$ anahtarlarını rastgele seç.
Bazı x değerleri için $F_{k^*}(x)$ ve $E(x \oplus k_1^*)$ birlikte tanımlıysa kötü bayrağı kurulur.
Bazı y değerleri için $F_{k^*}^{-1}(y)$ ve $E\text{-}1(y \oplus k_2^*)$ birlikte tanımlıysa kötü bayrağı kurulur.

Şekil 5.7 Oyun R' (Kilian and Rogaway, 2000)

Bu oyunda k*, k1* ve k2* anahtarları başlangıç yerine sonda

seçilmektedir. R' oyununun sonunda, eğer herhangi bir x değeri için, $F_k^*(x)$ ve $E(x \oplus k_1^*)$ birlikte tanımlanmışsa ve eğer herhangi bir y değeri için $F_{k^*}^{-1}(y)$ ve $E^{-1}(y \oplus k_2^*)$ birlikte tanımlanmışsa "bad" bayrağı kurulur. R' oyununda da "bad" bayrağının kurulması diğer oyunlarla aynı olduğu için PrR[BAD] değeri $Pr_{R'}$[BAD] değerine eşittir. Böylece üçüncü oyun içinde saldırganın avantajını bağlama hesabı yapılabilmektedir. R' oyunu için "bad" bayrağının kurulması, $2^{k+2n}$ adet k*, k1* ve k2* anahtar seçimi sonucu oluşmaktadır.

Killian ve Rogaway (1996), R' oyununda E(P) değerlerini gösteren, |E| değerini m değerine, $F_k(x)$ değerini gösteren, |F| değerini de t değerine sabitlemişlerdir. Bu oyunda her bir $P \oplus k1^* = x$ veya $C \oplus k2^* = y$ değerleri, normalde $y = Fk(x)$ ve $C = E(P)$ olduğu için, bu değerler ile çarpışmaya neden olmaktadır. Ancak herbir "bad" bayrağını kuran k*, k1* ve k2* anahtarları seçimi çarpışmaya neden olduğu için, çarpışmaya neden olan k*, k1* ve k2* anahtarlarının sayısını bir üst değer ile bağlayabilmek mümkün olmaktadır. $k^*, k1^*, k2^* \in \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^n$ olduğundan dolayı çarpışmaya neden olabilecek maksimum $2mt \cdot 2^n$ adet k*, k1* ve k2* anahtarı bulunmaktadır.

Sonuç olarak üçüncü oyunda, E ve F'den bağımsız olarak, üçlü k*, k1* ve k2* anahtarları rastlantısal seçilmektedir ve E ve F değeri ne seçilirse seçilsin, çarpışmaya neden olan bu anahtarları seçme olasılığı en fazla $2mt \cdot 2^n / 2^{k+2n} = mt \cdot 2^{-k-n+1}$ olmaktadır. Bu olasılık $Adv_A$ değerini bağlamımızı sağlamaktadır (Kilian and Rogaway, 2000).

Saldırganın "FX-veya-π" oyununu kazanmak için sahip olabileceği maksimum avantaj $mt.2^{-k-n+1}$ olarak karşımıza çıkmaktadır. Saldırganın sahip olduğu maksimum avantajı $t.2^{-k-n+1+\lg m}$ olarak ifade edersek, FX için etkili anahtar uzunluğu en az k+n-1-lg m bit olmaktadır. Bu ifadede belirtilmiş olan k değeri, F'nin anahtar uzunluğunu, n değeri blok uzunluğunu ve m değeri saldırganın sahip olabileceği düz veri ile şifreli veri çifti sayısını ifade etmektedir.

Killian ve Rogaway (1996)'e göre, giriş ve çıkış xor işlemlerinde aynı anahtar kullanılan model $FX'k.k1(x) = k1 \oplus Fk(x \oplus k1)$ ve $FXk.k1.k2(x) = k2 \oplus Fk(k1 \oplus x)$ arasında genel anahtar arama saldırısına karşı güvenlik olarak bir kayıp olmadığı belirtilmiştir. Anahtar arama saldırganın belirtilen $FX'k.k1(x) = k1 \oplus Fk(x \oplus k1)$ modelini kırmak için avantajı en çok $mt \cdot 2^{-K-n+1}$ olur. Burada düz veri önce k1 anahtarı ile xor işlemine tabi olup, F

rastlantısal permütasyonundan geçtikten sonra tekrar k1 anahtarı ile xor işlemi yapıldığı için "FX-veya-π" oyunu bu modelde de geçerli olacaktır. Önerilen model AESX$^+$ şifreleme algoritmasında, çıkışta bulunan anahtar, k ve k1 xor işleminden elde edilen bir anahtar olduğu için, anahtar arama saldırısına göre AESX ile etkili anahtar uzunluğu aynı olacaktır. Dolayısıyla AESX ve AESX$^+$ şifreleme algoritmaları arasında genel anahtar arama saldırısı açısından güvenlikleri arasında bir fark bulunmamaktadır.

AESX ve AESX$^+$ şifreleme algoritmaları için n-bit blok uzunluğunu 128 bit, k-bit anahtar uzunluğunu 128 bit olarak ele alırsak etkili anahtar uzunluğu $128 + 128 – 1 – \lg m = 255 – \lg m$ bit olarak elde edilmektedir.

### 5.2.2 AESX ve AESX$^+$ Anahtar Arama Analiz Sonuçları

AES-128, AES-256, AESX ve AESX$^+$ anahtar arama analiz sonuçları Çizelge 5.2'de gösterilmiştir. Çizelge 5.2'de görüldüğü üzere AESX ve AESX$^+$ şifreleme algoritmaları anahtar arama saldırısına karşı AES-128 şifreleme algoritmasından daha dirençlidir. AESX ve AESX$^+$ şifreleme algoritmaları genel anahtar arama saldırısına karşı yaklaşık olarak AES-256 şifreleme algoritması kadar güvenlik sağlamaktadır.

Çizelge 5.2 AES-128, AES-256, AESX ve AESX$^+$ anahtar arama analiz sonuçları

| Algoritma | Etkili anahtar uzunluğu |
|---|---|
| AES-128 | 128 bit |
| AESX | 255−lg m bit |
| AESX$^+$ | 255−lg m bit |
| AES-256 | 256 bit |

## 5.3 İlişkili Anahtar Saldırısı

Bir kripto sisteme saldırı yapabilmek için saldırganın sahip olabileceği

bazı veriler vardır. Saldırgan, saldırı yapacağı kripto sistemin, sadece şifreli veri dizisine veya düz veri dizisi ile birlikte bunların şifreli veri dizisine veya şifreli veri dizisini seçerek bunların düz verilerini oluşturmak suretiyle verilerine ulaşabilir. İlişkili anahtar saldırısında ise saldırgan farklı anahtarlar ile şifrelenmiş düz veri dizisini gözlemlemektedir.

İlişkili anahtar saldırısı, saldırganın başlangıçta değerlerini bilmediği fakat matematiksel olarak ilişkili olduklarını bildiği birden fazla anahtarın şifreli veri üzerinde etkilerini inceleyerek yaptığı bir kriptanaliz türüdür. Bu saldırı türü ilk olarak Biham (1994) tarafından ifade edilmiştir. Bu saldırı türünün başarılı olabilmesi için saldırganın bulmaya çalıştığı anahtarlar arasındaki ilişkiyi bilmesi gereklidir.

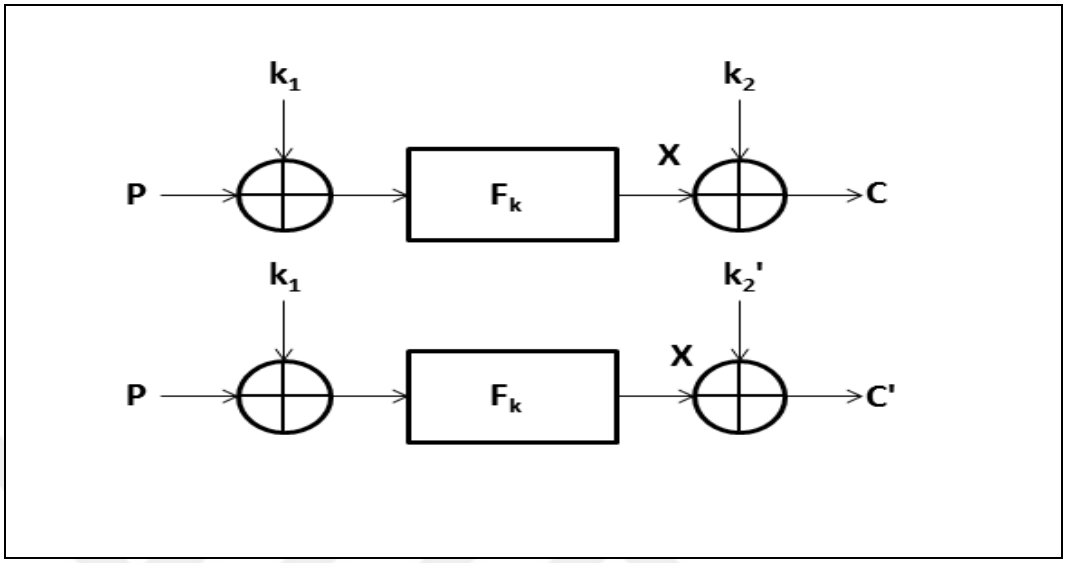
İlişkili anahtar saldırısında, blok şifreleme anahtarlı permütasyon $E : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ olmak zorundadır. Diğer saldırı türleri genellikle anahtarla şifrelenmiş verilerin girdi ve çıktılarıyla ilgilenirken ilişkili anahtar saldırısı anahtarlar arasındaki ilişkiyi kontrol etmekle ilgilenmektedir. Diğer saldırı türlerinde saldırgan tek bir anahtar için tahmin yaparken, ilişkili anahtar saldırısında saldırgan birden fazla anahtar için tahmin yapmaktadır. Saldırgan anahtarlar arasındaki ilişkiyi seçebilir ya da anahtarlar arasındaki ilişkiyi bilmektedir.

İlişkili anahtar saldırısının, bazı araştırmacılar tarafından teoride kalacağı ve pratikte uygulanmasının çok zor olacağı düşünülmektedir. Fakat bazı araştırmacıların son çalışmaları bu saldırının da pratikte uygulanabileceğini göstermiştir. İlişkili anahtar saldırılarının, karma (hash) fonksiyonlarda ve anahtar değişim protokolleri gibi bazı kriptografik uygulamalarda pratikte uygulanabilecek örnekleri bulunmaktadır (Phan, 2004).

### 5.3.1 AESX ve AESX$^+$ İlişkili Anahtar Analizi

AESX şifreleme algoritmasına ilişkili anahtar saldırısı yapmak için, DESX için uygulanmış olan ve anahtar beyazlatma tekniğini kullanmış olan tüm kripto sistemlere uygulanabilir olan ilişkili anahtar saldırısı kullanılacaktır (Phan and Shamir, 2008). AESX şifreleme algoritmasını $FX_{k.k1.k2}(x) = k_2 \oplus F_k(k_1 \oplus x)$ şeklinde ifade edersek ve düz verinin $k_2$ ve $k_2' = (k_2 \pm D) \bmod 2^n$ ile şifrelenmesi ile $C = k_2 \oplus F_k(k_1 \oplus x)$ ve $C' = k_2' \oplus F_k(k_1 \oplus$

x ) şifreli verileri elde edilir. Burada $F_k(k_1 \oplus x)$ yerine X koyarsak sonuç $C = k_2 \oplus X$ ve $C' = k_2' \oplus X$ şeklinde olacaktır. C ve C' xor işlemine tabi tutulursa $C \oplus C' = k_2 \oplus k_2' = k_2 \oplus (k_2 \pm D) \bmod 2^n$ sonucu elde edilir. Şekil 5.8'de görüldüğü üzere çıkış bloğundaki xor işlemine kadar iki şifreleme de aynıdır.




Şekil 5.8 İlişkili anahtar saldırısı

Burada iki anahtar arasındaki fark (D) ve C xor C' eşitliği biliniyorsa $k_2$ anahtarının tüm muhtemel değerleri hesaplanabilmektedir. Saldırının faklı düz veri ile veya farklı $k_2$ - $k_2'$ farkı ile üç kere tekrarlanması ile $k_2$ anahtarı için biri gerçek değeri olan iki muhtemel değer elde edilmektedir.

Saldırının üç kere tekrarlanmasının sebebi AESX şifreleme algoritmasının 384 bit uzunluğunda anahtar kullanıyor olmasıdır. AESX şifreleme algoritmasında belirli bir P düz verisi için 384 bitlik anahtar $2^{384} / 2^{128} = 2^{256}$ adet belirli C şifreli verisi oluşturacaktır. Yani elde edilecek olan ilk düz veri - şifreli veri (P,C) çiftinde, $2^{256}$ yanlış alarm üretilecektir. Yanlış alarm, $2^{384} / 2^{128x3} = 1$ oranına düşürüldükten sonra düz veri – şifreli veri (P,C) çiftinin doğru olduğundan emin olabiliriz (Stallings, 1999).

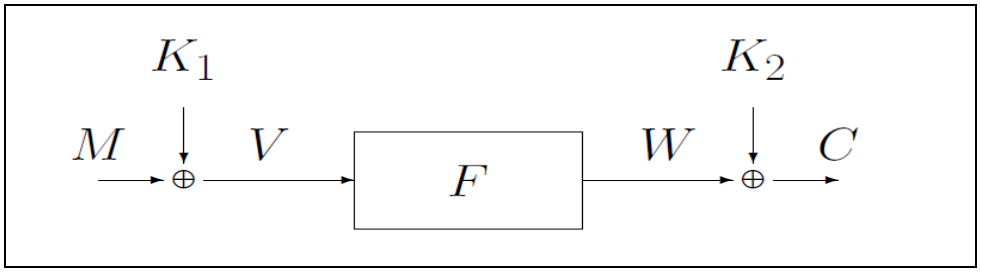
Böylelikle, 3 çift ilişkili anahtar ve bilinen düz veri dizisi yani 6 adet ilişkili anahtar – bilinen düz veri dizisi ile $k_2$ anahtarı ele geçirilebilir. Geriye $2^{128}$ AES şifrelemesi ve $2^{128}$ hafıza ile ortadaki adam saldırısı yapılabilmektedir. Ters yönden bilinen şifreli veri dizisi ile $k_1$ anahtarı elde edilebilmektedir. $k_1$ anahtarı da elde edildikten sonra, $2^{128}$ AES şifreleme için anahtar arama saldırısı yapılarak k anahtarı ele geçirilebilmektedir.

Dolayısıyla, AESX şifreleme algoritmasına, ilişkili anahtar saldırısı, ters yönden yapılan 3 çift ilişkili anahtar da hesaplanırsa, 6 x 2 = 12 ≈ $2^{3.5}$ ilişkili anahtar bilinen düz/şifreli veri dizisi ve $2^{128}$ AES şifreleme algoritmasına anahtar arama saldırısı ile yapılabilmektedir.

Phan ve Shamir (2008) ilişkili anahtar saldırısının AESX$^{+}$' a uygulanması bir açık metin P'nin Şekil 5.8'deki $k_1$ ve k sabit iken $k_2$ ve $k_2$' anahtarları ($k_2' = (k_2 \pm D) \bmod 2^n$) altındaki şifrelenmiş metinler C ve C' üzerinden olmaktadır. Burada $C \oplus C' = k_2 \oplus k_2' = k_2 \oplus (k_2 \pm D) \bmod 2^n$ olup anılan saldırı farklı D değerleri seçilerek k2 anahtar uzayının küçültülmesi ve saldırının 3 kez daha farklı açık metinler Ps ve anahtar farkları Ds'ler kullanılarak $k_2$ için sadece tek geçerli değer bırakılması esasına dayanmaktadır.

Oysa AESX$^{+}$ yapısında $k_2 = k \oplus k_1$ olup $k_1$ ve k sabit iken $k_2$ de sabit olmaktadır. Bir başka ifade ile $C \oplus C' = k_2 \oplus k_2' = k_2 \oplus k_2 = 0$ olup $C \oplus C'$ için sadece sıfır değeri mümkün olduğundan farklı $k_2$' veya P' değerleri seçerek $k_2$ anahtar uzayını küçültmek mümkün değildir. Dolayısıyla ilişkili anahtar saldırısının bu versiyonu AESX$^{+}$' a uygulanamaz.

AESX$^{+}$ şifreleme algoritmasına ilişkili anahtar saldırısını analiz edebilmek için Daemen (1992)'in Even-Mansour blok şifreleme yapısı için kullanmış olduğu bilinen metin saldırısı ve seçilmiş metin saldırıları analiz edilebilir. Even-Mansour blok şifreleme yapısı, bilinen bir F permütasyonu ile n-bit blok uzunluğunda $X \in \{0,1\}^n$ girdiyi F(X) ve $F^{-1}(X)$ hesaplamalarını yapabilen bir şifrelemedir. Şekil 5.9'da gösterildiği üzere M mesajı ile ve C şifreli verisi arasında ilişki $C \oplus K_2 = F(M \oplus K_1)$ eşitliğidir. Even-Mansour blok şifreleme yapısında görüldüğü üzere F permütasyonu yerine AES şifreleme algoritması kullanılabilmektedir. Böylelikle Even-Mansour blok şifreleme yapısı AESX ve AESX$^{+}$ şifreleme algoritmalarının analizlerini yapmak üzere kullanılabilmektedir.




Şekil 5.9 Even-Mansour blok şifreleme yapısı (Daemen, 1992)

Saldırganın, (Ma, Ca) ve (Mb, Cb) gibi iki bilinen düz-şifreli veri çiftini bildiğini varsayalım. Xa ve Xb bloklarının bitler halinde xor işlemine tabi tutulması $Xa \oplus Xb = \Delta X$ ile gösterilir. Saldırı, Şekil 5.6'da gösterilen tüm V değerleri için $\Delta W = F(V) \oplus F(V \oplus \Delta M)$ hesaplamasını içerir. Eğer V değeri $V = Ma \oplus K_1$ veya $V = Mb \oplus K_1$ ise $\Delta W = \Delta C$ eşitliği ortaya çıkar. Eğer F değeri rastlantısal ise sadece birkaç V değeri için bu eşitlik ortaya çıkacaktır. Yanlış V değerleri, $F(V') \neq Ma \oplus K_2$ ve $F(V') \neq Mb \oplus K_2$ eşitsizlikleri ile kontrol edilerek çıkarılır ve doğru V değerleri bulunduğunda iki anahtar kolaylıkla hesaplanabilmektedir. Bu saldırıda F üzerine ortalama $2^{n-1}$ deneme yapılmaktadır ve böylelikle etkili anahtar uzunluğu n-bit olmaktadır. Bu analizi $AESX^+$ şifreleme algoritması üzerinde yaptığımızda, saldırganın iki adet düz-şifreli veri çiftine ve 128 bit bir anahtar ile F için kullanılan AES şifeleme algoritmasının kullandığı anahtara da ihtiyacı olduğunu varsayalım. Böylelikle bilinen metin saldırısı için, 2 adet bilinen düz veri ile $2^{128}$ F fonksiyonu (AES şifreleme) için olmak üzere ve $2^{128}$ etkili anahtar uzunluğu (Even-Mansour blok yapısı için) olmak üzere toplamda $2^{256}$ adet şifreleme çözme işlemine ihtiyaç bulunmaktadır.

Seçilmiş düz veri saldırısında ise saldırgan k adet Mi ve Mi' düz veri çiftini, bir sabit olan her i değeri için, $Mi \oplus Mi' = \Delta M$ olacak şekilde seçer. Saldırgan bu düz verilere bağlı olarak Ci ve Ci' şifreli verilerini elde eder. Saldırgan, $k << 2^n$ için k adet farklı $\Delta Ci$ değeri elde eder ve bu değerler için $\Delta W = \Delta Ci$ eşitliğini hesaplar. Eğer k değeri için bir hafıza kısıtlaması yoksa ideal k değeri $2^{n/2}$ civarındadır. Bu durumda F hesaplamalarının sayısı $2^{n/2}$ çıkar ve etkili anahtar uzunluğunu n/2 bit kadar kısaltır. Bu analizi de $AESX^+$ şifreleme algoritması üzerinde yaptığımızda, saldırganın düz-şifreli veri çifti olarak ideal k sayısı olan $2^{n/2}$ için $2^{64}$ adet düz-şifreli veri çifti ve 128 bit bir anahtar ile F fonksiyonu için kullanılan AES şifeleme algoritmasının kullandığı anahtara da ihtiyacı olduğunu varsayalım. Böylelikle seçilmiş veri saldırısı için, saldırganın $2^{64}$ adet düz-şifreli veri çifti ile $2^{64}$ Even-Mansour blok yapısının etkili anahtar uzunluğu olmak üzere ve $2^{128}$ F fonksiyonu (AES şifreleme) için kullanılan anahtar ile birlikte toplamda $2^{192}$ adet şifreleme çözme işlemine ihtiyaç bulunmaktadır.

### 5.3.2 AESX ve $AESX^+$ İlişkili Anahtar Analiz Sonuçları

AES-128, AES-256, AESX ve $AESX^+$ ilişkili anahtar analiz sonuçları Çizelge 5.3'de gösterilmiştir. Çizelge 5.3'de görüldüğü üzere AESX şifreleme

algoritması Phan ve Shamir (2008)'in ilişkili anahtar saldırısına karşı AES-128 şifreleme algoritmasına göre daha dirençlidir. Önerilen model $AESX^+$ ve AESX şifreleme algoritmaları diğer ilişkili anahtar saldırılarına karşı AES-128 şifreleme algoritmasına göre daha dirençlidir.

Çizelge 5.3 AES-128, AES-256, AESX ve $AESX^+$ ilişkili anahtar analiz sonuçları

| Algoritma | Veri İhtiyacı | AES Şifreleme |
|---|---|---|
| AES-128 | - | $2^{128}$ |
| AESX | $2^{3.5}$ ilişkili anahtar-bilinen metin | $2^{128}$ |
| $AESX^+$ (Bilinen Metin Saldırısı) | 2 bilinen metin | $2^{256}$ |
| $AESX^+$ (Seçilmiş Metin Saldırısı) | $2^{64}$ seçilmiş metin | $2^{192}$ |
| AES-256 | - | $2^{256}$ |

## 5.4 Anahtar Beyazlatma Tekniğinin Güvenliği

Modern blok şifreleme algoritmaları yapısı yinelenen tur fonksiyonlarına dayalıdır. Bu tur fonksiyonları ise genellikle, bir alt anahtar ekleme kısmı ve ardından anahtarsız tersinir bir fonksiyon içermektedir. Bu fonksiyonlar AES ve DES şifreleme algoritmaları da dâhil olmak üzere çoğu blok şifreleme algoritmaları tarafından kullanılmaktadır.

Blok şifreleme algoritmalarında anahtar üretme fonksiyonları güvenlik açısından önemlidir. Burada üretilen anahtarların ilişkisiz olması ve birbirinden bağımsız olarak rastgele seçilmesi gereklidir. Ancak her ne kadar anahtar üretme fonksiyonları kripto sistemleri güvenli hale getirse de çoklu anahtar saldırılarına karşı etkisizdir (Mouha and Luykx, 2015).

Mouha ve Luykx (2015), bir anahtar beyazlatma tekniği olan tek anahtarlı Even-Mansour yapısının blok uzunluğu karşılaştırıldığı ideal blok şifreleme yapısının anahtarı ile aynı uzunlukta ise, ideal bir blok şifreleme yapısıyla aynı güvenliği sağladığını ispatlamışlardır.

Mouha ve Luykx (2015), çoklu anahtar saldırısına karşı tek anahtarlı Even-Mansour yapısını $E_K(P) = \pi(P \oplus K) \oplus K$ şeklinde tanımlamışlardır. Bu yapının güvenliğini ise $Adv_A \leq (D^2 + 2DT) / 2^n$ olarak ispatlamışlardır. Burada D, saldırganın $E_{K1}$, …, $E_{K\ell}$ $\ell$ sayısı kadar anahtar ile şifrelenmiş olan verilere yaptığı sorguları, T ise $\pi$ veya $\pi^{-1}$ fonksiyonuna yaptığı sorguları göstermektedir. n ise Even-Mansour yapısının blok uzunluğunu göstermektedir. İdeal bir blok şifreleme yapısının çoklu anahtar saldırısına karşı güvenliği ise $Adv_A \leq (D^2 + 2DT) / 2^{k+1}$ olarak ispatlamışlardır. Burada her bir anahtar için olan düz verinin sayısı az ise ve Even-Mansour yapısının blok uzunluğu ideal blok şifreleme yapısının anahtar uzunluğuna eşit ise sağladıkları güvenlik çok yakındır. Burada tek anahtarlı Even-Mansour yapısı kullanılmış olsa da Mouha ve Luykx (2015) çok anahtarlı Even-Mansour yapısının da aynı güvenliği sağladığını belirtmişlerdir.

Even-Mansour yapısının kullanıldığı blok şifrelemeye birçok faydası bulunmaktadır. Öncelikle kullanıldığı blok şifrelemede tur anahtarlarının tekrar hesaplanmalarına ve hafızada tutulmalarına gerek yoktur. Anahtar üretme fonksiyonunda tekrar hesaplamanın kaldırılması anahtar üretimini hızlandırmakta ve tekrar anahtar üretme maliyetinden kurtarmaktadır. Bu durum blok şifrelemenin etkinliğini artırmakta ve hafıza ihtiyacını düşürmektedir. Ayrıca kullanıldığı blok şifrelemede tur anahtarlarının güvenli bir şekilde tutulmasına da gerek yoktur. AES-128 şifreleme algoritmasında tur anahtarlarından birinin ele geçirilmesi tüm anahtarı ele geçirilmesine neden olacağı düşünüldüğünde bu durum güvenlik açısından oldukça önemlidir (Mouha and Luykx, 2015).

# ALTINCI BÖLÜM

## 6.SONUÇ VE TARTIŞMA

Bu tez çalışmasında AESX, AESX$^+$, AES-128 ve AES-256 şifreleme algoritmalarının performans ve güvenlik açısından karşılaştırılması incelenmiştir.

AES şifreleme algoritması, DES şifreleme algoritmasının güvenirliliğini kaybetmesi üzerine 2000 yılından bu yana DES şifreleme algoritmasının yerini almış ve kullanımı yaygın olan popüler bir blok şifreleme algoritmasıdır. AESX ve önerilen model AESX$^+$ şifreleme algoritmaları AES blok şifreleme algoritmasının anahtar beyazlatma tekniği uygulanmış halidir.

AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının AES-128 ve AES-256 şifreleme algoritmalarına göre performans karşılaştırması Crypto++ açık kaynak kütüphanesinde bulunan API'de mevcut olan sınıflar kullanılarak Linux Mint 17.3 Rosa Xfce İşletim sistemi üzerinde GCC derleyicisi kullanılarak yapılmıştır. Elde edilen sonuçlar AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının, AES-256 şifreleme algoritmasına göre daha hızlı olduklarını göstermektedir. AESX ve önerilen model AESX$^+$ şifreleme algoritmaları, AES-128 şifreleme algoritmasına, AES-256 şifreleme algoritmasının getirdiği ek yüke göre neredeyse yarıya yakın oranında ek yük getirmektedir.

AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının AES-128 şifreleme algoritmasına getirdiği ek yük ve Crypto++ açık kaynak kütüphanesi test sonuçlarında, DESX şifreleme algoritmasının DES şifreleme algoritmasına getirdiği ek yük sonuçları birbirine yakındır. Bu sonuçlara göre AES blok şifreleme algoritmasına uygulanmış olan anahtar beyazlatma işleminin Crypto++ açık kaynak kütüphanesinde DES blok şifreleme algoritmasına uygulanmış olan beyazlatma işlemi ile aynı olduğu anlaşılabilmektedir.

Bu tez çalışmasında AES-128 şifreleme algoritmasının şifreleme hızı AES-256 şifreleme algoritmasına göre %20 oranında daha hızlı olduğu sonuçlar elde edilmiştir. Ancak, AES-128 şifreleme algoritmasının, AES-256 şifreleme algoritmasına göre %40 oranında daha hızlı olması beklenir. Açık

kaynak kripto kütüphanelerinin sonuçları %40 civarındadır. Bu farklılık, testlerde kullanılan Crypto++ kütüphanesinin mevcut sınıflarının Unix sistemlerde kullanmış olduğu komut zinciri (pipelining) neden olmaktadır. Crypto++ kütüphanesi kullanılarak elde edilmiş olan benzer sonuçlar bulunmaktadır (Donta, 2007).

Yapılan test sonuçları ile AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının AES-256 şifreleme algoritmasına göre daha hızlı oldukları ispatlanmıştır. AES şifreleme algoritmasına ek olarak, maliyeti az olan, sadece iki adet xor işlemi yapıldığı için AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının AES-256 şifreleme algoritmasına göre daha hızlı olmaları beklenmektedir.

AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının AES-128 ve AES-256 şifreleme algoritmalarına göre güvenlik karşılaştırmaları ise anahtar arama saldırısı ve ilişkili anahtar saldırılarına göre dirençleri analiz edilerek yapılmıştır.

AESX ve AESX$^+$ şifreleme algoritmalarına yapılan anahtar arama saldırılarında AES şifreleme algoritması kara kutu gibi düşünülmüştür. Ancak AES şifreleme algoritmasının yapısal özelliklerini kullanan bir makine olabilir ve bu makinenin AES şifreleme algoritmasını kara kutu olarak kabul etmesi beklenemez. Ancak anahtar arama saldırısı için geçmişte önerilmiş olan makineler şifreleme kısmını kara kutu olarak kabul etmişlerdir.

AESX, AESX$^+$, AES-128 ve AES-256 şifreleme algoritmalarının anahtar arama saldırısına karşı analizleri Killian ve Rogaway'in "FX-veya-$\pi$" oyununa göre yapılmıştır. Elde edilen sonuçlara göre AESX ve önerilen model AESX$^+$ şifreleme algoritmalarının AES-128 şifreleme algoritmasına göre daha dirençli oldukları ve etkili anahtar uzunluklarının AES-256 şifreleme algoritmasının anahtar uzunluğuna yakın olduğu görülmektedir.

AESX, AESX$^+$, AES-128 ve AES-256 şifreleme algoritmalarının ilişkili anahtar saldırısına karşı analizleri ise Phan ve Shamir'in DESX şifreleme algoritması için uyguladıkları ve anahtar beyazlatma tekniğini kullanmış olan tüm kripto sistemlere uygulanabilir olan ilişkili anahtar saldırısı kullanılmıştır.

Mouha ve Luykx (2015) Even-Mansour yapısında AESX$^+$'da olduğu gibi anahtarlar arasında ilişki bulunmasının sakıncalı olduğunu belirtmişlerdir. Ancak anahtar kullanımının kısıtlı olduğu ortamlarda Even-Mansour

yapısında girişte ve çıkışta aynı anahtarı kullanmak yerine ikinci anahtarı üretmek için tercih edilebilir.

Sonuç olarak AESX ve önerilen model AESX$^{+}$ şifreleme algoritmaları AES-128 şifreleme algoritmasına göre etkili anahtar uzunluğu açısından daha güvenli ve AES-256 şifreleme algoritmasına göre daha hızlı şifreleme yapabilmektedir.

AESX, AESX$^{+}$, AES-128 ve AES-256 şifreleme algoritmalarının performans karşılaştırmaları Inline-Assembly programlama ile yapılması daha doğru sonuçlar verebilir.

AESX, AESX$^{+}$, AES-128 ve AES-256 şifreleme algoritmalarının güvenlik karşılaştırmaları için lineer ve diferansiyel saldırı türleri ile analiz yapılabilir. Bu saldırı türlerinden başka AES şifreleme algoritmasının içyapı özelliklerini kullanan saldırı türleri ile analiz yapılabilir.

# KAYNAKLAR DİZİNİ

**Arrag, S., Hamdoun, A., Tragha, A. and Khamlich, S.,** 2012, Several AES Variants under VHDL language In FPGA, International Journal of Computer Science Issues ,9(5):135-141 pp.

**Biham E.**, 1994, New Types of Cryptanalytic Attacks Using Related Keys. In: Helleseth T. (eds) Advances in Cryptology — EUROCRYPT ’93. EUROCRYPT 1993. Lecture Notes in Computer Science, vol 765. Springer, Berlin, Heidelberg

**Cryptopp.com**, “Crypto++ benchmarks” https://www.cryptopp.com/benchmarks.html (Erişim Tarihi: 13 Aralık 2016)

**Daemen, J.,** 1992, Limitations of the Even-Mansour construction. In: Imai H., Rivest R.L., Matsumoto T. (eds) Advances in Cryptology — ASIACRYPT '91. ASIACRYPT 1991. Lecture Notes in Computer Science, vol 739. Springer, Berlin, Heidelberg.

**Donta, P. K**., 2007, Performance Analysıs Of Securıty Protocols, MSc Thesis, University Of North Florida School Of Computing, 120 p (unpublished).

**Even, S. and Mansour, Y.,** 1992, A construction of a cipher from a single pseudorandom permutation, *Asiacrypt’91***,** 739:210-224 pp.

**Gnupg.org**, “libcrypt benchmarks” https://www.gnupg.org/blog/img/libgcrypt-1.6.0-cipher-bench.png, (Erişim tarihi: 30 Mayıs 2017)

**Frankel, S. and Herbert H.,** 2003, The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec. RFC 3566 (Proposed Standard).

**Kelsey J., Schneier B., Wagner D.,** 1997, Related-key cryptanalysis of 3-WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA. In: Han Y., Okamoto T., Qing S. (eds) Information and Communications Security. ICICS 1997. Lecture Notes in Computer Science, vol 1334. Springer, Berlin, Heidelberg.

**Kilian J., Rogaway P.,** 1996, How to Protect DES Against Exhaustive Key Search. In: Koblitz N. (eds) Advances in Cryptology — CRYPTO ’96.

## KAYNAKLAR DİZİNİ (devam)

**Mouha N. and Luykx A.,** 2015 Multi-key Security: The Even-Mansour Construction Revisited. In: Gennaro R., Robshaw M. (eds) Advances in Cryptology -- CRYPTO 2015. CRYPTO 2015. Lecture Notes in Computer Science, vol 9215. Springer, Berlin, Heidelberg

**Nachef V., Patarin J., Volte E.,** 2017 DES and Variants: 3DES, DES – X.: Feistel Ciphers. Springer, Cham

**Paar, C. and Pelzl, J.,** 2010, *Paar and Pelzl*. Berlin: Springer.

**Phan R.C.W.,** 2004, Related-Key Attacks on Triple-DES and DESX Variants. In: Okamoto T. (eds) Topics in Cryptology – CT-RSA 2004. CT-RSA 2004. Lecture Notes in Computer Science, vol 2964. Springer, Berlin, Heidelberg.

**Phan, R. C.-W., & Shamir, A.,** 2008, Improved related-key at tacks on DESX and DESX+.Cryptologia,32(1), 13–22.

**Rogaway, P**., 1996, The security of DESX, *Cryptobytes*, (Summer,1996):8-11pp.

**Sakallı, M.T.,** 2006, Modern Şifreleme Yöntemlerinin Gücünün İncelenmesi, Doktora Tezi, Trakya Üniversitesi.

**Stallings, W.,** 1999, Cryptography and network security: Principles and practice. Upper Saddle River, N.J: Prentice Hall.

**Sakallı, M.T.,** 2006, Modern Şifreleme Yöntemlerinin Gücünün İncelenmesi, Doktora Tezi, Trakya Üniversitesi

**Wiener, M.,** (1996), "Efficient DES key search." Technical Report TR-244, School of Computer Science, Carleton University (May 1994). Reprinted in *Practical Cryptography for Data Internetworks*, W. Stallings, editor, IEEE Computer Society Press, 31–79.

**Wiki.openwrt.org**, "Openssl benchmarks", https://wiki.openwrt.org/doc/howto/benchmark.openssl, (Erişim tarihi: 30 Mayıs 2017)

# ÖZGEÇMİŞ

Kişisel Bilgiler:

Ad Soyad: Mehmet TÜFEKÇİ

Doğum Tarihi: 21.10.1984

Doğum Yeri: ANKARA

İletişim Bilgileri:

Adres: Turgut Özal Mah. 2102. Cad. No:21/35 Yenimahalle/ANKARA

Tel: (0505) 773 93 08

e-mail: mtufekci1446[hotmail,gmail].com

Eğitim:

- 2015- Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü (UBE) Bilgi Teknolojileri Anabilim Dalı
- 2004-2008 Kara Harp Okulu Sistem Mühendisliği (Elektrik-Elektronik Bölümü)
- 2000-2004 Maltepe Askeri Lisesi

İş Tecrübeleri:

- 2008- K.K.K.lığı (Subay)

Yabancı Diller:

- İngilizce
- Almanca

# EKLER

Ek 1 Güvenlik Analizinde Kullanılan Kodlar

Ek 2 Türkçe-İngilizce Terimler Sözlüğü

## EK1 Güvenlik Analizinde Kullanılan Kodlar

### AESCBC.cpp

```
#include "osrng.h"
using CryptoPP::AutoSeededRandomPool;
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
#include <string>
using std::string;
#include <cstdlib>
using std::exit;
#include "cryptlib.h"
using CryptoPP::Exception;
#include "hex.h"
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;
#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;
#include "aes.h"
using CryptoPP::AES;
#include "ccm.h"
using CryptoPP::CBC_Mode;
#include "assert.h"
#include <sstream>

const double CLOCK_TICKS_PER_SECOND = 1000000.0;
int main(int argc, char* argv[])
{
AutoSeededRandomPool prng;
clock_t begin, end;
begin = clock();
```

```
byte key[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key, sizeof(key));
byte iv[AES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));
CBC_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
CBC_Mode< AES >::Decryption d;
d.SetKeyWithIV(key, sizeof(key), iv);
end = clock();
double diff = double(end - begin)* 1000.0 / CLOCK_TICKS_PER_SECOND;
clock_t startTime, finishTime;

std::string plain = "000102030405060708090A0B0C0D0E0F…….";
(Veri uzunluğu çok uzun olduğu için kısaltılmıştır)
std::string cipher, encoded, recovered;
startTime = clock();

 for ( int i = 0; i < 200000; i++ )
 {
        StringSource s(plain, true,
                new StreamTransformationFilter(e,
                        new StringSink(cipher)
                )
        );
  }

finishTime = clock();
double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;
std::cout << "Encryption loop execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;
std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;
 std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;
 double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;
```

```
std::cout << "Encryption/decryption loop execution time MİB/S: " << data_rate_MiBps * 200000 << " MİB/S." << std::endl;
        return 0;
}
```

**AESCBC256.cpp**

```
#include "osrng.h"
using CryptoPP::AutoSeededRandomPool;
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
#include <string>
using std::string;
#include <cstdlib>
using std::exit;
#include "cryptlib.h"
using CryptoPP::Exception;
#include "hex.h"
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;
#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;
#include "aes.h"
using CryptoPP::AES;
#include "ccm.h"
using CryptoPP::CBC_Mode;
#include "assert.h"
#include <sstream>

const double CLOCK_TICKS_PER_SECOND = 1000000.0;
int main(int argc, char* argv[])
{
```

```
AutoSeededRandomPool prng;
clock_t begin, end;
begin = clock();
byte key[AES::MAX_KEYLENGTH];
prng.GenerateBlock(key, sizeof(key));
byte iv[AES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));
CBC_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
end = clock();
double diff = double(end - begin)* 1000.0 / CLOCK_TICKS_PER_SECOND;
clock_t startTime, finishTime;

std::string plain = "000102030405060708090A0B0C0D0E0F……";
(Veri uzunluğu çok uzun olduğu için kısaltılmıştır)
std::string cipher, encoded, recovered;
startTime = clock();

for ( int i = 0; i < 200000; i++ )
{
        StringSource s(plain, true,
                new StreamTransformationFilter(e,
                        new StringSink(cipher)
                )
        );
 }

finishTime = clock();
double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;

std::cout << "Encryption loop execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;
std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;
 std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;
```

```
 double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;
std::cout << "Encryption/decryption loop execution time MİB/S: " << data_rate_MiBps * 200000 << " MİB/S." << std::endl;
return 0;
}
```

**AESCTR.cpp**

```
#include "osrng.h"
using CryptoPP::AutoSeededRandomPool;
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
#include <string>
using std::string;
#include <cstdlib>
using std::exit;
#include "cryptlib.h"
using CryptoPP::Exception;
#include "hex.h"
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;
#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;
#include "aes.h"
using CryptoPP::AES;
#include "ccm.h"
using CryptoPP::CTR_Mode;
#include "assert.h"
#include <sstream>

const double CLOCK_TICKS_PER_SECOND = 1000000.0;
```

```
int main(int argc, char* argv[])
{
AutoSeededRandomPool prng;
clock_t begin, end;
begin = clock();
byte key[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key, sizeof(key));
byte iv[AES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));
CTR_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
CTR_Mode< AES >::Decryption d;
d.SetKeyWithIV(key, sizeof(key), iv);
end = clock();
double diff = double(end - begin)* 1000.0 / CLOCK_TICKS_PER_SECOND;
clock_t startTime, finishTime;

std::string plain = "000102030405060708090A0B0C0D0E0F……..";
(Veri uzunluğu çok uzun olduğu için kısaltılmıştır)
std::string cipher, encoded, recovered;
startTime = clock();

for ( int i = 0; i < 200000; i++ )
{
	StringSource s(plain, true,
		new StreamTransformationFilter(e,
			new StringSink(cipher)
		)
	);
}

finishTime = clock();
double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;
```

```
std::cout << "Encryption loop execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;

std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;

 std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;

 double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;

std::cout << "Encryption/decryption loop execution time MİB/S: " << data_rate_MiBps * 200000 << " MİB/S." << std::endl;


        return 0;

}
```

**AESCTR256.cpp**

```
#include "osrng.h"

using CryptoPP::AutoSeededRandomPool;

#include <iostream>

using std::cout;

using std::cerr;

using std::endl;

#include <string>

using std::string;

#include <cstdlib>

using std::exit;

#include "cryptlib.h"

using CryptoPP::Exception;

#include "hex.h"

using CryptoPP::HexEncoder;

using CryptoPP::HexDecoder;

#include "filters.h"

using CryptoPP::StringSink;

using CryptoPP::StringSource;

using CryptoPP::StreamTransformationFilter;

#include "aes.h"

using CryptoPP::AES;

#include "ccm.h"
```

```
using CryptoPP::CTR_Mode;
#include "assert.h"
#include <sstream>

const double CLOCK_TICKS_PER_SECOND = 1000000.0;
int main(int argc, char* argv[])
{
AutoSeededRandomPool prng;
clock_t begin, end;
begin = clock();
byte key[AES::MAX_KEYLENGTH];
prng.GenerateBlock(key, sizeof(key));
byte iv[AES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));
CTR_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
CTR_Mode< AES >::Decryption d;
d.SetKeyWithIV(key, sizeof(key), iv);
end = clock();
double diff = double(end - begin)* 1000.0 / CLOCK_TICKS_PER_SECOND;
clock_t startTime, finishTime;

std::string plain = "000102030405060708090A0B0C0D0E0F……..……";
(Veri uzunluğu çok uzun olduğu için kısaltılmıştır)
std::string cipher, encoded, recovered;
startTime = clock();

for ( int i = 0; i < 200000; i++ )
{
        StringSource s(plain, true,
                new StreamTransformationFilter(e,
                        new StringSink(cipher)
                )
        );
```

```
 }


finishTime = clock();
double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;
std::cout << "Encryption loop execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;
std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;
 std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;
 double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;
std::cout << "Encryption/decryption loop execution time MİB/S: " << data_rate_MiBps * 200000 << " MİB/S." << std::endl;


        return 0;
}
```

**AESXCBC128.cpp**

```
#include "osrng.h"
using CryptoPP::AutoSeededRandomPool;
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
#include <string>
using std::string;
#include <cstdlib>
using std::exit;
#include "cryptlib.h"
using CryptoPP::Exception;
#include "hex.h"
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;
#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
```

```
using CryptoPP::StreamTransformationFilter;
#include "aes.h"
using CryptoPP::AES;
#include "ccm.h"
using CryptoPP::CBC_Mode;
#include "assert.h"
#include<sstream>
#include "misc.h"
#include "config.h"
const double CLOCK_TICKS_PER_SECOND = 1000000.0;

inline string XOR(const string& value, const string& key)
{
   string retval(value);
   CryptoPP::xorbuf((byte*)&retval[0], (const byte*)&key[0],
retval.length());
   return retval;
}

int main(int argc, char* argv[])
{
AutoSeededRandomPool prng;
clock_t begin, end;
begin = clock();
byte key1[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key1, sizeof(key1));
byte key[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key, sizeof(key));
byte key2[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key2, sizeof(key2));
byte iv[AES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));
CBC_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
end = clock();
```

```
double diff = double(end - begin)* 1000.0 / CLOCK_TICKS_PER_SECOND;
clock_t startTime, finishTime;
std::string cipher,encoded, encodediv, encodedkey1, encodedkey,
encodedkey2, recovered, prerecovered, postrecovered, prewhiten, postwhiten;

std::string plain = "000102030405060708090A0B0C0D0E0F……..";
(Veri uzunluğu çok uzun olduğu için kısaltılmıştır)

StringSource(key1, sizeof(key1), true,
        new HexEncoder(
                new StringSink(encodedkey1)));

StringSource(iv, sizeof(iv), true,
        new HexEncoder(
                new StringSink(encodediv)));

StringSource(key, sizeof(key), true,
        new HexEncoder(
                new StringSink(encodedkey)));

StringSource(key2, sizeof(key2), true,
        new HexEncoder(
                new StringSink(encodedkey2)));

startTime = clock();

for ( int i = 0; i < 200000; i++ )
 {
       prewhiten = XOR(plain, encodedkey1);
         StringSource s(prewhiten, true,
                new StreamTransformationFilter(e,
                        new StringSink(cipher)
                )
        );
```

```
        postwhiten = XOR(encoded, encodedkey2);
  }

finishTime = clock();
double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;

std::cout << "Encryption execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;

std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;
std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;

double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;

std::cout << "Encryption execution time MİB/S: " << data_rate_MiBps * 200000<< " MİB/S." << std::endl;

        return 0;
}
```

**AESXCTR128.cpp**

```
#include "osrng.h"
using CryptoPP::AutoSeededRandomPool;
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
#include <string>
using std::string;
#include <cstdlib>
using std::exit;
#include "cryptlib.h"
using CryptoPP::Exception;
#include "hex.h"
```

```
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;
#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;
#include "aes.h"
using CryptoPP::AES;
#include "ccm.h"
using CryptoPP::CTR_Mode;
#include "assert.h"
#include<sstream>
#include "misc.h"
#include "config.h"
const double CLOCK_TICKS_PER_SECOND = 1000000.0;

inline string XOR(const string& value, const string& key)
{
   string retval(value);
   CryptoPP::xorbuf((byte*)&retval[0], (const byte*)&key[0], retval.length());
   return retval;
}

int main(int argc, char* argv[])
{
AutoSeededRandomPool prng;
clock_t begin, end;
begin = clock();
byte key1[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key1, sizeof(key1));
byte key[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key, sizeof(key));
byte key2[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key2, sizeof(key2));
```

```
byte iv[AES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));
CTR_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
end = clock();
double diff = double(end - begin)* 1000.0 / CLOCK_TICKS_PER_SECOND;
clock_t startTime, finishTime;
std::string cipher,encoded, encodediv, encodedkey1, encodedkey,
encodedkey2, recovered, prerecovered, postrecovered, prewhiten, postwhiten;

std::string plain = "000102030405060708090A0B0C0D0E0F……..……";
(Veri uzunluğu çok uzun olduğu için kısaltılmıştır)

StringSource(key1, sizeof(key1), true,
        new HexEncoder(
                new StringSink(encodedkey1)));

StringSource(iv, sizeof(iv), true,
        new HexEncoder(
                new StringSink(encodediv)));

StringSource(key, sizeof(key), true,
        new HexEncoder(
                new StringSink(encodedkey)));

StringSource(key2, sizeof(key2), true,
        new HexEncoder(
                new StringSink(encodedkey2)) );

startTime = clock();

for ( int i = 0; i < 200000; i++ )
 {
       prewhiten = XOR(plain, encodedkey1);
```

```
        StringSource s(prewhiten, true,
                new StreamTransformationFilter(e,
                        new StringSink(cipher)
                )
        );
        postwhiten = XOR(encoded, encodedkey2);
 }

finishTime = clock();

double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;
std::cout << "Encryption execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;

std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;
 std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;

double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;

std::cout << "Encryption execution time MİB/S: " << data_rate_MiBps * 200000 << " MİB/S." << std::endl;

        return 0;
}
```

**AESXPLUSCBC.cpp**

```
#include "osrng.h"
using CryptoPP::AutoSeededRandomPool;
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
#include <string>
using std::string;
```

```
#include <cstdlib>
using std::exit;
#include "cryptlib.h"
using CryptoPP::Exception;
#include "hex.h"
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;
#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;
#include "aes.h"
using CryptoPP::AES;
#include "ccm.h"
using CryptoPP::CBC_Mode;
#include "assert.h"
#include<sstream>
#include "misc.h"
#include "config.h"
const double CLOCK_TICKS_PER_SECOND = 1000000.0;

inline string XOR(const string& value, const string& key)
{
   string retval(value);
   CryptoPP::xorbuf((byte*)&retval[0], (const byte*)&key[0],
retval.length());
   return retval;
}

int main(int argc, char* argv[])
{
std::string cipher,encoded, encodediv, encodedkey1, encodedkey,
xoredkey, recovered, prerecovered, postrecovered,
prewhiten, postwhiten;
AutoSeededRandomPool prng;
```

```
clock_t begin, end;
begin = clock();
byte key1[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key1, sizeof(key1));
byte key[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key, sizeof(key));
byte iv[AES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));
CBC_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
end = clock();

double diff = double(end - begin)* 1000.0 / CLOCK_TICKS_PER_SECOND;
clock_t startTime, finishTime;

std::string plain = "000102030405060708090A0B0C0D0E0F……..……..";
(Veri uzunluğu çok uzun olduğu için kısaltılmıştır)

StringSource(key1, sizeof(key1), true,
        new HexEncoder(
                new StringSink(encodedkey1)));

StringSource(iv, sizeof(iv), true,
        new HexEncoder(
                new StringSink(encodediv)));

StringSource(key, sizeof(key), true,
        new HexEncoder(
                new StringSink(encodedkey)));

xoredkey = XOR(encodedkey, encodedkey1);
startTime = clock();

for ( int i = 0; i < 200000; i++ )
```

```
{
        prewhiten = XOR(plain, encodedkey1);
          StringSource s(prewhiten, true,
                    new StreamTransformationFilter(e,
                              new StringSink(cipher)
                    )
          );
        postwhiten = XOR(encoded, xoredkey);
  }

finishTime = clock();

double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;
std::cout << "Encryption execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;

std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;
 std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;

double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;

std::cout << "Encryption execution time MİB/S: " << data_rate_MiBps * 200000 << " MİB/S." << std::endl;

        return 0;
}
```

**AESXPLUSCTR.cpp**

```
#include "osrng.h"
using CryptoPP::AutoSeededRandomPool;
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
```

```
#include <string>
using std::string;
#include <cstdlib>
using std::exit;
#include "cryptlib.h"
using CryptoPP::Exception;
#include "hex.h"
using CryptoPP::HexEncoder;
using CryptoPP::HexDecoder;
#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;
#include "aes.h"
using CryptoPP::AES;
#include "ccm.h"
using CryptoPP::CTR_Mode;
#include "assert.h"
#include<sstream>
#include "misc.h"
#include "config.h"
const double CLOCK_TICKS_PER_SECOND = 1000000.0;

inline string XOR(const string& value, const string& key)
{
   string retval(value);
   CryptoPP::xorbuf((byte*)&retval[0], (const byte*)&key[0],
retval.length());
   return retval;
}

int main(int argc, char* argv[])
{
std::string cipher,encoded, encodediv, encodedkey1, encodedkey,
xoredkey, recovered, prerecovered, postrecovered,
```

```
prewhiten, postwhiten;
AutoSeededRandomPool prng;
clock_t begin, end;
begin = clock();
byte key1[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key1, sizeof(key1));
byte key[AES::DEFAULT_KEYLENGTH];
prng.GenerateBlock(key, sizeof(key));
byte iv[AES::BLOCKSIZE];
prng.GenerateBlock(iv, sizeof(iv));
CTR_Mode< AES >::Encryption e;
e.SetKeyWithIV(key, sizeof(key), iv);
end = clock();

double diff = double(end - begin)* 1000.0 / CLOCK_TICKS_PER_SECOND;
clock_t startTime, finishTime;

std::string plain = "000102030405060708090A0B0C0D0E0F……..";
(Veri uzunluğu çok uzun olduğu için kısaltılmıştır)

StringSource(key1, sizeof(key1), true,
        new HexEncoder(
                new StringSink(encodedkey1)));

StringSource(iv, sizeof(iv), true,
        new HexEncoder(
                new StringSink(encodediv)));

StringSource(key, sizeof(key), true,
        new HexEncoder(
                new StringSink(encodedkey)));

xoredkey = XOR(encodedkey, encodedkey1);
startTime = clock();
```

```
for ( int i = 0; i < 200000; i++ )
 {
        prewhiten = XOR(plain, encodedkey1);
          StringSource s(prewhiten, true,
                    new StreamTransformationFilter(e,
                              new StringSink(cipher)
                    )
          );
        postwhiten = XOR(encoded, xoredkey);


  }

finishTime = clock();

double executionTimeInSec = double( finishTime - startTime ) / CLOCK_TICKS_PER_SECOND;

std::cout << "Encryption execution time: " << executionTimeInSec * 1000.0 << " microseconds." << std::endl;
std::cout << "Microseconds for Key Setup and Iv: " << diff  << std::endl;
 std::cout << "Plain text size: " << plain.size() << " bytes." << std::endl;

double data_rate_MiBps = ((double)plain.size() / 1048576) / ((double)executionTimeInSec) ;

std::cout << "Encryption execution time MİB/S: " << data_rate_MiBps * 200000 << " MİB/S." << std::endl;

        return 0;
}
```

## EK2 Türkçe-İngilizce Terimler Sözlüğü

(Türkiye Bilişim Derneği Bilişim Sözlüğü kullanılmıştır)

| | |
|---|---|
| Şifreleme / çözme | : Encryption / Decryption |
| Özel veya | : xor (Exclusive OR) |
| Anahtar arama saldırısı | : Exhaustive key search attack |
| İlişkili anahtar saldırsı | : Related key attack |
| Ortadaki adam saldırısı | : Meet in the middle attack |
| Anahtar beyazlatma | : Key whitening |
| Düz veri | : Plaintext |
| Şifreli veri | : Ciphertext |
| Simetrik şifreleme | : Symetric cipher |
| Blok şifreleme | : Block cipher |
| Akım şifreleme | : Stream cipher |
| Elektronik Kod Defteri Modu | : Electronic Code Book Mode |
| Şifreli Blok Zincirleme Modu | : Cipher Block Chaining Mode |
| Çıktı Geri Beslemeli Modu | : Output Feedback Mode |
| Şifre Geri Beslemeli Modu | : Cipher Feedback Mode |
| Sayıcı Şifreleme Modu | : Counter Mode |
| Baytların yer değiştirmesi | : Byte substitution |
| Satırların ötelenmesi | : Shift rows |
| Sütunların karıştırılması | : Mix column |

| | |
|---|---|
| Tur anahtarı eklenmesi | : Add round key |
| Anahtar üretme | : Key schedule |
| Kelime | :Word |
| Tur katsayısı | : Round coefficient |
| Uygulama programlama arayüzü | : Application programming interface |
| Tek yönlü karma fonksiyon | : One-way hash function |
| Mesaj asıllama kodları | : Message authentication codes |
| Komut zinciri | : Pipelining |
| İnkar edilemezlik | : Nonrepudiation |
| Yan-kanal analizi | : Side channel analysis |
| Kodkitabı saldırısı | :Codebook attack |
| Seçilmiş şifreli veri saldırısı | : Chosen ciphertext attack |
| Seçilmiş düz veri saldırısı | : Chosen ciphertext attack |
| Kısmi deşifreleme saldırısı | : Partial decryption attack |
| Bilinen metin saldırsı | : Known plaintext attack |
| İlişkili anahtar ayırıcı | : Related key distinguisher |
| Gelişmiş kayma saldırısı | : Advanced slide attack |
| Şifreleme kahini | : Encryption oracle |
| Karma | : Hash |
| Sol el tarafı eşitliği | : Left Hand Side Equation |
| DES tamamlama özelliği | : DES complementation property |