İSTANBUL ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

Data Şifreleme Algoritmaları ve Performans Analizi

İlhan CİĞER

Elektrik – Elektronik Mühendisliği Anabilim Dalı
Elektrik – Elektronik Mühendisliği Programı

Danışman
Prof. Dr. Sıddık Binboğa Yarman

İSTANBUL

Bu çalışma 06/06/2012 tarihinde aşağıdaki jüri tarafından . Elektrik – Elektronik Müh. Anabilim Dalı Elektrik – Elektronik programında Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Jürisi

Danışman Adı (Danışman) Prof.Dr.Sıddık Yarman
Üniversite
İstanbul Üniversitesi

Jüri Adı Prof.Dr.Aydın Akan
Üniversite
İstanbul Üniversitesi

Jüri Adı Prof.Dr.İlhan Kocaarslan
Üniversite
İstanbul Üniversitesi

Jüri Adı Prof.Dr.Ayten Kuntman
Üniversite
İstanbul Üniversitesi

Jüri Adı Yrd.Dç.Dr.Ümit Güz
Üniversite
Işık.Üniversitesi

I.ÖNSÖZ

Yüksek lisans öğrenimim ve tez çalışmalarım süresince gösterdikleri her türlü destek ve özverilerinden dolayı değerli hocalarım Prof. Dr. Sıddık Yarman a ve tezin oluşmasında sağladığı motivasyon, bir arkadaş gibi güler yüzle sabırlı yaklaşımı ve yol gösterici önerileri için Doç. Dr. Albert Levi'e en içten dileklerimle teşekkür ederim.

Ayrıca tez çalışmalarım boyunca manevi desteğini benden esirgemeyen sevgili aileme ve dostlarıma göstermiş oldukları sabır ve sonsuz güler yüz için de şükranlarımı sunarım.

Mayıs, 2012 İLHAN CİĞER

II.İÇİNDEKİLER

III. ŞEKİL LİSTESİ

IV. TABLO LİSTESİ

V. ÖZET

DATA ŞİFRELEME ALGORİTMALARI VE PERFORMANS ANALİZİ

Kriptografi bilgi güvenliğini inceleyen bilim dalıdır. Bir mesajın yada bilginin geçici olarak okunamaz hale dönüştürülerek hedefine ulaştırılması ve karşı tarafta bilginin tekrar okunabilir hale döndürülmesi için kullanılan şifreleme konusuna verilmiş genel addır. Bilgi güvenliğinin öneminin yüksek olmasından dolayı kriptografide data şifreleme algoritmaları ön plandadır.

Bu çalışmada data şifreleme algoritmaları tanıtılmış, kullandıkları anahtar bakımından farklılık gösteren simetrik ve asimetrik şifreleme kıyaslaması yapılmış, performans karşılaştırması sonuçları alınıp analiz edilmiştir. Analiz için ModelSim Altera Web Edition programı kullanılarak RSA,AES ve DES için kaynak kodları yazılmıştır. Akabinde algoritmaların şifreleme ve şifre çözmedeki yeteneklerinin hız ve memory gibi unsurlarda hesaba katılarak detaylı bir şekilde incelemesi yapılmıştır. Son olarak ses,video ve real time data transferlerinde şifreleme algoritmalarına değinilmiştir. Analiz sonuçlarına kısaca göz atarsak RSA algoritması işlemleri daha uzun sürede gerçekleştirir. AES algoritması RSA ve DES e göre çok hızlı, DES algoritması ise RSA ya göre daha hızlıdır. Ayrıca Blowfish algoritmasının kod çözmedeki performansı çok üstündür. Buna karşın güvenliğin üst düzeyde olduğu uygulamalarda RSA bir adım öndedir. Son olarak authentication ın büyük önem taşıdığı vpn uygulamalarında şifelemeye değinilmiştir. Burada iletişim kuran noktalar arasındaki tunelin durumuna göre şifrelemenin değişkenlik gösterdiği görülmüştür.

VI.SUMMARY

DATA ENCRYPTION ALGORITHMS AND PERFORMANCE ANALYSIS

Cryptology is a science that deals with the security of information. It has been coined as an encryption topic for delivering a message or information, by changing it to a temporary unreadable mode and turning it to a readable state on the opposite side. As it carries information of high with security of information, data encryption algorithms are in the foreground. In this workout, data encryption algorithms are introduced, asymmetric and symmetric algorithms are compared, which are differentiated on using keys are performed and the comparison results in a detailed analysis are implemented. By using ModelSim Altera Web Edition for analysis source codes, RSA, AES and DES have been written. Afterwards, a detailed investigation had been done for capabilities of algorithms on encryption and decryption, considering speed and memory requirements. Finally it is told encryption algorithms for voice,video and real time data transfer. According to analysis results, RSA algorithm performs operations slower than other algorithms, which were compared in this thesis. AES is the fastest algorithm regarding DES and RSA; however, DES is faster than RSA. Furthermore, Blowfish has the best performance level of decryption. On the other hand, RSA is one step ahead on applications that security has crucial importance. Lastly, encryption on VPN applications, which holds authentication with top priority, have been mentioned in this workout. Related with analysis, it was seen that encryption has been instable regarding situation of tunnel, which was established between communication points.

1.1.Giriş

İnternette yollanan veri paketleri halka açık networklerden geçer, bu da paketlere ulaşmayı mümkün kılar. Son derece gizli bilgiler internette transfer edilirken, bu durum önemli bir kaygı haline gelmiştir. Bu tür bilgilerin korunması sağlanmadan, internette iş yapmak veya yazışmalarda bulunmak güvenli olmayacaktır. Bilgi güvenliği başkası tarafından dinlenme, bilginin değiştirilmesi, kimlik taklidi gibi tehditlerin ortadan kaldırılması ile sağlanır ve bu amaçla kullanılan temel araç kriptografidir. Güvenilirlik, veri bütünlüğü, kimlik doğrulama gibi bilgi güvenliği konularıyla ilgilenen matematiksel yöntemler üzerine yapılan çalışmalar kriptografinin önemli konularıdır.

Güçlü şifreleme eskiden sadece askeri çalışmalarda kullanılırdı; ancak günümüz bilgi toplumunda gizlilik ve şahsi bilgilerin korunması için önemli araçlardan biri olmuştur. Günümüzde özellikle bankalarda veri alışverişinde, vpn gibi yüksek güvenlik sunan bağlantı türlerinde ve istihbarat açısından çok kritik öneme sahip bilgilerin iletilmesinde şifreleme yapılır.

Günümüz internet çağında ise bireysel iletişimin çok yaygınlaşması ve internet uygulamalarının finans, ticaret ve kamu hizmetleri gibi önemli alanlara yayılmış olması bilgi güvenliğinin sağlanmasının önemini daha da arttırmıştır. Bu ihtiyaç neticesinde şifreleme (kriptografi) alanında yapılan çalışmalarda büyük bir artış olmuştur.

Bu kavramlardan kriptografi; güvenilirlik, veri bütünlüğü ve veri kaynağının doğruluğu gibi bilgi güvenligi ile ilgili matematiksel tekniklerin bir araya geldigi bir çalışma alanıdır. Kriptanaliz kriptografik mekanizmaların nasıl çözüleceği üzerine çalışılması, kriptoloji ise kriptografi ve kriptanalizin bir arada olduğu bilim dalıdır. Deşifre etme (decryption) ise şifreleme olayının tersidir; şifrelenmiş verinin bilgi alınabilir bir forma yeniden dönüştürülmesidir. [9]

1.2.Amaç

Doğrulama (authentication) ve kimlik tespiti (identification), elektronik ticaret (e-commerce), belgeleme (certification),güvenli uzaktan ulaşım (vpn) gibi uygulama alanları kriptografinin günümüzde yaygın olarak kullanıldığı yerlerdir.

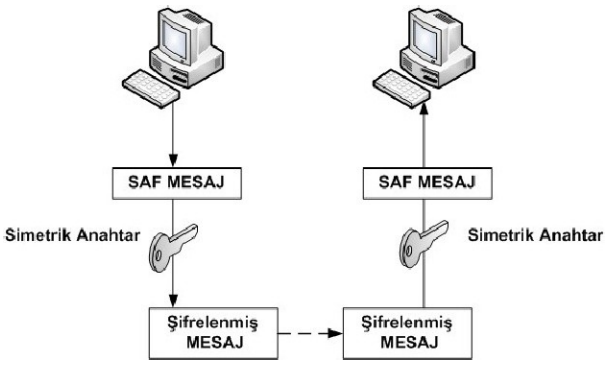
Kriptografide amaç verinin şifrelenmesi ve hedef bilgisayarda eski haline yani şifresiz okunabilir haline çevrilmesidir. Mesela mail hesabınıza girmek için kullanıcı adı ve şifre yazmanız gerekir. Yazdığınız şifre ve kullanıcı adı herhangi birinin eline geçmemeli veya geçse bile sizden başkası tarafından okunamamalı. İnternet üzerinden mail sunucularına gönderdiğimiz bu şifrenin güvenilir olarak bilgisayarına ulaşmasından emin olmamız gerekiyor. İşte şifreleme bu noktada devreye giriyor. Siz görmesenizde, arka planda e-posta şifreniz kriptografik olarak şifreleniyor ve mail bilgisayarlara şifrelenmiş haliyle yollanıyor ki arada sizin internet hattınızı dinleyen olası kötü niyetli kişiler e-posta şifrenizi göremesin.

Şifreleme sürecinde, düz bir veriyi(mesajı) okunamaz hale getirmek için çok kompleks matematiksel algoritmalar kullanılıyor. Bu algoritmalara örnek olarak DES (Data Encryption Algorithm) ve AES (Advanced Encryption Algorithm) gösterilebilir. [1]

Bu algoritmalar, kendilerine gelen mesajı belli parçalara bölüp her parça üzerinde matematiksel işlemler gerçekleştiriyor. Klavyede yazdığımız harfler bitlere (bit 0 yada 1 olabilir, bilgisayarlar makine dili olan bitler yani 0 ve 1 ler üzerinden anlaşıyor) dökülüyor ve bu bitler üzerine karmaşık ve rastgele olarak matematiksel fonksiyonlar uygulanıyor. [2]

Fakat önemli bir nokta var; bu algoritmalar belli bir süre sonra ya art niyetli kişiler tarafından bulunuyor ya da geliştiriciler tarafından açıklanıyor. Açıklanmasının sebebi ise, algoritmadaki eksikliklerin ya da geliştirilmesi gereken kısımların ilgilenen kişiler tarafından görülmesi ve bunun sonucunda algoritmanın geliştirilmesinin amaçlanmasıdır.

Peki algoritmalar herkes tarafından biliniyorsa gizlilik nerede rol oynamaktadır? Cevabı şifre ya da anahtar. Bir kriptografi algoritmasına gizliliği katan tek faktör şifredir. Bu yüzden çok sıkı bir şekilde korunması ve saklanması gerekiyor. Şifrelerin büyük bir çoğunluğu 512 bit uzunluğuna kadar uzanıyor; bu da 0 dan 2^512 ye kadar bir sayı demek. Şifrenin içeriğine göre, kriptografik algoritmanın işleyişi değişiyor. Yani şifredeki bir tane bile bit değişiminin sonucu bambaşka bir şifrelenmiş mesaj olarak sonuçlanıyor. [8]



Şekil 1.1 Simetrik Şifreleme Yapısı

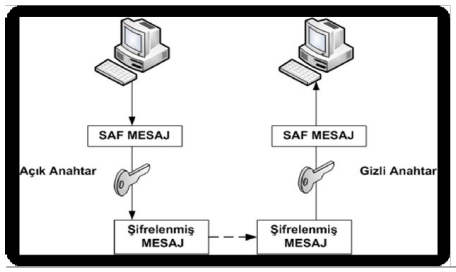
1.3. Şifreleme Türleri

Kriptografide iki tane şifreleme türü vardır. Simetrik ve asimetrik.

Simetrik şifrelemede kriptolayarak gönderilmek istenilen mesaj bir şifre tarafından algoritmaya sokulur ve şifrelenir. Bu şifrelenmiş mesajı alan tarafında mesajı okunabilir yani çözülmüş haline geri döndürmesi gerekiyor.

Şifrelenmiş mesajı alan taraf yine aynı simetrik şifreyi kullanarak mesajı açıyor. Yani simetrik şifrelemede kriptolamak ve çözmek için kullanılan anahtarlar (şifreler) aynıdır.

Asimetrik şifrelemede ise mesajı şifrelerken kullanılan anahtar ve çözmek için kullanılan anahtarlar farklıdır. Şifrelemek için public şifre çözmek için private key kullanılır. Örneğin mesajı A şifresiyle kriptoladınız ve gönderdiniz. Mesaj eski haline sadece B anahtarıyla çevrilebiliyor. Tam tersi yönde de B anahtarıyla şifrelenen bir mesaj sadece A anahtarıyla çözülebiliyor.



Şekil 1.2 Asimetrik Şifreleme Yapısı

Peki neden iki farklı sisteme gereksinim duyulmaktadır. Simetrik sisteme bakarsak; sistemin çalışması için kriptolamak ve çözmek için kullanılan anahtarların mesajı gönderen ve alan tarafların ikisinde de bulunması gerekiyor. Bunun sonucunda bu şifrenin gönderici ve alıcı taraflara güvenli olarak iletilmesi gerekiyor. Bu başlı başına bir sorun. İkinci olarak simetrik şifrelemede mesajın kim tarafından şifrelendiği açık değildir. Yani ben size 5 dakika içinde simetrik anahtarla şifrelenmiş bir mesaj gönderebilir ve 10 dakika sonra bunu inkar edebilirim. Aksini kanıtlayamazsınız çünkü mesajı açmak için aynı şifre sizde de bulunuyor.

Asimetrik şifreleme bu sorunlara çözümler sunuyor. İlk soruna yani şifrenin taşınması ve taraflara dağıtılmasına bakarsak; asimetrik şifrelemede mesajı şifreleyen tarafın kullandığı anahtarın çözücü eşi alıcı tarafında tutuluyor. Bu sistem de bu anahtarlardan birine "açık" (yani herkes tarafından öğrenilebilir ve kullanılabilir) diğerine ise "gizli" (sadece taraflardan birine ait) anahtar deniliyor. Gizli anahtar sadece bir tarafa aittir. Yani sizin gizli anahtarınız sizin için oluşturulmuş ve sadece sizin tarafınızdan kullanılabilen bir anahtar. Bu anahtarın eşi olan açık anahtar ise "Anahtar kütüphanelerinde" tutuluyor. Yani size kriptolanmış bir anahtar yollamak isteyen biri, bu anahtar kütüphanelerinden birinden sizin "açık" anahtarınızı alıyor ve mesajı bu "açık" anahtarla şifreleyip size gönderiyor. Bu "açık" anahtarın çözücü eşi olan "gizli" anahtar zaten sadece sizde var. Bu şekilde mesajı aldıktan sonra siz çözebiliyorsunuz.

1.4.Hash

Basitçe “hash” gönderilecek verinin (bilginin yada mesajın) belirli bir fonksiyona sokulup, sonucunda matematiksel olarak tek bir sonuç elde edilmesidir. Farzedelim karşı tarafa mesaj göndermek istiyoruz. Bu mesajın bilgisayar dilinde karşılığı doğal olarak 0 ve 1 lerden oluşan bir dizi olacak. Bu dizi 0100100001010011 olsun. 0100100001010011 “hash” fonksiyonuna dahil olduktan sonra hash sonucu tamamen farklı 0100001011011010 olacaktır. Önemli nokta ise hash sürecinin sadece tek yönlü olmasıdır.

Yani, hash fonksiyonu sonucu ortaya çıkan bilgiden hash öncesi orjinal bilgiyi elde etmenin yolu yoktur. Bu durumda hash sonucu karşıdaki bilgisayara gönderildiği esnada, orjinal mesajın da hash sonucuyla birlikte gönderilmesi gerekiyor. Alıcı bilgisayar, orjinal mesaja hash fonksiyonunu uygulayıp, çıkardığı sonucu kendisine orjinal mesajla gönderilen hash sonucuyla karşılaştırıyor. Eğer alıcı bilgisayarın oluşturduğu hash sonucu, mesajla birlikte gönderilen hash sonucuyla aynıysa alıcı bilgisayar kendisine gelen orjinal mesajın üzerinde oynama yapılmadığından emin olmuş oluyor.

Hash sonucunun orjinal mesajla birlikte gönderilmesinin tek sebebi orjinal mesajın gönderim sırasında değişikliğe uğramadığından emin olmaktır. [11]

1.5.Şifreleme Tarihçesi:

Yapılan araştırmalar bilgiyi gizleme ve koruma olayının yaklaşık 4000 yıllık bir serüvene sahip olduğunu ortaya koymaktadır. Eski Mısırlıların, M.Ö. 1900 lü yıllarda bir tanrının efsanesini anlatan hiyeroglifflerin bazı yerlerinde ilginç sembolleri kullandığı görülmüştür. İncelemeler, yazının bazı yerlerinin bu değişik sembollerle kodlandığını göstermiştir. Mısırlılardan sonra Mezopotamyalılar ve İbraniler de bazı metinleri benzeri şekillerde kodlamışlardır.

En ilginç kripto sistemlerden biri M. Ö. 400 yıllarında Spartalılar tarafından geliştirilmiş olan "scytale" denilen sistemdir. Bu sistemde bir mesajı "encrypt" etmek için uzun bir parsömen ya da papirüs silindirik bir sopa etrafına sarılıyordu. Gizlenecek mesajın kelimeleri uzunlamasına sopa üzerine her bir şerit turunda 1 harf gelecek şekilde yazılıyordu. Daha sonra şerit açılır ve kaldırılırdı. Böylece anlamsız harflerin oluşturduğu metin ortaya çıkardı. Mesajın "decrypt" (şifre çözme) edilebilmesi için gereken kritik şart "encrypt" işleminde kullanılan silindirle aynı çapa sahip silindir kullanılması şartıydı. Farklı çaptaki silindirler anlamsız metinlerin ortaya çıkmasına sebep oluyordu.

En önemli gelişmelerden biri 2000 yıl önce Julius Caesar'ın Galya savaşlarında kendi adıyla anılan şifreleme yöntemini kullanmasıdır. Caesar bu yöntemde her harfi kendinden sonra gelen üçüncü harf ile değistirmiştir. Bu basit yerine koymalı şifrelemenin örneklerindendir. Daha sonra Caesar'ın kullandığı yöntem geliştirilerek monoalfabetik yerine koymalı şifre geliştirilmiştir.

Monoalfabetik yerine koymalı şifrede alfabedeki her harfin yerine gönderici ve alıcının bildiği bir harf konulur. Caesar şifresinden farkı bir düzene bağlı olmamasıdır.

Arap bilim adamı Al-Kindi'nin 1987 de İstanbul'da bulunan el yazmasından monoalfabetik yerine koymalı şifrenin Araplar tarafından kırıldığını bulması onların bu konudaki bilgisini göstermektedir. Şifrelerin çözülmesi yeni şifrelerin geliştirilmesine yol açmıştır. Bunun sonucu olarak her harfin, ortak kelimelerin ve boşlukların yerine birden fazla sembol yerleştirilmeye başlanmıştır. Bu şifrenin kayıtlara geçen örnekleri İngiltere kraliçesi I. Elizabeth'e yapılan suikast planları ve 17. yy Fransa'sında XIV Louis'in Büyük Şifresidir (Great Cipher). Louis'in bu şifresi, 2 yy boyunca kumandan Etienne Bazeries tarafından kırılana kadar çözülememiştir.

20 .yüzyıla kadar bu süreçte 2.DÜNYA savaşına kadar belirli bir ilerleme olmamıştır.
İttifak devletlerinin ünlü Alman "Enigma" ve Japonların "Purple" kodlarını kırmaları II. Dünya Savaşının sonucunu belirleyen faktörler olarak görülür. Enigma algoritması Polonyalı matematikçi Marian Rejewski tarafından kırılmıştır. Böylece kriptografi yeni bir boyut kazandı.

IBM tarafından 1970'lerin başlarında çalışmaları başlayan ve 1977 yılında bilgilerin kriptolanması için ABD'nin federal bilgi işleme standardı olarak benimsenen veri kriptolama standardı (Data Encrypting Standard – DES) tarihteki en yaygın kullanıma sahip kriptografi mekanizmasıdır.

Des algoritmasının ilk açıklandığı dönemde 56 bitlik anahtar kullanılarak 2^56 lık anahtar ihtimali yani 72,057,594,037,927,936 muhtemel anahtar bulunmaktaydı. İlk başlarda kırılmayacak gibi görünse de bilgisayar teknolojisinin 1990 larda gelişmesiyle DES algoritmasının kırılabildiği görüldü. Bu gelişmelerden sonra DES algoritmasının geçerliliğini kaybetmesiyle DES in 3 defa peşpeşe çalıştırılması anlamına gelen 3DES algoritması ortaya çıktı. 1997 de NIST(Ulusal Standart ve Güvenlik Enstitüsü) DES algoritmasının yerini alacak algoritma için bir yarışma düzenledi. Kazanan Rijmen algoritmasını ortaya çıkaran Belçikalı Vincent Rijmen ve Joan Daemen oldu. Şifreleme algoritması 26 Kasım 2001 de yayımlandı. Daha önceki DES ve 3DES in aksine AES algoritması 64 bitlik bloklar halinde değil 128 bitlik bloklar halinde şifreleme yapar. [12]

1.6.Saldırı Tipleri:

DES algoritmasına yönelik 3 tip saldırı yöntemi vardır.

Brute Force Atak : Saldıran kişi özel bir program yardımıyla tüm olasılıkları deneyerek anahtarı tahmin etmeye çalışır.

Differansiyel Kriptanaliz Atak : Açık metin ataktır ve saldırı mantığı S kutularının düzgün olmayan differansiyel dağıtım tablolarına dayanır.

Lineer Kriptanaliz Atak : Differansiyel kriptanaliz atağa göre bu saldırıya karşı daha savunmasızdır.

AES algoritmasına yönelik bilinen en önemli saldırı tekniği XSL ataktır.
Saldırı şifreli verinin analizi ve quadratik eş zamanlı eşitlikleri elde etmek üzerine kuruludur. Bu eşitlikler çok geniştir. Örneğin 128 bitlik AES için 1600 değişken ve 8000 eşitliktir. [13]

Kriptoloji alanındaki en çarpıcı gelişme, 1976 yılında Diffie ve Helman tarafından yayınlanan "*New Directions in Cryptography*" bildirisi ile gerçekleşmiştir. Bu bildiri, genel-anahtar kriptografi kavramı ile birlikte, anahtar alışverişi için yeni ve saf bir metot ortaya koymuştur. 1978'de Rivest, Shamir ve Adleman tarafından günümüzde RSA olarak bilinen ilk genel-anahtar kriptolama ve imzalama yapısı geliştirilmiştir. RSA'nın yapısı, geniş tamsayıların çarpanlara ayrılması mantığına dayanır. [7]

RSA tezin uygulama kısımlarında değineceğimiz gibi güvenlik açısından diğer algoritmalara göre üstündür. Bu sebeple data güvenliğinde IPSEC/IKE - IP, web ortamında yani data iletim güvenliğinde TLS/SSL, email güvenliğinde PGP ve terminal bağlantı güvenliğinde SSH platformlarında RSA tercih edilir. [13]

RSA algoritmasına yönelik 2 çeşit saldırı yöntemi vardır.

Faktörizasyon Atak : Giriş metni bozmak için çeşitli faktorizasyon algoritmaları vardır.RSA bunu yenmek için n parametresinin 300 decimal bit değerinden daha fazla olmasını ister. Yani modülün minimum 1024 bit olması istenir.

Lattice Tabanlı Atak : Lattice indirgeme algoritmaları ile parametreleri bulmaya çalışır.

2.GENEL KISIMLAR

2.1. ŞİFRELEME ALGORİTMALARI:

Orijinal veriyi M (Message) ile, şifrelenmiş veriyi ise C (cipher) ile gösterelim. Şifreleme fonksiyonu E (Encryption), M üzerinde işlem yapar ve C elde edilir.

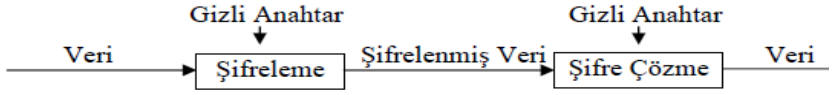
Matematiksel olarak,
E(M) = C şeklindedir.

Şifreleme fonksiyonunun tersi olan çözme fonksiyonu D (Decryption), C üzerinde işlem yapar ve M elde edilir.

D(C) = M

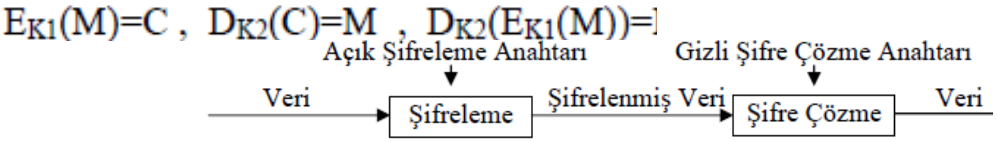
Algoritmanın güvenliği, algoritmanın nasıl işlediğini gizli tutmakla sağlanıyor ise böyle algoritmalara sınırlı (restricted) algoritma adı verilir. Modern kriptoloji bu problemi anahtar yardımıyla çözer. Anahtar K (Key) harfi ile gösterilmektedir. Anahtarın alabileceği değer aralığına anahtar uzayı (key space) denir.

$E_K(M)=C$, $D_K(C)=M$
Bu fonksiyonlarda $D_K(E_K(M))=M$ eşitliğini sağlamak durumundadır.



DES kriptolama, 64 bit blok büyüklüğüne sahip bir blok kriptolama tekniği iken AES kriptolama 128 bit'lik bloklarla işlem yapar. RSA ve Diffie-Hellman kriptolama tekniklerinde ise blok büyüklükleri sabit değil değişkendir.

Bazı algoritmalar birbirinden farklı şifreleme ve çözme anahtarı kullanır. Bu algoritmalarda K1 şifreleme anahtarını, K2 ise çözme anahtarını gösterir. İki farklı anahtar ile şifreleme yöntemi aşağıdaki şekilde verilmiştir.



İKİ FARKLI ANAHTARLA ŞİFRELEME VE ŞİFRE ÇÖZME

Kullanılan şifreleme anahtarının özelliğine bağlı olarak, iki farklı şifreleme yöntemi bulunmaktadır.

1- Simetrik (Gizli anahtar) şifreleme algoritmaları
    Akış şifreler (stream ciphers)
    Blok şifreler (blok ciphers)
2- Asimetrik (Açık anahtar) şifreleme algoritmalarıdır.

Simetrik algoritmalara bazen geleneksel algoritmalar da denir. Şifreleme anahtarının çözümleme anahtarı üzerinden hesaplanabildiği bir algoritmadır. Çoğu simetrik algoritmada şifreleme ve çözümleme anahtarları aynıdır. Bu algoritmalar, aynı zamanda gizli anahtar algoritmaları, tek anahtar algoritmaları veya bir(aynı) anahtar algoritmaları diye de adlandırılır, gönderici ile alıcının güvenle iletişime başlamadan önce bir anahtar üzerinde anlaşmalarını gerektirir. Bir simetrik algoritmanın güvenliği anahtara dayanır; anahtarın açığa çıkması herkesin mesajları şifreleyebileceği ve çözebileceği anlamına gelir. İletişimin gizli kalması gerektiği sürece, anahtar gizli kalmalıdır.

Simetrik algoritmalar iki sınıfa ayrılabilir. Bazıları belli bir anda bir bitlik açık metni şifreleyebilir; bunlar akış(stream) algoritmaları veya akış şifreleri olarak adlandırılır. Diğerleri açık-metini bit gruplarına bölerek işler. Bit grupları blok ve algoritmaları blok algoritmaları veya blok şifreleri diye adlandırılır. Çağdaş bilgisayar algoritmaları için, tipik bir bit bloğunun uzunluğu 64 bittir.

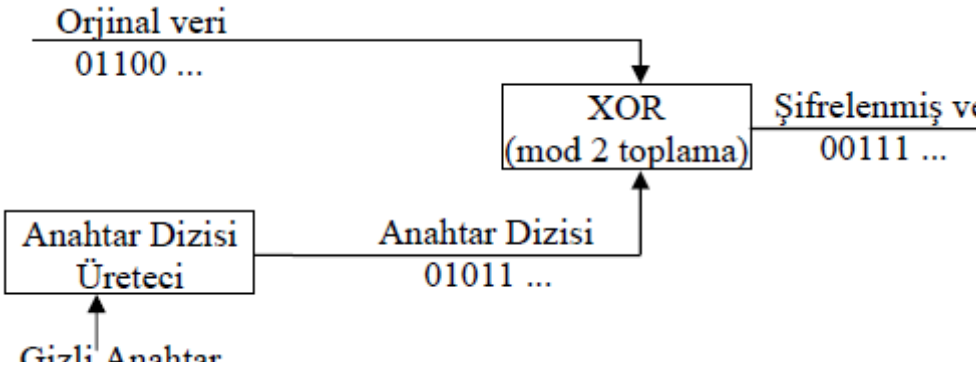
2.2.SİMETRİK ŞİFRELEME ALGORİTMALARI

2.2.1.AKIŞ ŞİFRELER:

Akış şifreleme (stream ciphers) algoritmaları, orijinal veri olarak bit dizilerini almakta ve çıktı olarak da bit dizileri üretmektedirler.

Akış şifreleme işleminde ilk olarak gizli anahtar ve üreteç yardımı ile bir anahtar dizisi oluşturulmaktadır. Daha sonra bu dizi ve girdi mesajın her bir biti ayrı ayrı XOR (mod 2 toplama) işlemine tabi tutulmaktadır. Çözme işlemide aynı şekilde gerçekleştirilmektedir. Akış şifreleme algoritmalarının güvenlikleri anahtar dizisi üreteci tarafından yaratılan dizinin ne kadar rastgele olduğuna bağlıdır.

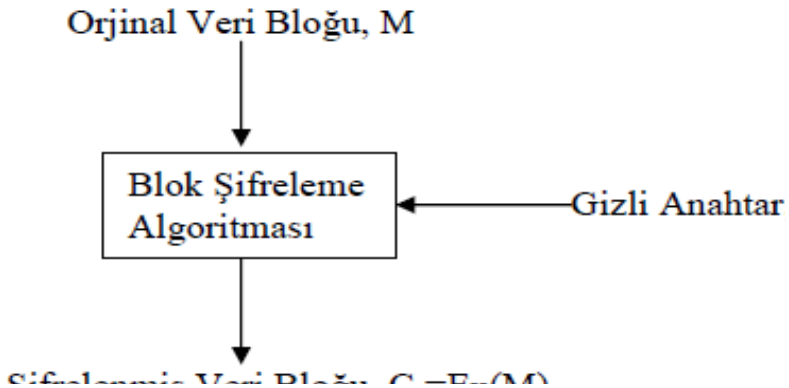
Bu nedenle anahtar dizisi olarak tamamıyla rastgele verinin kullanılması en ideal durumdur. Gerçek uygulamalarda tam anlamı ile rastgele anahtar dizisi yaratılması imkansız olduğundan, anahtar üreteci ve onun girdisi olan gizli anahtar yardımı ile anahtar dizileri yaratılmaktadır. Yani bir akış şifreleme algoritmasının en önemli bileşeni kullandığı anahtar dizisi üretecidir. Yaratılan anahtar dizisinin kendini tekrarlamaması ve sonraki anahtar bitlerinin öncekiler yardımı ile elde edilememesi anahtar üreteçlerinin sağlaması gereken önemli özelliklerindendir.



Şekil 2.1 Temel Şifreleme Örneği

2.2.2 BLOK ŞİFRELER:

Blok şifreleme algoritmaları, orijinal veri olarak bit gruplarını alır. Bu bit gruplarına blok adı verilirken, kullanılan algoritmalara da blok şifreleri denir. Modern bilgisayar algoritmalarında genel olarak tipik blok boyutu, genel olarak 32, 64 veya 128 bit olarak seçilmiştir.



Şekil 2.2 Blok Şifreleme Yapısı

BLOK ŞİFRELEME ALGORİTMASI

Feistel ağları ve Yer değiştirme-Permütasyon agları (SPN) olmak üzere iki ana blok şifreleme yapısı vardır. Birden fazla şifreleme işleminin birleşmesi ile oluşturulurlar. Her şifreleme adımı döngüdür. Her döngüde farklı anahtar materyali kullanılır. SPN ve Feistel mimarileri en yaygın olanlardır. Blok şifreleme algoritmalarında aynı girdi blogu aynı anahtarla şifrelendiğinde her zaman aynı çıktı bloğunu oluşturmaktadır. Akış şifreleme algoritmalarında ise bir mesaj içinde farklı pozisyonlarda yer alan aynı girdi blokları farklı çıktılar oluşturmaktadır.

Blok şifreleme sistemlerinin parametreleri:

Blok Uzunluğu : Bir blok şifre sisteminin güvenli olabilmesi için, blok uzunluğunun bazı blokların diğerlerinden daha fazla görünmeyeceği şekilde uzun olması gerekir. Blok uzunluğu büyüdükçe sistemin uygulaması da daha karışık hale gelmektedir.

Anahtar ve gerçek anahtar uzunluğu : Bir blok şifre sisteminin anahtarı deneme yanılma ile bulunamamalıdır. Bunu sağlamak için anahtar uzun olmalıdır. DES anahtar uzunluğunun kısa olmasıyla bilinirdi fakat DES'in gerçek anahtar uzunluğu 128 bite çıkarıldı.

Blok şifreleme sistemlerinin tasarım ölçütleri:

Güvenli bir blok şifre sisteminin kırılması zor ama uygulaması kolay olmalıdır. Şifreleme ve deşifreleme fonksiyonlarının kolay uygulanabilir olması gerekirken, C=EK(M) ve M=DK(C) eşitliklerinden K yı bulmanın zor olması gerekir. İlk defa Claude Shannon tarafından önerilen tasarım ölçütleri yayılma (confusion) ve nüfuz etmedir (diffusion).

Yayılma : Bir blok şifre sistemini ya da genel olarak bir şifreleme sistemini yayılma ölçütüne göre tasarlamak demek, şifreli metinle anahtar arasındaki ilişkiyi mümkün olduğunca karışık yapmaktır.

Daha açık bir tanım verirsek, yayılma, anahtarın açık ve şifreli metne bağlılığının kriptanaliz için faydalı olmayacak kadar karışık olması demektir. Yani blok şifre sistemini tanımlayan eşitliklerin doğrusal olmaması; karışık olması ve böylece C=E(P,k) denkleminden anahtarı bulmanın imkansız olması gerekir.

Nüfuz etme : Bu ölçüte göre her anahtar için şifreleme fonksiyonu öyle olmalı ki, açık metin ve şifreli metin yapıları arasında istatistiksel bağlılık olmamalıdır. Bu ölçütün olabilmesi için anahtarın ve açık metinin her bitinin şifreli metini etkilemesi gerekir.

2.3.ASİMETRİK ŞİFRELEME ALGORİTMALARI

Bu algoritma şifreleme için kullanılan anahtarın çözümleme için kullanılacak anahtardan farklı olması için tasarlanmıştır. Bu algoritmalara genel anahtar denir. Çünkü şifreleme anahtarı kamuya açılabilir.

Rasgele anahtar hakiki mesajı simetrik bir algoritma kullanarak şifrelemek için kullanılır. Bu bazen hibrid (melez) şifreleme olarak adlandırılır. En çok çalışılan ve en yaygın simetrik DES'tir.

Günümüzde 3 çeşit farklı anahtar kriptoloji tekniği bulunmaktadır.

-Büyük sayıları çarpanlarına ayırma ilkeli tabanlı sistemler: RSA
-Ayrık logaritma sistemleri: Diffie Hellman Açık Anahtar Kriptoloji Algoritması
-Eliptik eğri ayrık logaritma sistemleri: Eliptik Eğri Kriptoloji Teknikleri

Uzun yıllardır şifreleme ürünleri donanımsal olarak gerçeklenmekte iken artık yazılımsal ürünler de kullanılmaktadır. Hızdan dolayı askeri firmalar donanım uygulamalarını tercih etmektedir.

Dezavantajları,
Hızı
Maliyeti
Kolay değiştirilebilir olmasıdır.

Avantajları,
Esneklik
Taşınabilirlik
Kolaylıkla kullanılabilmesi
Yenilenebilir olması
Daha geniş uygulamaların içine yerleştirilebilir olmasıdır.

2.4.SİMETRİK VE ASİMETRİK KRİPTOLAMANIN KARŞILAŞTIRILMASI

1. Simetrik anahtar kriptolama genellikle çok yüksek veri taşıma oranı düşünülerek tasarlanır. Bazı donanımsal uygulamalar saniyede yüzlerce megabyte veriyi kriptolamayı başarabilirken, yazılımsal uygulamalarda saniyede megabyte'lar düzeyinde gerçekleşir. Açık anahtar kriptolamada (asimetrik kriptolama) işlemler simetrik anahtar kriptolamaya oranla daha düşük hızda gerçekleşir.

2. Simetrik anahtar kriptolamada anahtar uzunluğu, açık anahtar kriptolamaya göre nispeten kısadır.

3. Simetrik anahtar kriptolamada, iki farklı taraf arasında gerçekleşen iletişim için anahtarı her iki tarafında bilmesi ve gizli tutması zorunlu iken, açık anahtar kriptolamada tarafların sadece kendilerine ait özel anahtarı gizli tutmaları yeterlidir.

4. Simetrik anahtar kriptolamada anahtarın güvenlik açısından sık sık değiştirilmesi gerekirken, açık anahtar kriptolamada özel/genel anahtar çiftinin uzun süreler boyunca değiştirilmesine gerek duyulmaz.

5. Çoğu açık anahtar yapılarında, simetrik anahtar yapılarına göre oldukça verimli dijital imza mekanizmaları elde edilir.

3.MALZEME VE YÖNTEM

3.1.DES - DATA ENCRYPTION STANDART

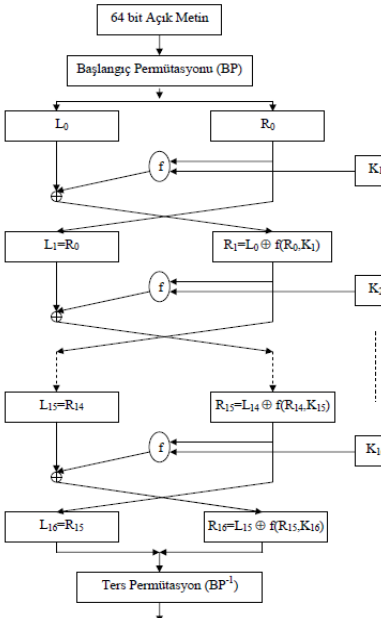
IBM tarafından 1970 lerde geliştirilmiştir. DES veri şifrelemek (encryption) ve şifrelenmiş verileri açmak (decryption) için geliştirilmiş bir standarttır.

DES, 64-bit bloklarda veriyi şifreleyen bir blok şifredir. Yani basitçe şifrelenecek olan açık metni (plain text) parçalara bölerek (blok) her parçayı birbirinden bağımsız olarak şifreler ve şifrenmiş metni (cipher text) açmak içinde aynı işlemi bloklar üzerinde yapar. Bu blokların uzunluğu 64 bittir. DES simetrik bir algoritmadır. Aynı algoritma ve anahtar hem şifrelemede hem de deşifrelemede (şifre çözme) kullanılır.

Algoritma, mesajı hem şifrelemek hem de çözmek için kullanılmaktadır. İki tane 64 bitlik giriş veri bloklarını alır. Biri orijinal veri diğeri ise şifreleme anahtarı bloğudur. Algoritma orijinal veriyi şifrelenmiş veriye ya da şifrelenmiş veriyi orijinal veriye dönüştürür. 64 bitlik şifreleme anahtar bloğunun 56 biti direkt olarak algoritmaya girer. Kalan 8 bit tek eşlenik kontrolü (odd parity check) için kullanılır. Algoritmanın güvenliği anahtara bağlı olduğundan 56 bitlik anahtar değiştirilebilir. Algoritma permütasyon, yerine koyma (substitution) ve mod 2 (XOR) işlemlerine dayanmaktadır. Algoritmanın 3 çeşidi vardır.

Düz permütasyon
Genişletme permütasyonu
Permütasyonlu seçenekler dir.

Düz permütasyonda, bitler basit bir şekilde yeniden düzenlenir. Genişletme permütasyonunda, bazı bitler ikişer kere kullanılır ve yeniden düzenlenir. Permütasyonlu seçeneklerde ise bitler ihmal edilir ve kalanlar tekrar düzenlenir. DES'te yerine koyma işlemleri S-kutuları adı verilen birbirinden farklı tablodan oluşur. DES'te S-kutularının girişleri 6 bit, çıkışları ise 4 bit'tir.



Şekil 3.1 Des Algoritma Yapısı

DES 64-bit blok üzerinde açık metinleri işler. 64 bitlik veri bloğunu alır ve başlangıç permütasyonu (initial permütasyon) işleminden sonra 32 bitlik sağ ve sol yarılara ayırır. Sonra verinin anahtar ve f fonksiyonu ile birleştirildiği 16 döngülük işlemler gerçekleştirilir. 16. döngüden sonra sağ ve sol yarı tekrar biraraya getirilir. Döngülerdeki işlemler XOR işlemidir.

Başlangıç permütasyonu (BP)

64 bitlik veri bloğu alındıktan sonra Başlangıç Permütasyonu işlemi uygulanır. Bu permütasyonda tabloya göre bitler sırayla karıştırılır. Tablo soldan sağa, yukarıdan aşağıya doğru okunur. Tabloya göre 58. bit çıkışta 1.bit, 50.bit'te çıkışta 2.bit vb. olarak çıkmaktadır.

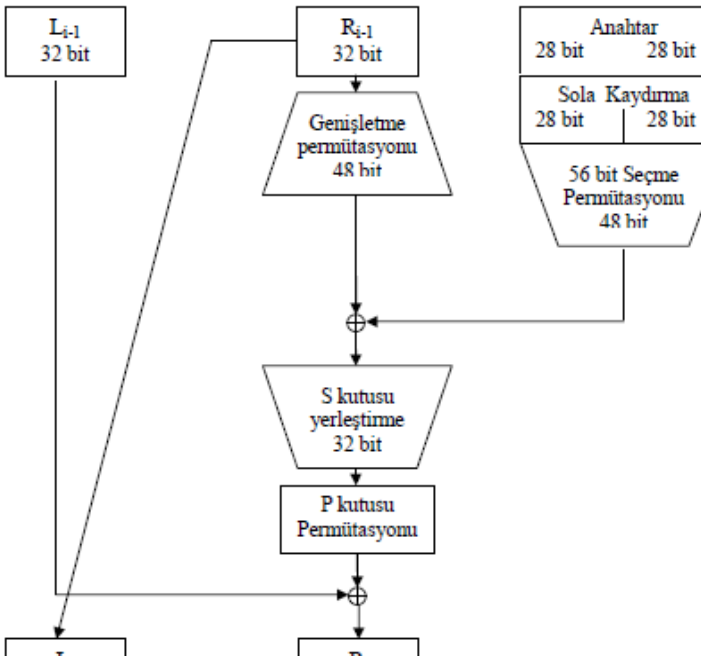
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Her döngüde anahtar (şifreleme ise sola, çözme ise sağa) kaydırılır ve 56 bitin 48'i seçilir. Sağ yarıdan gelen 32 bit, genişletilmiş permütasyon yardımı ile 48 bite dönüştürülür ve 48 bitlik anahtar ile XOR'lanır. Sonuç 8 tane S-kutusuna gönderilir. Çıkışta 32 bit üretilir ve çıkış bitlerine P permütasyonu uygulanır. Bu dört işlem f fonksiyonunu oluşturur.

F fonksiyonunun çıktısı soldan gelen 32 bitlik veri ile XOR'lanır. Bu işlemin sonucu yeni sağ yarıyı, eski sağ yarı da yeni sol yarıyı oluşturur. Bu işlemler 16 kez tekrarlanarak DES'in 16 döngüsü gerçekleştirilmiş olur.

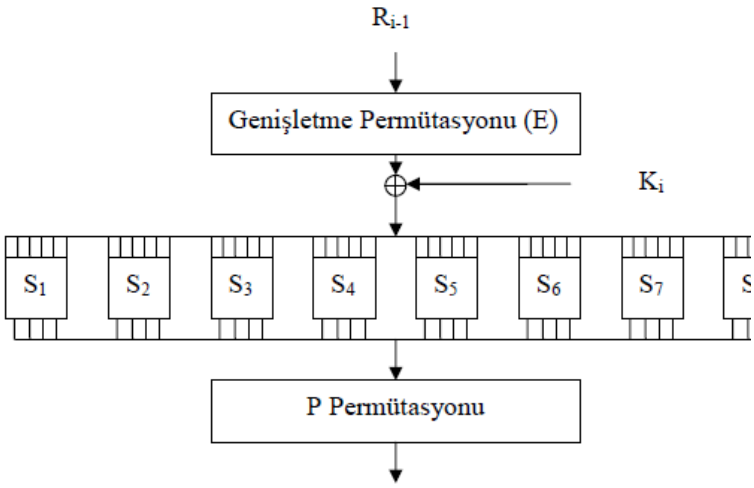
$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$



Şekil 3.2 Des Başlangıç,Genişletme ve P Kutusu Permütasyonu

Her sağ yarı genişletme permütasyonu yardımı ile 32 bitten 48 bite genişletilir.



Şekil 3.3 S Kutusu Yerleşimi

DES TEKİ f fonksiyonu detayı

Alttaki tablo girişteki bitlerin çıkışta hangi sırada çıktığını gösterir. Örneğin 12.bit 4 sırada

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

GENİŞLETME PERMÜTASYONU
S KUTULARI:

Yerleştirme işlemi 8 tane S- kutusu kullanılarak gerçekleştirilir. S- kutusu 6 bitlik verinin 4 bit olarak çıkmasını sağlayan bir tablodur. Genişletme permütasyonundan çıkan 48 bitlik veri, 48 bitlik anahtar ile XOR'landıktan sonra S1S2.....S8 olmak üzere 8 tane 6 bitlik bloklara ayrılır. Her S-kutusunda 4 satır ve 16 sütun bulunmaktadır. 6 bitlik giriş verisinin birinci ve altıncı bitlerinin oluşturduğu 2 bitlik sayı, satır numarasını verirken arada kalan 4 bitlik sayı sütun numarasını verir. Satır ve sütun numarasının kesişme noktasındaki veri, çıkışta görülecek olan 4 bitlik veridir. Örneğin S7 bloğuna girecek olan veri 010011 ise satır numarası 01, yani 1'dir. Sütun numarası ise 1001, yani 9'dur. Birinci satır ve dokuzuncu sütundaki veri 3'tür. Dolayısı ile çıkışta 0011 verisi görülür.

S kutusu kritik öneme sahiptir ve doğrusal olmadığı için algoritmaya güvenlik sağlar.

P KUTUSU PERMÜTASYONU:

S- kutusundan çıkan veriler P adı verilen permütasyona tabi tutulur. Bu tabloya göre 16. bit 1. bit olarak çıkarken, 7. bit 2. bit vb olarak çıkar.

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	31	10
2	8	24	14
19	13	30	6

S kutuları içeriğinde ;

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	
S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	
S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	
S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	
S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	
S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	
S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	
S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	

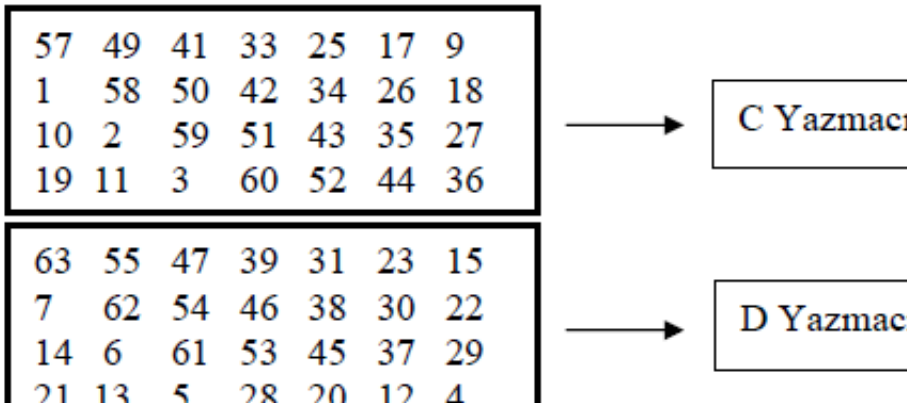
Ters Permütasyon:

Yukarıda anlatılan islemler 16 kez gerçeklestirildikten sonra baslangıç permütasyonu islemini tersi olan ters (son) permütasyonu asagıda verilen sekile göre gerçeklestirilir. Bu tabloya göre 40. bit 1. bit, 8. bit 2. bit vb. olarak çıkar.

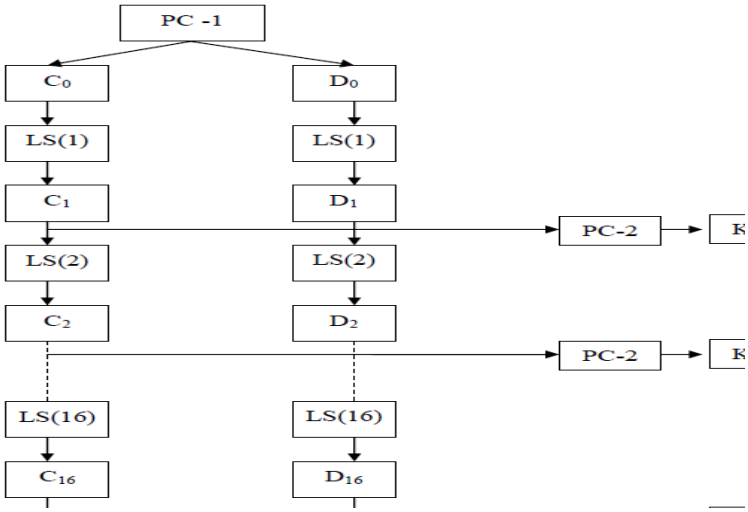
40	8	48	16	56	24	64	3
39	7	47	15	55	23	63	3
38	6	46	14	54	22	62	3
37	5	45	13	53	21	61	2
36	4	44	12	52	20	60	2
35	3	43	11	51	19	59	2
34	2	42	10	50	18	58	2
33	1	41	9	49	17	57	2

Anahtar Oluşturma:

64 bitlik anahtar bloğu Permütasyonlu Seçenek 1 (PC1)'e tabi tutulur. Elde edilen veri bloğu başlangıç permütasyonunda olduğu gibi 56 bittir. 56 bitlik blok, 2 bitlik iki bloğa ayrılır. Sonra C ve D yazmaçlarına yerleştirilir. Bu tabloya göre C yazmacının 1. biti 57.bit,
D yazmacının 1. biti ise 63. bittir.



Permütasyonlu Seçenek PC1



Şekil 3.4 Des Algoritmasında Permütasyonlu Seçenek Ve Döngüler

PC-1: Permuted Choice 1

Bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**
1	57	49	41	33	25	17	9
8	1	58	50	42	34	26	18
15	10	2	59	51	43	35	27
22	19	11	3	60	52	44	36
29	63	55	47	39	31	23	15
36	7	62	54	46	38	30	22
43	14	6	61	53	45	37	29
50	21	13	5	28	20	12	4

Burada 64 bitten 56 bit geçişini yaparken bulunduğu satır ve sütun toplanır. Örneğin 30 için 36 ile 5 toplanır. Böylece yeni 56 bit lik key in 41.biti olmuş olur.

C ve D yazmaçlarındaki gerekli sola kaydırma işlemleri gerçekleştirildikten sonra bir araya getirilen C ve D blokları Permütasyonlu Seçenek 2 (PC2)'ye tabi tutulur ve bit büyüklügü 48'e indirilir. Her döngüde gerçekleştirilen PC2 permütasyonunun sonucu ana döngüde kullanılacak olan her bir "alt anahtarı" verir.

Döngü	*Anahtar (K)*	*PC2'den Önce Sola Kaydırma Sayısı (Şifrelemede)*	*PC2'den Önce Sağ Kaydırma Sayısı (Şifre çözmede)*
1	K_1	1	0
2	K_2	1	1
3	K_3	2	2
4	K_4	2	2
5	K_5	2	2
6	K_6	2	2
7	K_7	2	2
8	K_8	2	2
9	K_9	1	1
10	K_{10}	2	2
11	K_{11}	2	2
12	K_{12}	2	2
13	K_{13}	2	2
14	K_{14}	2	2
15	K_{15}	2	2

Permütasyonlu Seçenek 2 PC2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Tabloya göre 14.bit 1.bit olarak çıkar.

ŞİFRE ÇÖZME:

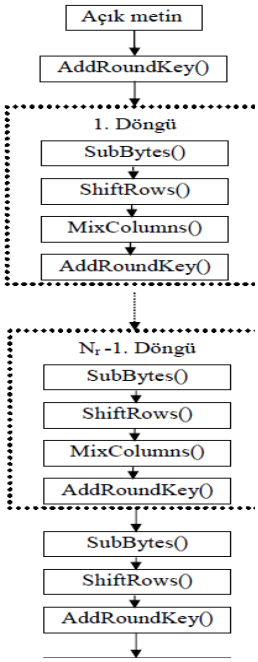
DES algoritmasında, çözme işlemi için kullanılacak algoritma, şifrelemek için kullanılan algoritmanın aynısıdır. Tek fark, çözme işleminde kullanılacak olan alt anahtarların aynı zamanda şifreleme anahtarlarının ters sıradaki değerleridir. Yani K1, şifreleme alt anahtarının birincisiyse çözme anahtarlarının on altıncısıdır.

3.2.AES (GELİŞMİŞ ŞİFRELEME STANDARDI)

Des algoritmasının anahtar uzunluğu 56, blok uzunluğu 64 bittir. Algoritma teknolojisindeki gelişmelere paralel DES algoritmasının arka arkaya çalıştırılması anlamına gelen 3DES algoritması ortaya çıkmıştır. Bu yöntem ile algoritmanın anahtar uzunluğu 112 bite çıkmakla beraber blok uzunluğu 64 bitte kalmaktadır. Fakat bu algoritmanın yavaş olmasından dolayı AES algoritması ortaya çıkmıştır.

AES (Rijndael) algoritması degişik blok boyunu ve değişik anahtar büyüklüğünü destekleyen şifreleme algoritmasıdır. AES 128, 192 ve 256 bit uzunlukta anahtar boyutunu ve blok uzunluğunu desteklemektedir. Kriptanaliz alanındaki çalışmalardan dolayı 128 bit standart olarak kabul edilmiştir. Standart 128 bit blok boyunu ve 128, 192, 256 bit anahtar uzunluğunu içermektedir.

Ana akış şeması;



Şekil 3.5 Aes Algoritma Döngüsü

AES'te döngü sayısı anahtar uzunluğuna göre değişmektedir. 128 bit anahtar için 10 döngüde şifreleme yaparken 192 ve 256 bit anahtar için sırasıyla 12 ve 14 döngüde şifreleme yapmaktadır. Her döngüde 4 katman vardır. Bunlar sırasıyla Subbytes(), Shiftrows(), MixColumns() ve AddRoundKey() işlemleridir. Her döngünün çıktısı bir sonrakinin giriş değeridir. Nr değeri döngü sayısını ifade eder. Buda 10,12,14 (128,192,256 bit anahtar için) değerlerinden biridir. AES te en küçük işlem birimi byte dır. Şifrelenecek metin, şifrelenmiş metin ve anahtar bilgileri bayt dizileri olarak kabul edilirler. Bu diziler 4 satır ve Nb adet sütündan oluşur ve her bir hücre baytlık bilgi tutar.

Metin 4 baytlık sütün vektörleri şeklinde, yani 128 bit için 4x4 (Nb= 4), 192 bit için 4x6 (Nb= 6) ve 256 bit içinde 4x8'lik (Nb= 8) matrislerle ifade edilir.

Subbytes Fonksiyonu (Baytların yerdeğiştirmesi)

S kutusunun olduğu katmandır. Giriş matris bilgisini alıp, her bir baytı tanımlanmış bir S kutusundan geçirerek, bu baytın karşılığı olan S kutusu değeri ile değiştirir. Baytların yer değiştirilmesinde 16 bayt değerinin her biri 8 bit girişli ve 8 bit çıkışlı S kutusuna sokulur.

ShiftRows Fonksiyonu (Satırların ötelenmesi)

Bu fonksiyon matrisi alır ve son üç satırını belli değerlere göre dairesel olarak sola öteler. Nb = 4 olmak üzere öteleme miktarı 2. satır için 1, 3. satır için 2, 4. satır için 3'tür.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

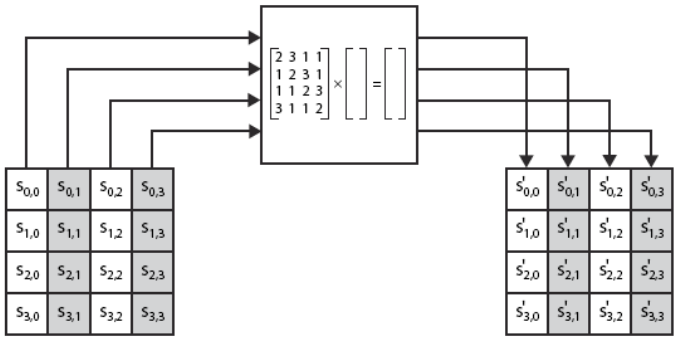
Öteleme öncesi

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,1}$	$S_{1,2}$	$S_{1,3}$	$S_{1,0}$
$S_{2,2}$	$S_{2,3}$	$S_{2,0}$	$S_{2,1}$
$S_{3,3}$	$S_{3,0}$	$S_{3,1}$	$S_{3,2}$

Öteleme sonrası

MixColumns Fonksiyonu (Sütunların karıştırılması)

Bu fonksiyon matris üzerinde sütün bazında çalışır. Her sütun GF(28)'de 4 terimli bir polinom olarak kabul edilerek sabit bir a(x) = {03}x3 + {01}x2 + {01}x + {02} polinomu ile çarpılır ve elde edilen polinom başlangıç sütununun yerine geçer.



AddRoundKey Fonksiyonu (Anahtar ekleme)

Bu fonksiyonda, bir döngü anahtarı matrise xor işlemiyle eklenir. Her döngü anahtarı Nb kelimeden oluşmaktadır. Nb = 4 için döngü anahtarının boyu 128 bittir.

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{1} \\ k_{20} & k_{21} & k_{22} & k_{2} \\ k_{30} & k_{31} & k_{32} & k_{3} \end{bmatrix}$$

$$= \begin{bmatrix} a_{00} \oplus k_{00} & a_{01} \oplus k_{01} & a_{02} \oplus k_{02} & a_{03} \oplus k_{03} \\ a_{10} \oplus k_{10} & a_{11} \oplus k_{11} & a_{12} \oplus k_{12} & a_{13} \oplus k_{13} \\ a_{20} \oplus k_{20} & a_{21} \oplus k_{21} & a_{22} \oplus k_{22} & a_{23} \oplus k_{23} \\ a_{30} \oplus k_{30} & a_{31} \oplus k_{31} & a_{32} \oplus k_{32} & a_{33} \oplus k_{33} \end{bmatrix}$$

AES (Rijndael) algoritması hem donanım hem de yazılım uygulamalarında iyi performans vermiştir. Tasarımında kullanılan S kutusu sonlu alanda ters alma işleminin kullanılması ile gerçekleştirilmiştir.

Kod çözme adımında shiftRows fonksiyonu diğer yöne sağa ötelenir, mixcolumns fonksiyonunda ters matris işleme sokulur. Kısacası her adımın tersi farklı anahtar kullanılarak uygulanır.

3.3.RSA ŞİFRELEME ALGORİTMASI:

Bir genel anahtarlı şifreleme tekniği olan RSA, çok büyük tamsayıları oluşturma ve bu sayıları işleminin zorluğu üzerine düşünülmüştür. Anahtar oluşturma işlemi için asal sayılar kullanılarak daha güvenli bir yapı oluşturulmuştur. Anahtar oluşturma algoritması şu şekildedir.

• P ve Q gibi çok büyük iki asal sayı seçilir.
• Bu iki asal sayının çarpımı N = P.Q ve bu bir eksiklerinin φ(N)=(P-1)(Q-1) hesaplanır.
• 1'den büyük φ(N)'den küçük φ(N) ile aralarında asal bir E tamsayısı seçilir.
• Seçilen E tamsayısının mod φ(N)'de tersi alınır, sonuç D gibi bir tamsayıdır.
• E ve N tamsayıları genel anahtarı, D ve N tamsayıları ise özel anahtarı oluşturur.

Genel ve özel anahtarları oluşturduktan sonra gönderilmek istenen bilgi genel anahtar ile şifrelenir. Şifreleme işlemi şu şekilde yapılmaktadır. Şifrelenecek bilginin sayısal karşılığının E' ninci kuvveti alınır ve bunun mod N deki karşılığı şifrelenmiş metni oluşturmaktadır. Genel anahtar ile şifrelenmiş bir metin ancak özel anahtar ile açılabilir. Bu yüzden şifrelenmiş metin, yine aynı yolla şifrelenmiş metnin sayısal karşılığının D'ninci kuvveti alınır ve bunun mod N deki karşılığı orijinal metni oluşturur.

Basit bir örnek ile algoritmayı tekrar anlatırsak;
Öncelikle genel ve özel anahtarları oluşturmak gerekiyor.

• P=7 ve Q=19 gibi iki asal sayı seçelim.
• Bu iki asal sayının çarpımı N=P.Q; N=133 ve bu iki asal sayının bir eksiklerinin çarpımı φ(N)=(P-1)(Q-1); φ(N)=108 olarak hesaplanır.
• 1'den büyük 108'den küçük 108 ile aralarında asal bir E=5 tamsayısı seçelim.
• Seçilen E=5 tamsayısının mod 108'da tersi alınır, sonuç D=43 gibi bir tamsayıdır.
• 5 ve 133 tamsayıları genel anahtarı, 43 ve 108 tamsayıları ise özel anahtarı oluşturur.

Bu algoritmada iki asal sayının çarpımını kullanarak anahtar oluşturulmasının sebebi, iki asal sayının çarpımını asal çarpanlarına ayırmak asal olmayan sayıları ayırmaktan daha zorlu olmasıdır. Şimdi oluşturduğumuz {5, 119} ve {77, 119} anahtarlarımızı kullanarak şifreleme yapalım. Örnek olarak, 21 sayısını genel anahtarımızla {5, 133} şifreleyelim. 21 sayısının 5'inci kuvvetinin mod 133 deki karşılığı olan 70, 21 sayısının RSA şifrelenmiş halidir.

Avantajları

- Simetrik şifreleme, şifrelenmiş veriyi alan tarafın veriyi deşifre edebilmesi için, gizli anahtar paylaşımını gerekli kılar. Ancak RSA asimetrik bir şifreleme tekniği olduğu için gizli anahtarın paylaşılmasına gerek yoktur. Kullanıcıların gizli anahtarlarının saklanması gerekmez. Bu da sistemi büyük bir depolama yükünden kurtarır.
- Büyük sayılarla işlem yapmak zor olduğu için güvenilirliği son derece yüksek olan bir şifreleme tekniğidir.

Dezavantajları

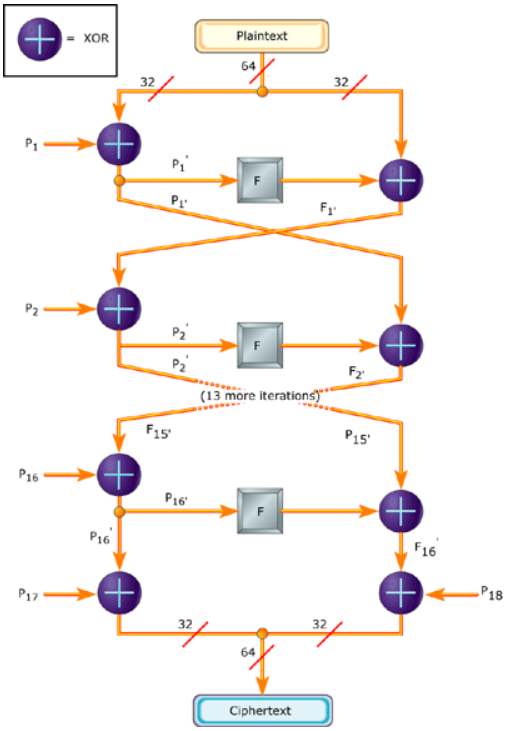
- RSA algoritmasının en büyük dezavantajı, asimetrik bir şifreleme algoritması olması ve büyük sayılarla işlem yapması nedeniyle yavaş olmasıdır.
- Özellikle kablosuz ağ sistemlerinde bu algoritmanın kullanılması bazı sorunlara yol açabilir. Çünkü bant genişliğini fazlaca tüketir ve sistemi yavaşlatarak performans düşüşüne neden olur.

.NET teknolojisi ile rsa şifreleme

.NET teknolojisi ile herhangi bir veride şifreleme yapmak için .NET Framework içerisinde yer alan **System.Security.Cryptography** kütüphanesini kullanılır. Yazılım geliştiricinin RSA algoritmasını kullanarak şifreleme yapabilmesi için **RSACryptoServiceProvider** sınıfını kullanması gerekmektedir.

3.4.BLOWFISH ALGORİTMASI :

Blowfish algoritması Bruce Schneier tarafından 1993 yılında dizayn edilmiştir.Blowfish'te veriler 64 bitlik bloklar halinde şifrelenir ve 32 bitlik 18 adet alt anahtar bulunmaktadır. Anahtar uzunluğu 32 bitten 448 bite kadar olabilir. F fonsiyonu için 4 adet S-Box kullanılır. Veriler, basit bir fonksiyonun 16 kez kullanılmasıyla şifrelenir.Aynı anahtarı hem şifreleme hem de şifre çözmede kullandığı için simetrik bir algoritmadır. [5]



Şekil 3.6 Blowfish Algoritma Yapısı

Algoritma

Feistel yapısı kullanılır. Pi sayısının kesirli kısmının on altılı sayı sistemi gösterimi de sabit değerler olarak belirlenmiştir. Pi sayısının kullanılma nedeni, basamaklarını oluşturan sayıların rastgelelik özelliği taşımasıdır. Bu sabit değerler, 256 adet 32 bitlik on altılı sayı sistemi değerlerinden oluşmaktadır. Bu sabitler, alt anahtarların ve gizli kutuların oluşturulmasında kullanılacaktır. 64 bitlik açık veri 32 bitlik iki parçaya ayrılır.

Sol taraftaki 32 bitlik blok ile P dizisinin ilk elemanı XOR işlemine girer. Buradan çıkan sonuç P' değeri olur ve F fonksiyonuna gönderilir. F fonksiyonundan dönen değer ile 32 bitlik sağ taraftaki blok XOR işlemine girer. Buradan çıkan sonuç F' değerini alır. Son olarak sol taraftaki P' değeri yeni turda sağ blok, sağ taraftaki F' değeri de sol blok kabul edilerek, aynı işlemler 15 tur daha tekrar edilir.

Sonuçta P' ve F' ler P dizisinin en son 2 girişine(17. ve 18. giriş) kadar fonksiyona değer olarak gönderilir.

Son turda yer değiştirme işlemi gerçekleşmez. P(16)' değeri ile P(17) değeri XOR işlemine sokulur. F(16)' değeri ile de P(18) değeri XOR işlemine sokulur.

Son olarak sol taraftaki 32 bit veri ile sağ taraftaki 32 bit veri birleştirilerek 64 bit şifrelenmiş veri elde edilir.

Döngü Sayısının Bulunması

128 bit uzunluğundaki anahtar, 32 adet 4 bitlik bloklara ayrılır. Bu 4 bitlik bloklar birbirleri ile XOR işlemine sokulur. Çıkan 4 bitlik sayı onluk sayı sistemine çevrilir. Bu tamsayı değerinin çift ve 16-32 aralığında olması için tek ise 17, değilse 16 eklenir. Çıkan sonuç döngü sayısını verir.

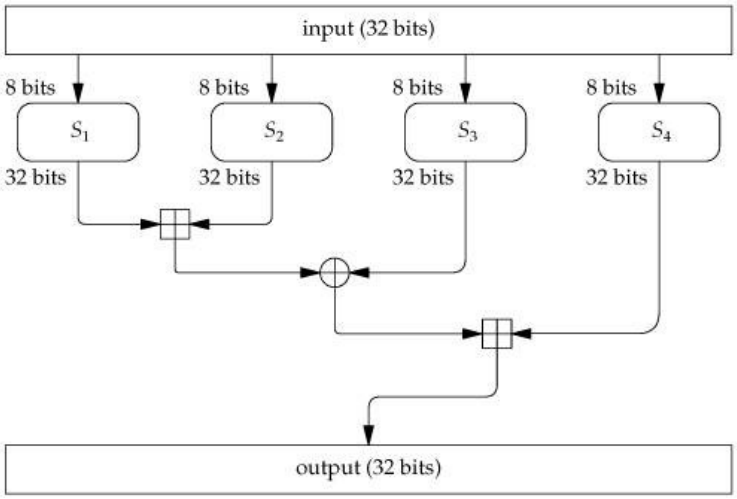
Alt Anahtarların Bulunması

Blowfish algoritması 18 farklı alt anahtar (P) kullanmaktadır.

128 bitlik gizli anahtarımız 32 bitlik olmak üzere 4 farklı blok haline getirilir. Bunlara (K1,K2,K3,K4) diyelim. K dizisi elemanları sırası ile P dizisi elemanları ile XOR lanır. Bu işlemden çıkan sonuçlar ise yine P dizisi elemanlarını ifade eder.

Daha sonra sadece '0' lardan oluşan bir metin(T) Blowfish algoritmasına sokulur. Bu algoritma sonucunda gelen 64 bitlik metin (T) 32 bit, 32 bit olmak üzere 2 parçaya ayrılır. İlk 32 bitlik dizi P(1) in, ikinci 32 bitlik dizide P(2) nin yeni değerleridir.

T metni tekrar algoritmaya gönderilir fakat bu sefer algoritmada yeni P elemanları kullanılır, algoritmadan dönen değerler P dizisinin 3. ve 4. elemanlarıdırlar.
Bu işlem P dizisi tamamlanıncaya kadar (P=18) tekrarlanır.



Şekil 3.7 Blowfish Algoritması Alt Anahtarların Bulunması

Gizli Kutuların Bulunması (S-Box)

İlk gizli kutunun bulunması;
1. 128 bitlik anahtar 4 adet 32 bitlik bloklara ayrılır; *k0, k1, k2, k3*
2. İlk gizli kutunun ilk elemanını oluşturmak için 32 bitlik anahtar blokları ve P0 toplanır. Böylece ilk gizli kutunun ilk elemanı bulunmuş olur.
3. SK0 alt anahtarının ilk 32 biti ile ikinci 32 biti çarpılır.
4. Bir önceki çıkan gizli kutu elemanı ile *P1* çarpılır.
5. 3 nolu adımdan ve 4 nolu adımdan çıkan sonuçlar toplanır.
6. Bu şekilde 255 kere döngü devam eder, her eleman bir önceki elemanı kullanarak bulunur. Her döngüde *P* indisi bir artar. Son alt anahtar kullanıldığı zaman, alt anahtar tekrar SK0'dan itibaren kullanılmaya başlanır

İkinci gizli kutunun bulunması;
1. Bir önceki gizli kutunun son elemanı ile k1 ve *P0* XOR işlemine sokulur. Çıkan sonuç ikinci gizli kutunun ilk elemanıdır.
İlk kutu oluşturulurken kullanılan 2. adımdan devam edilir.

Üçüncü ve dördündü gizli kutunun oluşturulması;
Üçüncü ve dördüncü gizli kutuların oluşturulmasında tek fark, ilk elemanlardır. Üçüncü gizli kutuda ilk eleman, ikinci kutunun son elemanı, *k2* ve *P0* XOR işlemine sokularak bulunur. Dördüncü gizli kutuda ilk eleman, üçüncü kutunun son elemanı, *k3* ve *P0* XOR işlemine sokularak bulunur.

Yukarıdaki işlemleri özetlersek;

Gizli kutular 4 adettir ve 8*32 boyutlarındadır.
İlk değerleri Pi sayısından elde edilen değerleri içerir.
P dizisinde son iki elemanı (T) yine algoritmaya gönderilir. Buradan dönen sonuç 1. gizli kutunun ilk iki elemanını ifade eder.(S(0,0) ve S(0,1))
İlk elemanı oluşturan metin (T) tekrar algoritmaya gönderilir bu sefer algoritmada yeni S değerleri kullanılır ve bu işlem sonucunda dönen değer 3. ve 4. elemanları oluşturur.
Bu işlem 1. kutu tamamlanıncaya kadar devam eder.

Bu döngü geriye kalan diğer 3 gizli kutu içinde gerçekleştirilir. Tek farkı ilk gönderilecek metin (T) bir önceki kutunun son iki elemanın değerleridir.

Tüm bu döngü işlemleri sonunda pi sayısından elde edilen geçici değerlerin yerini asıl değerler alacaktır.

Deşifreleme

Blowfish algoritmasında deşifreleme işlemi, şifreleme işleminin tamamen aynısının adım adım tersten uygulanmış halidir.

Deşifreleme algoritması Blowfish algoritmasının alt anahtarlarının tersten kullanılmış şeklidir. Yani deşifreleme algoritmasındaki P(1) değeri şifreleme algoritmasındaki P(18) değerine eşittir.

3.5. VPN UYGULAMALARINDA ŞİFRELEME:

Vpn Nedir?

Sanal özel ağlar (VPN), özel veya Internet gibi ortak ağlar üzerindeki noktadan noktaya bağlantılardır. VPN istemcisi, VPN sunucusu üzerindeki sanal bir bağlantı noktasına sanal bir arama gerçekleştirmek için, tünel protokolleri adı verilen özel TCP/IP tabanlı protokolleri kullanır. Tipik bir VPN dağıtımında istemci, Internet üzerinden uzaktan erişim sunucusuyla sanal noktadan noktaya bağlantı başlatır. Uzaktan erişim sunucusu aramaya yanıt verir, arayanın kimliğini doğrular, verileri VPN istemcisi ile kuruluşun özel ağı arasında aktarır.

Veriler, noktadan noktaya bağlantıyı taklit etmek amacıyla üstbilgi kullanılarak kapsüllenir veya sarılır. Özel ağ bağlantısını taklit etmek için, gönderilen veriler gizlilik amacıyla şifrelenir. Paylaşılan veya ortak ağda ele geçirilen paketlerin şifreleri, şifreleme anahtarları olmadan çözülemez. Özel ağ verilerinin kapsüllendiği ve şifrelendiği bağlantı VPN bağlantısı olarak bilinir.
2 çeşit vpn vardır.

Uzaktan erişim VPN

Uzaktan erişim VPN bağlantıları, evinden çalışan veya yolda olan kullanıcıların Internet gibi ortak bir ağ tarafından sağlanan altyapıyı kullanarak özel ağ üzerindeki bir sunucuya erişmelerine olanak sağlar.

Siteden siteye VPN

Siteden siteye VPN bağlantıları (yönlendiriciden yönlendiriciye VPN bağlantıları olarak da bilinir), kuruluşların farklı ofisler arasında veya diğer kuruluşlarla ortak bir ağ üzerinden yönlendirilmiş bağlantılar kullanabilmelerine olanak verirken, iletişim güvenliğinin sağlanmasına da yardım eder. Siteden siteye VPN bağlantısı özel bir ağın iki bölümünü birbirine bağlar. VPN sunucusu, bağlı bulunduğu ağa yönlendirilmiş bağlantı sağlar. Yanıtlayan yönlendirici (VPN sunucusu) arayan yönlendiricinin (VPN istemcisi) kimliğini doğrular ve karşılıklı kimlik doğrulama amacıyla, arayan yönlendirici de yanıtlayan yönlendiricinin kimliğini doğrular.

VPN bağlantılarının özellikleri:

PPTP, L2TP/IPsec ve SSTP kullanan VPN bağlantıları aşağıdaki özelliklere sahiptir.

*Kapsülleme

*Kimlik doğrulama

*Veri şifreleme

Kapsülleme:

VPN teknolojisinde özel veriler, geçiş ağını çapraz geçmelerine izin verecek yönlendirme bilgilerini içeren bir üstbilgiyle kapsüllenir.

VPN bağlantılarında kimlik doğrulama üç farklı biçimde yapılır:

1.PPP kimlik doğrulama kullanılarak kullanıcı düzeyinde kimlik doğrulama

VPN bağlantısı oluşturmak için, VPN sunucusu bağlanmayı deneyen VPN istemcisinin kimliğini, Noktadan Noktaya Protokolü (PPP) kullanıcı düzeyinde kimlik doğrulama yöntemi kullanarak doğrular ve VPN istemcisinin uygun yetkilendirmeye sahip olduğunu onaylar.

2.Internet Anahtar Değişimi (IKE) kullanarak bilgisayar düzeyinde kimlik doğrulama

Internet Protokolü güvenliği (IPsec) güvenlik ilişkisi oluşturmak üzere VPN istemcisi ve VPN sunucusu, bilgisayar sertifikaları veya önceden paylaşılan bir anahtar değişimi için IKE protokolünü kullanır. Her iki durumda da VPN istemcisi ve sunucusu, birbirlerinin kimliklerini bilgisayar düzeyinde doğrular. Bilgisayar sertifikası kimlik doğrulaması çok daha güçlü bir kimlik doğrulama yöntemi olduğundan daha fazla önerilir. Bilgisayar düzeyinde kimlik doğrulama yalnızca L2TP/IPsec bağlantıları için uygulanır.

3.Veri kaynağı için kimlik doğrulama ve veri bütünlüğü

VPN bağlantısı üzerinden gönderilen verinin, bağlantının diğer ucundan gönderilmiş olduğunu ve aktarım sırasında değiştirilmediğini onaylamak için, veride yalnızca gönderenin ve alanın bildiği bir şifreleme anahtarına dayalı şifreleme sağlama toplamı bulunur.

Veri şifreleme

Veriler, paylaşılan veya ortak geçiş ağından çapraz geçerken gizliliğinin sağlanması amacıyla gönderen tarafından şifrelenir ve şifreleri alan tarafından çözülür. Şifreleme ve şifre çözme işlemleri gönderenin ve alanın ortak kullandığı bir şifreleme anahtarına bağlıdır. Şifreleme anahtarının uzunluğu çok önemli bir güvenlik parametresidir.

Vpn Tunel Protokolleri

Tünel oluşturma, bir protokol türündeki paketin başka bir protokol datagramı içinde kapsüllenmesini sağlar. Örneğin VPN, IP paketlerini Internet gibi ortak bir ağ üzerinden kapsüllemek için PPTP'yi kullanır. Noktadan Noktaya Tünel Protokolü (PPTP), Katman İki Tünel Protokolü (L2TP) veya Güvenli Yuva Tünel Protokolü'ne (SSTP) dayalı bir VPN çözümü yapılandırılabilir.

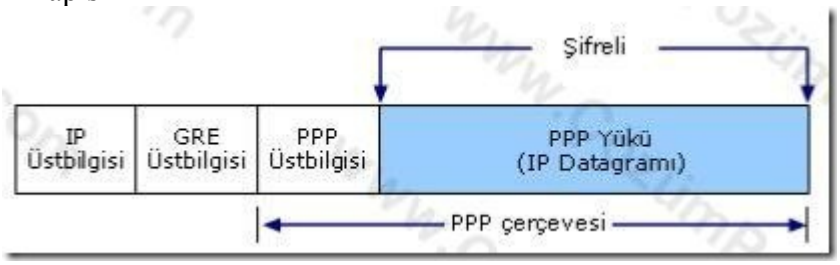
PPTP

PPTP, birden çok protokol trafiğinin şifrelenmesini ve ardından IP ağı veya Internet gibi ortak IP ağı üzerinden gönderilmek üzere bir IP üstbilgisi ile kapsüllenmesini sağlar. PPTP uzaktan erişim ve siteden siteye VPN bağlantıları için kullanılabilir. Internet, VPN için ortak ağ olarak kullanıldığında, PPTP sunucusu, biri Internet üzerinde diğeri de intranet'te bulunan iki arabirime sahip PPTP etkin bir VPN sunucusudur.

Kapsülleme

PPTP, ağ üzerinden aktarım için PPP çerçevelerini IP datagramları içinde kapsüller. PPTP, tünel yönetimi için bir TCP bağlantısını ve tünel oluşturulan veri için PPP çerçevelerinin kapsüllenmesi amacıyla Genel Yönlendirme Kapsüllemesi'nin (GRE) değiştirilmiş bir sürümünü kullanır. Kapsüllenen PPP çerçevelerinin yükleri şifrelenebilir, sıkıştırılabilir veya her ikisi de uygulanabilir. Aşağıdaki şekilde bir IP datagramını içeren PPTP paketinin yapısı gösterilmektedir.

PPTP Paketinin Yapısı



Şekil 3.8 PPTP Paketinin Yapısı

L2TP

L2TP birden çok protokol trafiğinin şifrelenmesini ve ardından IP veya zaman uyumsuz aktarım modu (ATM) gibi noktadan noktaya datagram teslimini destekleyen herhangi bir medya üzerinden gönderilmesini sağlar. L2TP, Cisco Systems, Inc. tarafından geliştirilen PPTP ve Katman İki İletme (L2F) protokollerinin birleşiminden oluşan bir teknolojidir. L2TP, PPTP ve L2F'nin en iyi özelliklerini alır.

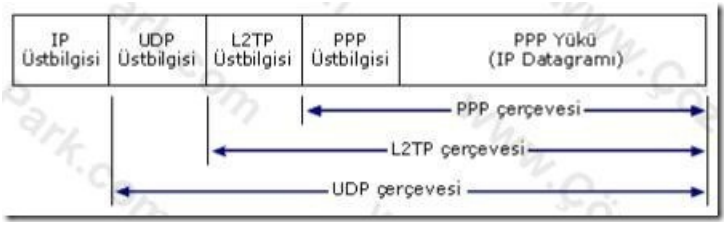
PPTP'nin aksine, Microsoft'un L2TP uygulaması, PPP datagramlarının şifrelenmesinde MPPE'yi kullanmaz. L2TP, şifreleme hizmetleri için Aktarım Modunda Internet Protokolü güvenliğine (IPsec) dayanır. L2TP ve IPsec'in birleşimi L2TP/IPsec olarak bilinir.

Kapsülleme

L2TP/IPsec paketlerinin kapsüllenmesi iki katmandan oluşur.

BİRİNCİ KATMAN:L2TP KAPSÜLLEME

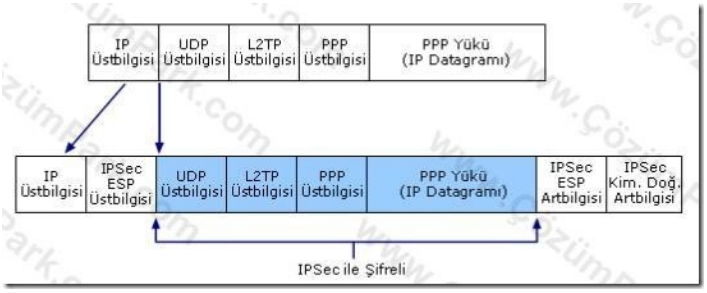
PPP çerçevesi (IP datagramı) L2TP üstbilgisi ve UDP üstbilgisi ile sarılır.



Şekil 3.9 IP Datagramı İçeren L2TP Paketinin Yapısı

İKİNCI KATMAN:IPSEC KAPSÜLLEME

Ortaya çıkan L2TP iletisi daha sonra bir IPsec Kapsüllenen Güvenlik Yükü (ESP) üstbilgisi ve altbilgisi, iletinin bütünlüğünü ve kimlik doğrulamayı sağlayan IPsec Kimlik Doğrulama altbilgisi ve son olarak IP üstbilgisiyle sarılır. IP üstbilgisinde VPN istemcisine ve VPN sunucusuna karşılık gelen kaynak ve hedef IP adresi bulunur.



Şekil 3.10 L2TP Trafiğinin IPsec ESP ile Şifrelenmesi

Şifreleme : L2TP iletisi, Internet Anahtar Değişimi (IKE) anlaşma işlemiyle oluşturulan şifreleme anahtarlarını kullanarak, Veri Şifreleme Standardı (DES) veya Üçlü DES (3DES) ile şifrelenir.

3.6.ENTROPİ:

Kriptolanmış bit dizilerinin güvenliği yani şifrelerin gücü entropi ile ölçülür. Yüksek bit miktarına sahip, fazla entropili şifrelerin kırılması daha zordur.

Entropi, bilginin ya da belirsizliğin matematiksel bir ölçüsü olarak düşünülebilir ve olasılık dağılımının bir fonksiyonu olarak hesaplanabilir. Mesela bir bozuk parayı fırlattığınızda kesinlikle yazı ya da tura geleceğini bilemezsiniz. Yani bu işlem maksimum entropiye sahiptir. Bir sistemin entropisi sadece mümkün olan durum sayısı ile değil, o durumda bulunma olasılıkları ile bağlantılıdır.

Bir $X \equiv \begin{pmatrix} x_1, x_2, ..., x_n \\ p_1, p_2, ..., p_n \end{pmatrix}$ sistemini göz önüne alalım. Sistemin içinde bulunduğu durumun entropisi H(X) olsun. Sistem hakkında bazı "bilgiler" sağlandıkça, örneğin "sistem x_1, x_2, x_3 durumlarından birindedir" gibi bir olayın gerçekleştiği bilindiğinde, sistemin entropisi düşecektir. Sistem hakkında önceki entropi H(X) ve "bilgi" birikimi sonrası entropi H1(X)<H(X) olsun. H(X)- H1(X) farkını düşünelim. Eğer edinilen "bilgiler" sistemi kesin belirliyorsa H1(x)= 0 olacaktır. Bir X sisteminin kesin olarak belirlenmesine yarayan bilgi değeri I(X) sistemin söz konusu bilgiyi sağlamadan önceki entropisi H(X) olarak tanımlanabilir.

$* I(x_i) = -\ln p_i = \ln \frac{1}{p_i}$, $i = 1, 2, ..., n$

değerine x_i durumunun bilgi içeriği (information content) veya x_i durumunun kendi-bilgisi yada x_i durumuna ait münferit-bilgi (self-information) denir.

Sistemin bilgi değeri,

$I(X) = \sum_{i=1}^{n} p_i I(x_i)$ sistem durumlarının bilgi içeriklerinin durum olasılıkları ile ağırlıklı ortalamasıdır.

* X rasgele değişkeni (sistem) düzgün dağılıma sahip olduğunda, her duruma ait münferit-bilgi eşit olur ve alttaki gibi bir durum ortaya çıkar.

$I(x_i) = -\ln p_i = -\ln \frac{1}{n} = \ln n$, $i = 1, 2, ..., n$

Sistemin bilgi değeri de $I(X) = \sum_{i=1}^{n} p_i I(x_i) = \ln n$ olur.

*Kesikli bir X rasgele değişkeni (sistemi) için X rasgele değişkeninin aldığı değer x olduğunda, bu sonucun (durumun) bilgi içeriği;

$$I(x) = -\ln P(X = x) = \ln \frac{1}{P(X = x)}$$

Bir sonucun (durumun) bilgi içeriği sadece sonucun gerçekleşmesi olasılığına bağlı olup, bu olasılık ne kadar küçükse, bilgi içeriği o kadar büyüktür.

* Kesikli bir X rasgele değişkenin olasılık fonksiyonu f_X olsun. $X = x$ sonucuna (durumuna) ait münferit-bilgi;
$I(x) = -\ln f_X(x)$ olmak üzere

$I(X) = E(-\ln\{ f_X(X)\} = -\sum_x f_X(x) \ln f_X(x)$ değerine rasgele değişkenin bilgi entropisi (information entropy) de denmektedir.

* $X \equiv \begin{pmatrix} x_1, x_2, ..., x_n \\ p_1, p_2, ..., p_n \end{pmatrix}$ ve $Y \equiv \begin{pmatrix} y_1, y_2, ..., y_m \\ r_1, r_2, ..., r_m \end{pmatrix}$ gibi iki sistemin bileşkesi olan sistemin durumlarının kümesi,

$\{(x_i, y_j) : i = 1,2,...,n \ , \ j = 1,2,...,m\}$ ve

$f_{X,Y}(x_i, y_j) = P(X = x_i, Y = y_j) = p_{ij} \ , \ i = 1,2,...,n \ , \ j = 1,2,...,m$

$I(X,Y) = H(X,Y) = -\sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} \ln p_{ij}$ değerine (X,Y) bileşik sistemin bilgi değeri denir.

Sürekli Sistemlerde Enformasyon Kavramı

Sürekli sistemlerde (rasgele değişkenlerde) bilgi ile ilgili kavramlar, kesikli rasgele değişkenlerdeki formüllerde toplam işareti Σ yerine integral işareti ve olasılık fonksiyonu değerleri yerine olasılık yoğunluk fonksiyonları yazılması ile oluşturulmaktadır.

Sürekli bir X sisteminin (sürekli bir rasgele X değişkeninin) bilgi değeri,

$$I(X) = -\int_{-\infty}^{\infty} f(x) \ln f(x) dx$$

olarak tanımlanmaktadır.

$$H(X) = -\int_{-\infty}^{\infty} f_X(x) \ln f_X(x) dx - \ln \Delta x = -E[\ln f_X(X) \Delta x]$$

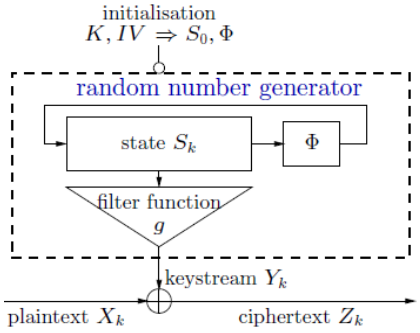
Sürekli dağılımlarda, rasgele değişkenin belli bir değer eşit olması sıfırdır. Sürekli dağılımlar (sistemler) için kısmi bilgi değerinden bahsedilemez.

ŞifreEntropisi

Örneğin ATM'lerde kullandığımız PIN'leri ele alalım; yani sadece rakam kullanabildiğimiz şifreleri. Her karakter 0-9 arasında bir rakam olmak zorunda. Bilgisayar ortamında şifrelerin gücü, boyutuyla yani bit miktarıyla ölçülür. ASCII karakter setinde, her karakter teoride 8 bit alan kaplar fakat çoğu bit sıkıştırılarak "sayının esas niteliği" bozulmadan göz ardı edilebilir. Mesela, 0 için 0000 veya 1 için 0001 gösterimi kullanılabilir. Bu durumda karakter başına gerçek boyut 3.3 bit olur. Ve her rakamı farklı rastgele 4 rakamlı bir şifre ortalama 13 bit boyutundadır.

Yani 4 rakamlı bir PIN'i tahmin etmek bir bozuk parayı üst üste 13 kere yazı ya da tura atmakla eşdeğer şansta. Ufak bir hesaplamayla 2^13=8.192 farklı yolu olduğunu görüyoruz. Bunca farklı kombinasyon arasından doğru şifreyi bulmak zor görünmektedir.

Şifrelenmiş Dizi Modeli:



İlk Durum : So

Güncelleme Fonksiyonu : $S_{k+1} = \Phi(S_k) \text{ for } k \geq 0$

Anahtar Dizisi : $Y_k = g(S_k)$

Şifrelenmiş Metin : $Z_k = X_k \oplus Y_k$

Olasılıklı Model :

Yukardaki şifreleme diyagramında entropi hesaplamak için olasılıklı modeli baz alırız ve bunun için önce aşağıdaki bazı parametreleri belirtmek gerekir.

Durum Uzayı : $\Omega_n = \{\omega_1, \omega_2, \ldots \omega_n\}$

İlk Dağıtım : $\{p_i\}_{i=1}^n \text{ with } p_i = Pr[S_0 = \omega_i]$

Rasgele Güncelleme Fonksiyonu : $Pr[\Phi = \varphi] = 1/n^n$
$\text{for all } \varphi \in \mathcal{F}_n = \{\varphi : \Omega_n \to \Omega_n\}$

*Öncelikle olasılık Φ nin k iterasyonundan sonra wi değerine sahipken bulunduğu durumudur.

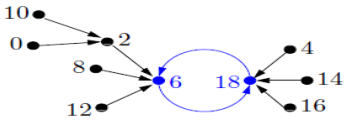
$p_i^{\Phi}(k) = Pr[S_k = \omega_i] = Pr[\Phi^k(S_0) = \omega_i]$

*Daha sonra Shannon un entropisi yani rasgele değişkendeki bilgi ölçümü $H \leq \log_2(n)$

* Son olarak entropi durumunda ise ;

$$H_k^{\Phi} = \sum_{i=1}^{n} p_i^{\Phi}(k) \log_2 \left(\frac{1}{p_i^{\Phi}(k)} \right)$$

Entropi Kestirimi :



16 farklı sayı içerisinden k =2 bit için kestirim sözkonusudur.

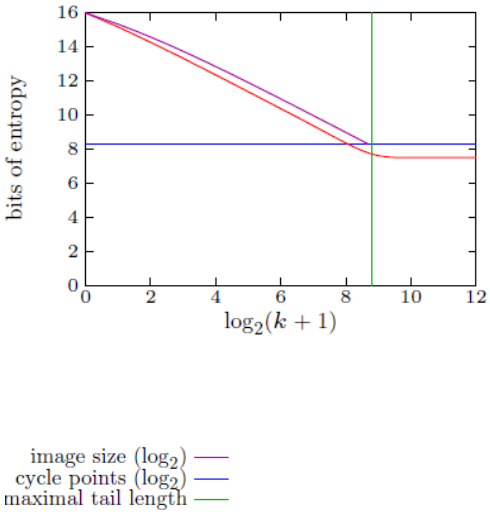
asomptotik değerler (entropi değeri) $n \to \infty$

beklenen daire nokta sayısı (cycle points) : $\mathrm{cp}(n) \sim \sqrt{\pi n/2}$

beklenen en büyük kuyruk uzunluğu (maximal tail length) : $\mathrm{mt}(n) \sim \sqrt{\pi n/8}$

beklenen r nodeları kaynak sayısı : $\mathrm{rn}(n,r) \sim \frac{n}{r!e}$

$n = 2^{16}$ için ;



Şekil 3.11.Bit Entropi İlişkisi

[14]

4.BULGULAR

4.1.MODEL SIM KULLANARAK RSA ALGORİTMASINI GERÇEKLEME VE REAL TIME SİMÜLASYON

Model sim altında Rsa algoritmasını gerçeklemek için kullanılan kaynak kodları:

Parametreler:
Clk: devre için clock girişi sağlar.

Reset:clock ile counter döngüsü kontrolü için gerekli parametredir.

Ku1,Ku2,message : MESSAGE girişte belirtilen şifrelenecek data dizisidir. KU1,KU2 simulasyon için kullanılan RsaWorkbench için girilmesi gereken giriş değişkenleridir.

data_rdy: stg1,stg2 çalışırken tek bitlik giriştir.
Herhangi bir değişkeni reg ya da wire ile belirtmezsek 1 bit genişliğinde kabul edecektir.

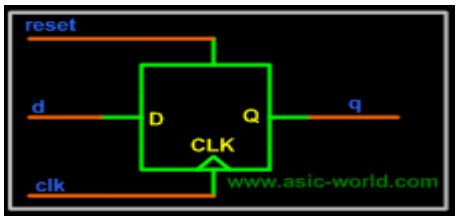
wire: Birleştirme noktası olarak işlev görür. Birleşimsel devrelerde kullanılır fakat bu tip devreler değerleri depolayamaz.

```
1 module wire_example( a, b, y);
2   input a, b;
3   output y;
4
5   wire a, b, y;
6
7   assign y = a & b;
8
9 endmodule
```

reg:birleşimsel ve seri devrelerde kullanılır. Değerleri hafızada tutabilir, initial ve always bloklarında kullanılır.

```
 1 module reg_combo_example( a, b, y);
 2 input a, b;
 3 output y;
 4
 5 reg   y;
 6 wire a, b;
 7
 8 always @ ( a or b)
 9 begin
10   y = a & b;
11 end
12
13 endmodule
```

```
1 module reg_seq_example( clk, reset, d, q);
2 input clk, reset, d;
3 output q;
4
5 reg   q;
6 wire clk, reset, d;
7
8 always @ (posedge clk or posedge reset)
9 if (reset) begin
10   q <= 1'b0;
11 end else begin
12   q <= d;
13 end
14
15 endmodule
```



Always ve initial komutları: Initial block sadece simulasyonun başında process i çalıştırırken always komutu tüm simülasyon boyunca çalıştırır. Ayrıca gecikme de nitelendirebilir. Always den sonra kullanılan @ işareti parantezden sonraki duruma geldiğinde process tetiklenecektir. Komut anlamı giriş değişkeni değiştiğinde çıkış da değişir. Flip flop devreleri gibi köşe duyarlı devrelerde sürekli devreyi resetlemek durumundayız, posedge komutu 0 dan 1 geçişini yapar.

=: birleşimsel devrede kullanılır
<= : birbiri ardına devam eden blokta kullanılır

Delay için #5 clk = ~clk;
Burada işlemi 5 zaman birimi geciktirir.

```
1 always  @ (posedge clk )
2 if (reset == 0) begin
3   y <= 0;
4 end else if (sel == 0) begin
5   y <= a;
6 end else begin
7   y <= b;
8 end
```

mod_out,mod_num1,mod_num2 mod alma işlemi için tanımlanan değişkenlerdir

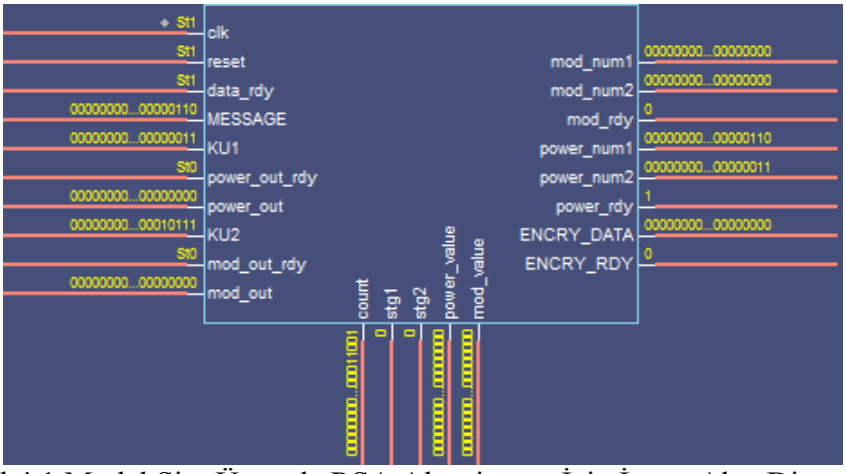
power_num1,power_num2,power_out ve power_rdy güç fonksiyonunu hesaplamak için kullanılan değişkenlerdir.

reg [7:0]count : 8 bit lik 0 dan 255 e kadar devam eden sayaçtır. Bu şekilde durumlar tespit edilir.

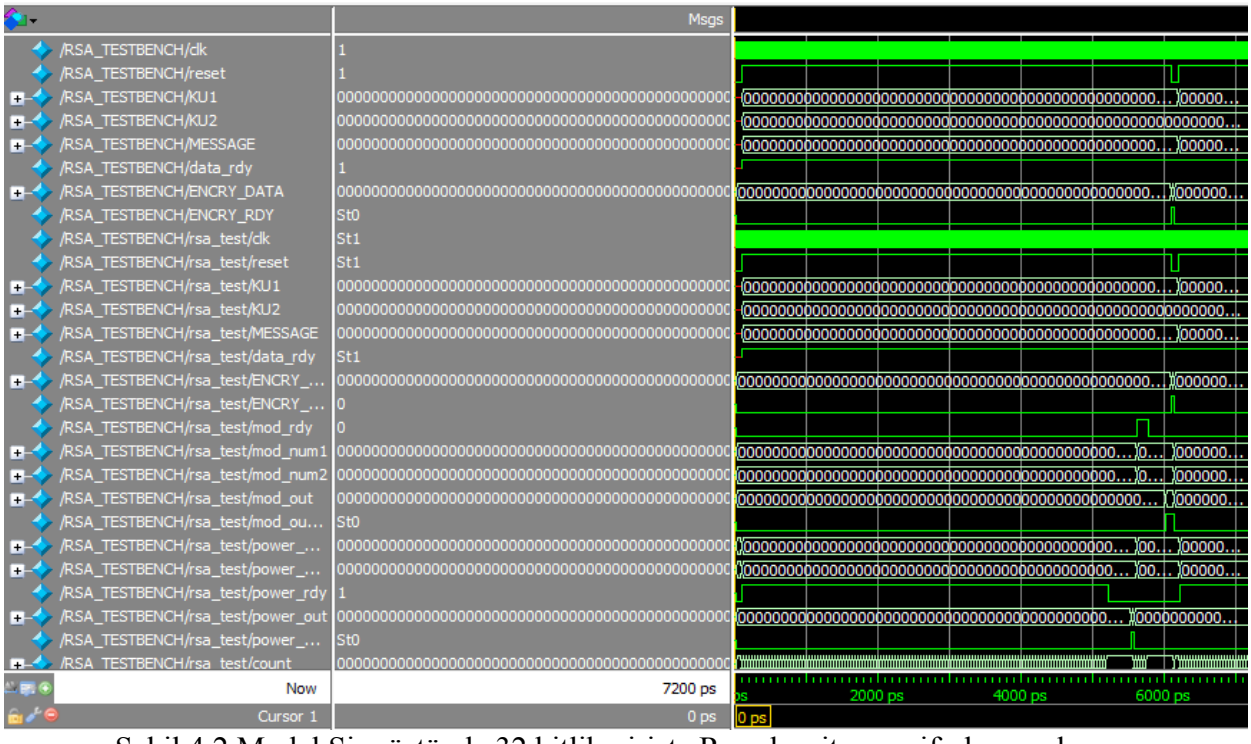
reg [31:0]power_value: Sayacın başlamasıyla güç değeri hesaplanır.

reg [31:0]mod_value: Sayacın başlamasıyla mod hesaplanır.

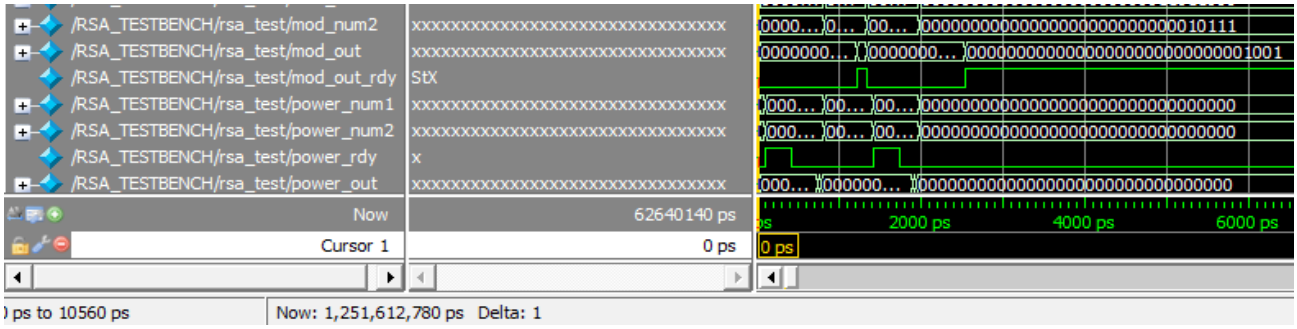
reg stg1,stg2 : MESSAGE,KU1 değerlerini power_num1 ve power_num2 e geçiren process i yöneten değişkenleri depolar.



Şekil 4.1.Model Sim Üstünde RSA Algoritması İçin İşaret Akış Diyagramı



Şekil 4.2.Model Sim üstünde 32 bitlik girişte Rsa algoritması şifreleme çıkışı



Şekil 4.3.Model Sim üstünde 32 bitlik girişte Rsa simülasyonu ve şifreleme çıkışı

Simülasyon delta darbeleriyle senkron bir şekilde stop yapana kadar devam eder. Farklı bit girişteki şifrelemeyi yapabilmesi ve simülasyonu çalıştırması için giriş ve çıkış değerleri (KU1,KU2,message,encrypt_data,encrypt_rdy) , sayaç değerleri (count) , register ve durum inceleme değişkenlerinin (power_value,mod_value,mod_rdy,power_rdy,data_rdy,stg1,) değerlerini değiştirmek gerekir.

Performans Sonuçları

Tablo 4.1 Model Sim Kullanılarak Elde Edilen Rsa Şifreleme ve Kod Çözme Süreleri

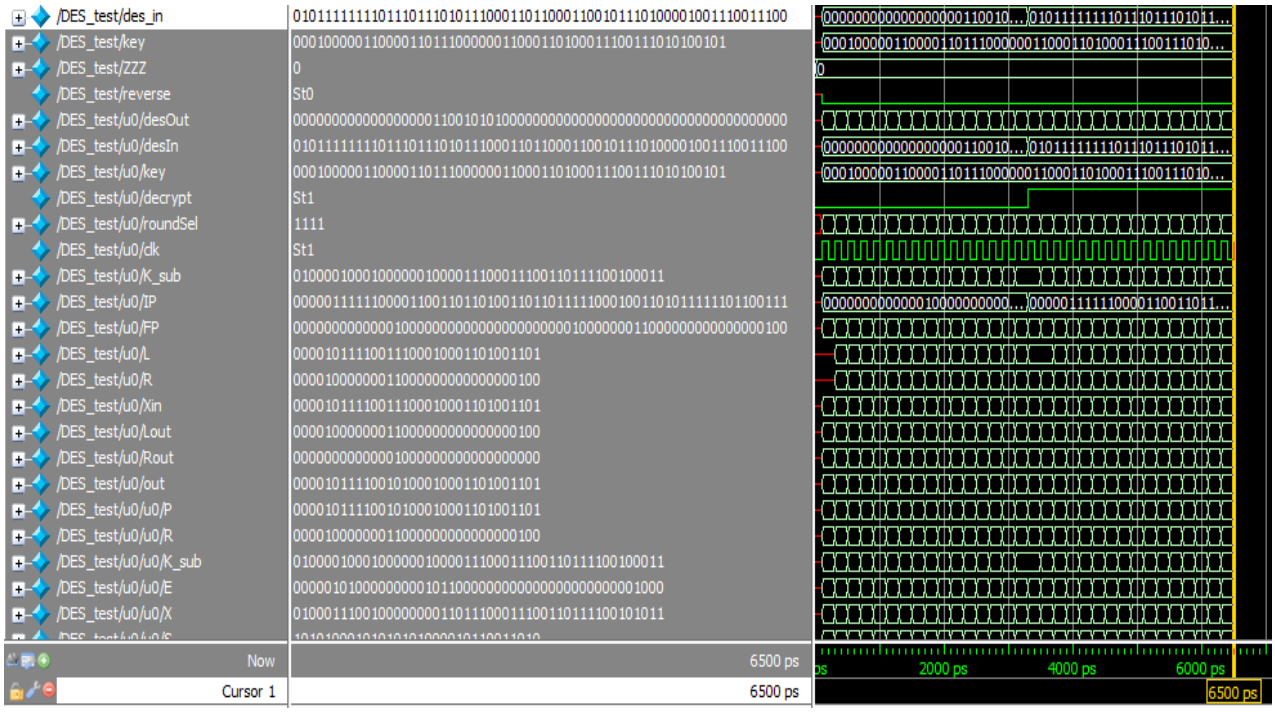
RSA ALGORITHM Text File (bit)	Encryption (µs)	Decryption (µs)
32	7,2	6,2
128	12,32	10,8
256	22,56	20,4
512	37,2	36,2
1K	62,32	60,8
2K	112,56	100,4

Sonuçlar incelendiğinde kod çözme süresinin şifreleme süresine oranı girişteki mesaj uzunluğu ile doğru orantıda arttığı görülmektedir.

4.2.MODEL SIM KULLANARAK DES ALGORİTMASINI GERÇEKLEME VE REAL TIME SİMÜLASYON

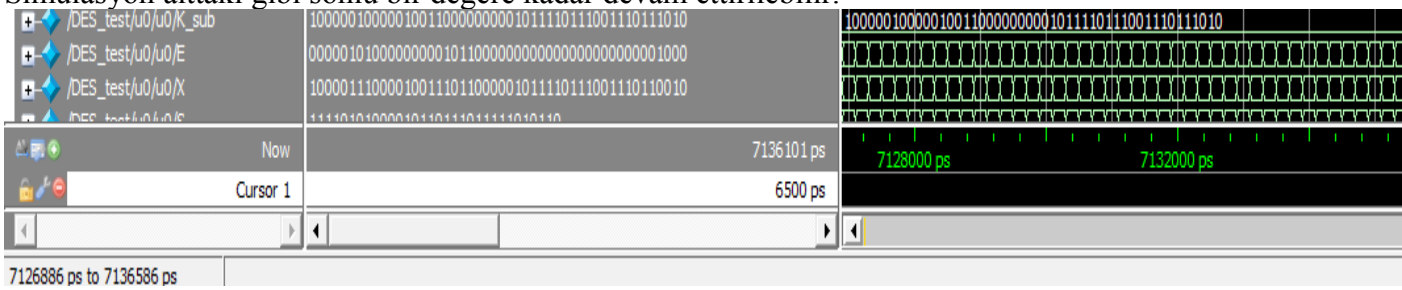
Des algoritmasının Model sim simulasyonunda S kutusu yerleşimi, alt anahtarların elde edildiği key_sel, permütasyon adımlarının işlev gördüğü crp ve ana algoritmanın olduğu des olmak üzere 4 farklı alt programlama adımı vardır. Ana algoritmanın olduğu des dosyasında diğer 3 process çağrılmaktadır ve çıktı elde edilmektedir.

Simülasyon Sonuçları:



Şekil 4.4.Model Sim üstünde 64 bitlik girişte Des algoritması şifreleme çıkışı

64 bitlik giriş için elde edilen şifreleme süresi 6500 ps dir.
Simulasyon alttaki gibi sonlu bir değere kadar devam ettirilebilir.



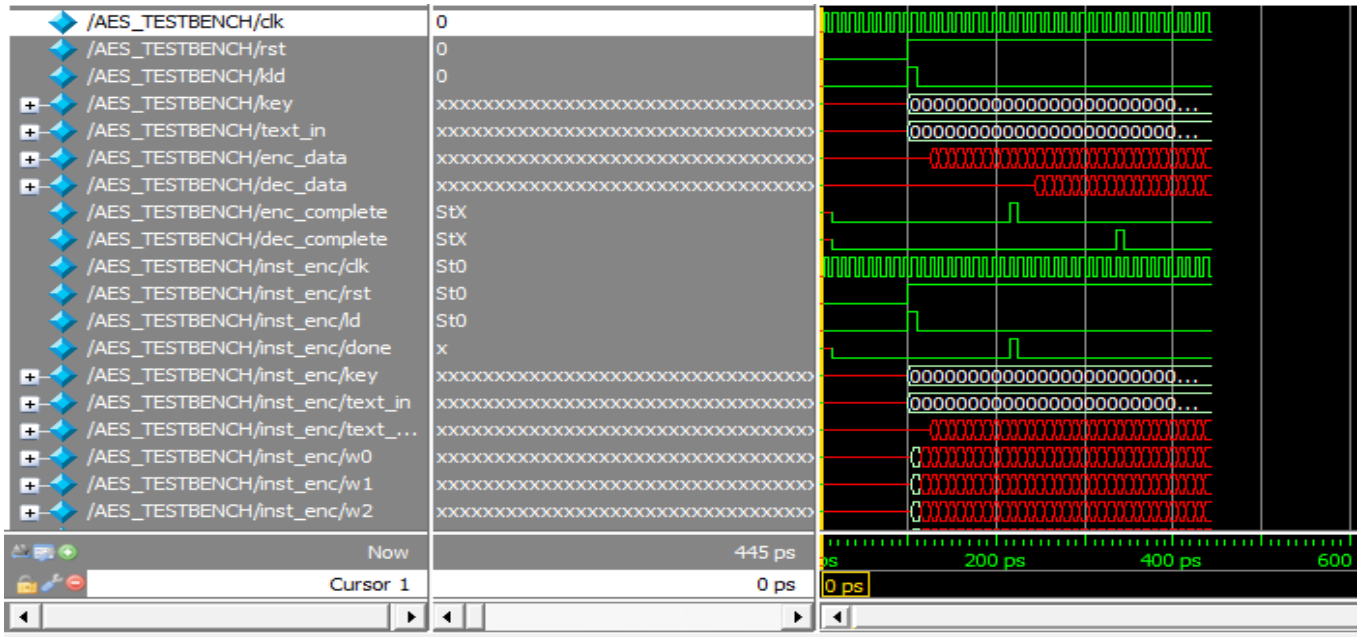
Şekil 4.5.Model Sim üstünde 64 bitlik girişte Des algoritması simülasyonu çıkışı

Farklı bit giriş değerleri için model sim altında Des alt kütüphanede DesIn ve Desout parametrelerini , Des Test alt kütüphanede wire DesIn ve wire Desout parametrelerini değiştirmek gerekmektedir. Des algoritması şifrelemeyi 64 bit lik bloklar halinde yaptığı için simülasyon aynı işlemleri giriş değerine bakarak döngüyü tekrarlayarak gerçekler.

Tablo 4.2 Model Sim Kullanılarak Elde Edilen Des Şifreleme ve Kod Çözme Süreleri

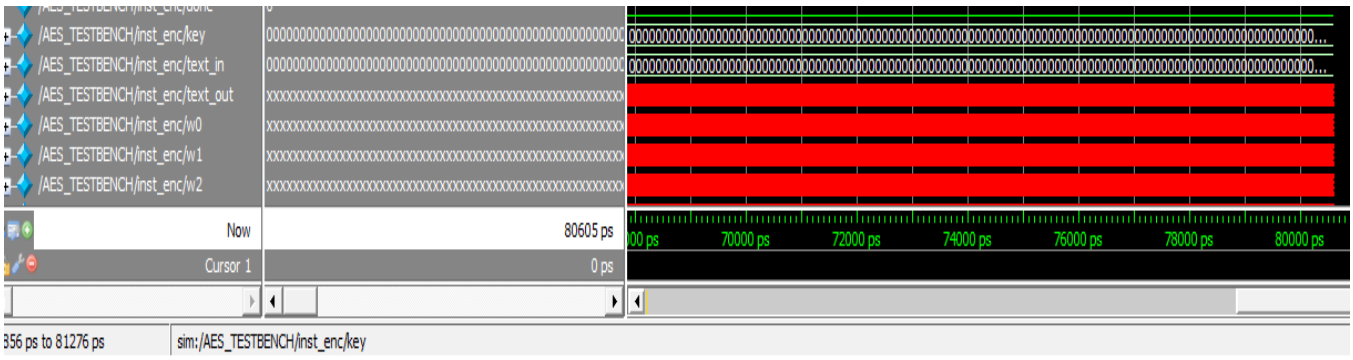
DES ALGORITHM Text File (bit)	Encryption (µs)	Decryption (µs)
32	5,2	4,2
64	6,5	5,4
128	7,12	6,8
256	12,36	10,4
512	17,2	16,2
1K	32,32	30,8
2K	62,56	50,4

4.3.MODEL SIM KULLANARAK AES ALGORİTMASINI GERÇEKLEME, REAL TIME SİMÜLASYON VE SONUÇLARI



Şekil 4.6.Model Sim üstünde 64 bitlik girişte Aes algoritması şifreleme çıkışı

32 bitlik girişte Aes algoritması şifreleme süresi 445 ps dir.
Simulasyon alttaki gibi sonlu bir değere kadar devam ettirilebilir.



Şekil 4.7.Model Sim üstünde 64 bitlik girişte Aes algoritması simülasyonu çıkışı

Tablo 4.3 Model Sim Kullanılarak Elde Edilen Aes Şifreleme ve Kod Çözme Süreleri

AES ALGORITHM Text File (bit)	Encryption (µs)	Decryption (µs)
32	0,2	0,12
128	0,445	0,4
256	1,56	1,4
512	2,72	2,2
1K	5,32	4,8
2K	10,56	10,4

5.TARTIŞMA VE SONUÇ

5.1. SES,VİDEO VE REAL TIME DATA ŞİFRELEME:

Ses Şifreleme :

Ses şifrelemede H.323 ve XSIP (extended SIP) sinyalleşmeleri 256 bitlik AES şifrelemesine olanak sağlar. AES 256 kripto algoritması 128 bitlik data bloklarında 256 bitlik anahtarları kullanarak ses güvenliğini sağlar.

Her bir oturumda anahtarların değişimi yine güvenli bir metod olan Diffie-Hellman prosedürleri kullanılarak yapılır. Böylece Man-in-the-Middle (MIM) atakları önlenir ve güvenlik derecesi daha da yükselir. Ses kodlayıcılardan alınan ses paketleri kriptolanarak RTP paketleri içine yerleştirilir. [16]

Ses kriptolama algoritmaları H.323 uçnoktadan bir başka uçnoktaya olabileceği gibi, daha fazla güvenlik için H.323 uçnoktadan santrala entegre gatekeeper`a da olabilir.

Alıcı ve gönderen ortak bir kullanıcı adı ve şifreyi paylaşırlar. Bu paylaşım esnasında güvenli gatekeeper metodları uygulanır. Kayıt için, H.225 RAS mesajlarının gönderilip alınması H.235 Baseline Güvenlik Profili ile gerçekleşir. Bu profilde, HMAC-SHA1-96 hashing algoritmaları uygulanmaktadır.

Ses şifreleme algoritmaları GSM, VoIP gibi real time uygulamalarda uçtan uca konuşmayı garanti altına almak için kullanılır.

Çok fazla tercih edilmese de DES,FEAL ve IDEA gibi algoritmalar da kullanılabilir. [15]

*Gsm platformunda public key şifreleme olduğundan RSA tercih edilmez.
*Gsm ortamında 40 bit keye sahip A5 şifreleme algoritmaları tercih edilir. Ayrıca yaşam süresi fazla olduğundan 56 bitlik DES algoritması da kullanılabilir.
*3G (UMTS), 2G (EDGE) and Wi-Fi ortamlarında key uzunluğunun gücü AES algoritmasını kullanma sebebidir.

Video Şifreleme :

Multimedya güvenliği standart simetrik algoritmalar ile sağlanır. Fakat sınırlamaların çeşitliliği bunu kolaylıkla uygulamayı engeller. Video şifrelemede alttaki gibi bazı zorluklar vardır.

*Tipik Mpeg sıkıştırılmış video dosyası boyut olarak çok büyüktür.
*Decoding örneğin kod çözme işlemi real time süresinde gerçekleştirilmelidir.
*Playback ve fast forward işlemleri mümkün olabildiğince hızlı olmalıdır.

Video şifrelemede DES ve IDEA algoritmalarına göre hızlı ve daha güvenli olmasından dolayı AES algoritması kullanılır. [18]

Real Time Data Transfer Durumunda Şifreleme :

Real time data transferini yapan Secure Real-time Transport Protocol (SRTP) şifreleme algoritması olarak AES i tercih eder.

*RTP datasını şifreleme standartı artan tamsayı sayacıdır. Bu modda şifreleme için en uygun algoritma AES dir.
*Şifreleme anahtarı ve salt anahtarın aynı olması 3G UMTS platformunda AES in seçilme sebebidir.

5.2.ELDE EDİLEN DEĞERLERİN KARŞILAŞTIRILMASI VE YORUMLAR

Tablo 5.1 DES AES RSA Şifreleme Süreleri Karşılaştırma Tablosu

Giriş Bit Değeri (bit)	Şifreleme Süreleri (µs)		
	RSA	DES	AES
32	7,2	5,2	0,2
128	12,32	7,12	0,445
256	22,56	12,36	1,56
512	37,2	17,2	2,72
1024	62,32	32,32	5,32
2048	112,56	62,56	10,56

Tablo 5.2 DES AES RSA Şifre Çözme Süreleri Karşılaştırma Tablosu

Giriş Bit Değeri (bit)	Şifre Çözme Süreleri (µs)		
	RSA	DES	AES
32	6,2	4,2	0,12
128	10,8	5,4	0,4
256	20,4	6,8	1,4
512	36,2	10,4	2,2
1024	60,8	16,2	4,8
2048	100,4	30,8	10,4

Tablo 5.3 Genel Karşılaştırma Tablosu

Analiz Parametresi	RSA	DES	AES
Şifreleme	Yavaş	Hızlı	Çok hızlı
Şifre Çözme	Yavaş	Hızlı	Çok hızlı
Anahtar Elde Etme	Kolay	Zor	Zor
Güvenlik	Üstün	Düşük	Orta

Analiz için ModelSim Altera Web Edition programı kullanılarak RSA, AES ve DES için kaynak kodları yazılmıştır. Akabinde algoritmaların şifreleme ve şifre çözmedeki yeteneklerinin hız, memory ve anahtar gibi unsurlarda hesaba katılarak detaylı bir şekilde incelemesi yapılmıştır.

Analiz sonuçlarını yorumlarsak ;

Genel olarak kullanılan anahtar uzunlukları DES için 56 bit, AES için 128,192 ve 256 bit, RSA için 1024,2048 ve 4096 bit tir. Anahtar uzunluğunu arttırmak AES algoritmasına daha fazla komplekslik katarken şifreleme süresini çok az miktarda arttırmaktadır.

AES in kod çözmedeki ve şifrelemedeki performansı çok üstündür. Ayrıca yazılımsal ve donanımsal anlamda uygulanması kolaydır. Buna karşın güvenliğin üst düzeyde olduğu uygulamalarda RSA bir adım öndedir. AES te anahtar üretimi de zordur.

RSA ayrıca en fazla memory gereksinimi duyan algoritmadır.

SBoxlar DES"in en zayıf olduğu noktalardır. Zaten DES bu SBoxlardaki açıklar yüzünden kırılabilmektedir.

RSA da sonuçlar incelendiğinde kod çözme süresinin şifreleme süresine oranı
girişteki mesaj uzunluğu ile doğru orantıda arttığı görülmektedir. Ayrıca anahtar uzunluğunu arttırdığımızda kod çözme süresi daha fazla oranda artmaktadır. Buna karşın yüksek güvenlik gerektiren uygulamalarda anahtar uzunluğunu arttırmak gerekmektedir.

KAYNAKLAR:

[1] CHALLA NARASIMHAM(1), JAYARAM PRADHAN(2), 2008 JATIT, *Evaluation of performance characterictics of cryptosystem using text files,*
1 Assoc. Prof., Dept of Computers, MVGR College of Engineering, Vizianagaram, India-535 005 2 Professor, Dept of Computer Science, Berhampur University, Orissa, India.

[2] DAEMEN J. & RIJMEN V., 2000 , *A Spesification for Rijndael ,The Aes Algorithm*

[3] DEMİR NECATİ, DALKILIÇ GÖKHAN, 2007 *Akademik Bilişim'07 - IX. Akademik Bilişim Konferansı Bildirileri Dumlupınar Üniversitesi, Kütahya*
Dokuz Eylül Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35160, İzmir
ndemir@demir.web.tr, dalkilic@cs.deu.edu.tr

[4] DIAA SALAMA ABD ELMINAAM, HATEM MOHAMED ABDUAL KADER , MOHIY MOHAMED HADHOUD, 2010, *Evaluating The Performance of Symmetric Encryption Algorithm ,* International Journal of Network Security, Vol.10, No.3, PP.216–222

[5] GATLIFF BILL, 2003, *Courtesy of Embedded Systems Programming* http://www.design-reuse.com/articles/5922/encrypting-data-with-the-blowfish-algorithm.html

[6] SHASHI MEHROTRA SETH, RAJAN MISHRA, 2011, *Comparative Analysis Of Encryption Algorithms For Data Communication*
Dept. of CS & IT, MERI College of Engg. & Tech., ASANDA (near Sampla), Bahadurgarh, Haryana, India IJCST Vol. 2, Issue 2

[7] SÖNMEZ REMZİYE, 2002, *"Veri Sifreleme Standardı (DES) ve Rivest Shamir Adleman (RSA) Güvenlik Algoritmalarının VLSI Tasarımı",* Yüksek Lisans Tezi, Hacettepe Üniversitesi.

[8] ŞEN EVREN , http://www.scribd.com/doc/2581535/Kriptografi-ve-Kull-Alan-I , [Ziyaret Tarihi: 4 OCAK 2012]

[9] ŞEN ŞENOL, 2006 " *İndirgenmiş SPN (Substitution Permutation Network) Algoritması İçin Lineer Kriptanaliz Uygulaması* " Yüksek Lisans Tezi Trakya Üniversitesi

[10] MITTAL MOHIT, 2012, *Performance Evaluation of Cryptographic Algorithm* , Computer Science & Engineering Guru Nanak Dev University, Amritsa , Volume 41– No.7

[11] WAYNE KEVIN , SEDGEWICK ROBERT , *Algorithms in Java, Chapter 14 Hashing* Princeton University

[12] HALE ROBIN, 2012 , *A History of Encryption* , Ezinearticles

[13] SETTI NEETU, 2010 , *Cryptanalysis of Modern Cryptographic Algorithms ,*
Department of Electronics and Telecommunication , Guru Tegh Bahadur Institue Of Technology, GGSIPU, New Delhi, India

[14] ROCK ANDREA, 2008 , *Stream Ciphers Using a Random Update Function: Study of the Entropy of the Inner State* , Inria Paris-Rocquencourt, Team Secret France

[15] BRANDAU MARKUS ALBERT , 2008 , *Implementation of a real-time voice encryption system,* Master Thesis Universitat Politècnica de Catalunya EUETIT

[16] Cellular Voice Encryption , http://www.spyworld.com/equipment/listings/l0004.html [Ziyaret Tarihi: 1 MAYIS 2012]

[17] SEIDEL TANYA , SOCEK DANIEL , SRAMKA MICHAL , *Cryptanalysis of Video Encryption Algorithms ,* Department of Mathematical Sciences Florida Atlantic University, 777 Glades Road Boca Raton, FL 33431, U.S.A. teseidel@aol.com

[18] NEHETE JAYSHRI , BHAGYALAKSHMI K , CHAUDRAI SHASKIANT , *A Real-time MPEG Video Encryption Algorithm using AES ,* Central Research Laboratory Bharat Electronics Ltd., Bangalore-560013,

EK

1-RSA Algoritmasını Model Sim de gerçeklemek için kaynak kodları:

```
module RSA_ALGORITHM(clk,
                     reset,
                     KU1,
                     KU2,
                     MESSAGE,
                     data_rdy,
                     ENCRY_DATA,
                     ENCRY_RDY
                     );

input clk;
input reset;

input [31:0]KU1;
input [31:0]KU2;
input [31:0]MESSAGE;
input data_rdy;

output [31:0]ENCRY_DATA;
output ENCRY_RDY;

reg [31:0]ENCRY_DATA;
reg ENCRY_RDY;


reg mod_rdy;
reg [31:0]mod_num1;
reg [31:0]mod_num2;
wire [31:0]mod_out;
wire mod_out_rdy;

reg [31:0]power_num1;
reg [31:0]power_num2;
reg power_rdy;

wire [31:0]power_out;
wire power_out_rdy;

reg [7:0]count;
reg [31:0]power_value;
reg [31:0]mod_value;

reg stg1,stg2;

always @(posedge clk )
```

```
begin

if(~reset)   “giriş biti ile if döngüsü başlar”
begin
mod_num1<=32'd0;
mod_num2<=32'd0;
mod_rdy<=1'b0;
power_num1<=32'd0;
power_num2<=32'd0;
power_rdy<=1'b0;
count<=8'd0;
stg1<=1'b0;
stg2<=1'b0;
power_value<=32'd0;
mod_value<=32'd0;
ENCRY_DATA<=32'd0;   “mod değeri atanan 32 bitlik datadır.”
ENCRY_RDY<=1'b0;        “stg1,stg2 yükseldikçe sinyali de yükselten tek bitlik datadır.
Bu şartlar sağlanırsa döngü devam eder.”
end
else if (data_rdy && ~stg1 && ~stg2 )  “Bu şartlar da sağlanmalı.”
begin

mod_num1<=32'd0;
mod_num2<=32'd0;
mod_rdy<=1'b0;
power_num1<=MESSAGE;
power_num2<=KU1;
if (count >8'd7)    “a”
begin
power_rdy<=1'b0;  “b“
end

else
begin
power_rdy<=1'b1;  “c”
count<=count+8'd1;  “d”
end

“Burada “a b c d “ ile  işaretlenen satırlarda counter 7 den büyükse power_rdy(referans
değişkeni ) =0 olur. Power_rdy =1 ise counter 1 arttırarak saymaya başlar.”
if (power_out_rdy)
begin
stg1<=1'b1;
stg2<=1'b0;
power_value<=power_out;
count<=8'd0;
end
  “Burada stg1,stg2 message ve ku1 değerlerinden power değerlerine geçişi sağlar.”
else
```

```
begin
stg1<=1'b0;
stg2<=1'b0;
power_value<=power_value;
end

end

else if (data_rdy && stg1 && ~stg2 )
begin

mod_num1<=power_value;
mod_num2<=KU2;

power_num1<=32'd0;
power_num2<=32'd0;

if (count >8'd3)  “e”
begin
mod_rdy<=1'b0;  “f”
end

else
begin
mod_rdy<=1'b1;  “g”
count<=count+8'd1;  “h”
end
```

“Burada “e f g h “ ile işaretlenen satırlarda counter 3 den büyükse mod_rdy=0 olur . Mod_rdy=1 olursa counter 1 arttırarak saymaya başlar.”

```
if (mod_out_rdy)
begin
stg1<=1'b1;
stg2<=1'b1;
mod_value<=mod_out;
end

else
begin
stg1<=1'b1;
stg2<=1'b0;
mod_value<=mod_value;
end
```

“stg2=0 ve stg1=1 olduğunda mod_value değeri olarak atanır. Fakat stg2 ve stg1 her ikisi de 1 olduğunda mod value değeri mod_out olarak çıkar ve döngü biter.”

```
end

else if (stg1 && stg2)
begin
```

```
ENCRY_DATA<=mod_value;
ENCRY_RDY<=1'b1;
end
  “Şifrelenmiş data mod değeri ile eşleştirilir.“

end

mod_operationrsa  inst_mod(.clk(clk),
                  .rst(reset),
                  .op_rdy(mod_rdy),
                  .sign(1'b0),
                  .in1(mod_num1),
                  .in2(mod_num2),
                  .out(mod_out),
                  .out_rdy(mod_out_rdy)
                  );
power_process inst_power(.clk(clk),
                   .reset(reset),
                   .data1(power_num1),
                   .data2(power_num2),
                   .in_rdy(power_rdy),
                   .out(power_out),
                   .out_rdy(power_out_rdy)
                   );
Endmodule
```

2- DES Algoritmasını Gerçeklemek İçin Kaynak Kodları:

2.1.Alt Anahtarların Elde Edilmesi:

```
module  key_sel(K_sub, K, roundSel, decrypt);
output [1:48] K_sub;  “Yer değiştirme permütasyonudur”
input [55:0] K; “İlk baştaki 56 bitlik anahtardır”
input [3:0] roundSel; “ Multiplexer da istenen girişi çıkışa bağlayan girişle beraber dahil olan
Select kısmıdır”
input   decrypt;

reg  [1:48] K_sub;
wire [1:48] K1, K2, K3, K4, K5, K6, K7, K8; “Alt anahtarlar”

wire [2:0] roundSelH;
wire   decryptH;

assign roundSelH[2:0] = roundSel[3] ? (~roundSel[2:0]) : roundSel[2:0];
assign decryptH  = decrypt ^ roundSel[3];

always @(K1 or K2 or K3 or K4 or K5 or K6 or K7 or K8 or roundSelH)
“Tüm simülasyon boyunca çalıştırır always komutu ile “
    case (roundSelH)     “Tek değişken var fakat birçok değer mevcut.Bu durumlarda case
kontrol parametrelesi kullanılır.”
```

```
        0:  K_sub = K1;
        1:  K_sub = K2;
        2:  K_sub = K3;
        3:  K_sub = K4;
        4:  K_sub = K5;
        5:  K_sub = K6;
        6:  K_sub = K7;
        7:  K_sub = K8;
   endcase

assign K8[1]  = decryptH ? K[6]  : K[24];
assign K8[2]  = decryptH ? K[27] : K[20];
assign K8[3]  = decryptH ? K[10] : K[3] ;
.........
assign K1[46] = decryptH ? K[49] : K[1] ;
assign K1[47] = decryptH ? K[0]  : K[7] ;
assign K1[48] = decryptH ? K[21] : K[28];
endmodule
“Bütün K8 den K1 e kadar devam eder.
```

2.2.S Kutusu Yerleşimi:

```
module  sbox1(addr, dout);
input   [1:6] addr;  “ S kutusundaki elemanlar için giriş 6 bittir “
output  [1:4] dout;  “ S kutusundaki elemanlar için çıkış 4 bittir “
reg     [1:4] dout;

always @(addr) begin
   case ({addr[1], addr[6], addr[2:5]})        “ 1 ve 6.bitler satır no yu verir. Aradaki 2 den 5 e
kadarki bitler toplamı sütun no dur”

        0:  dout =  14;
        1:  dout =   4;
        2:  dout =  13;
        3:  dout =   1;
        4:  dout =   2;
        5:  dout =  15;
        6:  dout =  11;
        7:  dout =   8;
        8:  dout =   3;
        9:  dout =  10;
       10:  dout =   6;
       11:  dout =  12;
       12:  dout =   5;
       13:  dout =   9;
       14:  dout =   0;
       15:  dout =   7;
```

```
16:  dout =  0;
17:  dout = 15;
18:  dout =  7;
19:  dout =  4;
20:  dout = 14;
21:  dout =  2;
22:  dout = 13;
23:  dout =  1;
24:  dout = 10;
25:  dout =  6;
26:  dout = 12;
27:  dout = 11;
28:  dout =  9;
29:  dout =  5;
30:  dout =  3;
31:  dout =  8;

32:  dout =  4;
33:  dout =  1;
34:  dout = 14;
35:  dout =  8;
36:  dout = 13;
37:  dout =  6;
38:  dout =  2;
39:  dout = 11;
40:  dout = 15;
41:  dout = 12;
42:  dout =  9;
43:  dout =  7;
44:  dout =  3;
45:  dout = 10;
46:  dout =  5;
47:  dout =  0;

48:  dout = 15;
49:  dout = 12;
50:  dout =  8;
51:  dout =  2;
52:  dout =  4;
53:  dout =  9;
54:  dout =  1;
55:  dout =  7;
56:  dout =  5;
57:  dout = 11;
58:  dout =  3;
59:  dout = 14;
60:  dout = 10;
61:  dout =  0;
62:  dout =  6;
63:  dout = 13;
```

```
   endcase
   end

endmodule
```

2.3.Permütasyon Adımları Crp:

```
module  crp(P, R, K_sub);
output  [1:32]  P;          "P kutusu permütasyon kısmıdır,en sondaki 32 bitlik çıkıştır"
input   [1:32]  R;          " Ro 32 bitlik algoritmanın sağ tarafındaki giriştir."
input   [1:48]  K_sub;   " k_sub yer değiştirme perm dur "

wire    [1:48] E;              "E genişletilmiş perm dur"
wire    [1:48] X;              " S kutusu işleminden önceki 48 bitlik kısımdır"
wire    [1:32] S;              "S kutusu adımıdır"

assign E[1:48] = {      R[32], R[1], R[2], R[3], R[4], R[5], R[4], R[5],
                        R[6], R[7], R[8], R[9], R[8], R[9], R[10], R[11],
                        R[12], R[13], R[12], R[13], R[14], R[15], R[16],
                        R[17], R[16], R[17], R[18], R[19], R[20], R[21],
                        R[20], R[21], R[22], R[23], R[24], R[25], R[24],
                        R[25], R[26], R[27], R[28], R[29], R[28], R[29],
                        R[30], R[31], R[32], R[1]};

assign X = E ^ K_sub;  " k_sub yer değiştirme perm dur. E genişletilmiş perm dur"

"addr X girişinde 6 bit olup 1 ve 6.bitler 01 örneğin 1.satır satır no yu ve S kısmında çıkışı 4
bittir"
sbox1 u0( .addr(X[01:06]), .dout(S[01:04]) );
sbox2 u1( .addr(X[07:12]), .dout(S[05:08]) );
sbox3 u2( .addr(X[13:18]), .dout(S[09:12]) );
sbox4 u3( .addr(X[19:24]), .dout(S[13:16]) );
sbox5 u4( .addr(X[25:30]), .dout(S[17:20]) );
sbox6 u5( .addr(X[31:36]), .dout(S[21:24]) );
sbox7 u6( .addr(X[37:42]), .dout(S[25:28]) );
sbox8 u7( .addr(X[43:48]), .dout(S[29:32]) );

assign P[1:32] = {      S[16], S[7], S[20], S[21], S[29], S[12], S[28],
                        S[17], S[1], S[15], S[23], S[26], S[5], S[18],
                        S[31], S[10], S[2], S[8], S[24], S[14], S[32],
                        S[27], S[3], S[9], S[19], S[13], S[30], S[6],
                        S[22], S[11], S[4], S[25]};
"P kutusu permütasyonu yerleşimi rasgele değerler gelerek oluşur"
endmodule
```

2.4.DES Ana Algoritma Adımı:

```
module des(desOut, desIn, key, decrypt, roundSel, clk);

output  [63:0]  desOut; “des algoritma
input   [63:0]  desIn; “des algoritma girişi “
input   [55:0]  key; “56 bitlik anahtar bloğu”
input           decrypt;
input   [3:0]   roundSel; “ Multiplexer da istenen girişi çıkışa bağlayan girişle beraber dahil olan Select kısmıdır”
input           clk; “devreye clock girişi sağlar”

wire    [1:48]  K_sub; “Yer değiştirme permütasyonudur”
wire    [1:64]  IP, FP; “IP başlangıç permütasyonu ve FP ters permütasyonudur”
reg     [1:32]  L, R;”Algoritmada L sol kısımı R sağ kısımı ifade eder”
wire    [1:32]  Xin;”Döngünün sağına gelecek soldaki kısmı ifade eder”
wire    [1:32]  Lout, Rout;”Lout sağ kısımın bir önceki değerine eşittir Rout  Xin ile out parametresinin xor una  eşittir.”
wire    [1:32]  out; “Döngünün sağ kısmındaki f fonksiyonu çıkışıdır.”

assign Lout = (roundSel == 0) ? IP[33:64] : R;
assign Xin  = (roundSel == 0) ? IP[01:32] : L;
assign Rout = Xin ^ out;
assign FP = { Rout, Lout}; “Algoritmadaki son işlem olan ters permütasyon adımıdır”

crp u0( .P(out), .R(Lout), .K_sub(K_sub) ); “ P(out) P kutusu permütasyon kısmı çıkışıdır. R(Lout) Algoritmanın sağ kısmındaki 32 bitlik kısma algoritmanın sol tarafındaki L çıkışı eklenir ve xor döngüsüne dahil olur. K_sub yer değiştirme permütasyonudur.”

always @(posedge clk) “Always den sonra kullanılan @ işareti parantezden sonraki duruma geldiğinde process tetiklenecektir. Always komutu ile işlem tüm simülasyon boyunca çalışır.
” L <= #1 Lout; “Algoritmanın sol kısmı(left) Lout yani bu bölümün çıkışından küçükse gecikme ile döngü devam ettirilir.”

always @(posedge clk)
 R <= #1 Rout; “Algoritmanın sağ kısmı(right) Rout yani bu bölümün çıkışından küçükse gecikme ile döngü devam ettirilir.”

// Select a subkey from key.
key_sel u1(
        .K_sub(         K_sub       ),
        .K(             key         ),
        .roundSel(      roundSel    ),
        .decrypt(       decrypt     )
        ); “key_sel dizini altında  alt anahtarlar elde edilir.”
```

```
// Perform initial permutation
assign IP[1:64] = {     desIn[06], desIn[14], desIn[22], desIn[30], desIn[38], desIn[46],
                        desIn[54], desIn[62], desIn[04], desIn[12], desIn[20], desIn[28],
                        desIn[36], desIn[44], desIn[52], desIn[60], desIn[02], desIn[10],
                        desIn[18], desIn[26], desIn[34], desIn[42], desIn[50], desIn[58],
                        desIn[00], desIn[08], desIn[16], desIn[24], desIn[32], desIn[40],
                        desIn[48], desIn[56], desIn[07], desIn[15], desIn[23], desIn[31],
                        desIn[39], desIn[47], desIn[55], desIn[63], desIn[05], desIn[13],
                        desIn[21], desIn[29], desIn[37], desIn[45], desIn[53], desIn[61],
                        desIn[03], desIn[11], desIn[19], desIn[27], desIn[35], desIn[43],
                        desIn[51], desIn[59], desIn[01], desIn[09], desIn[17], desIn[25],
                        desIn[33], desIn[41], desIn[49], desIn[57] };
// Perform final permutation
assign desOut = {       FP[40], FP[08], FP[48], FP[16], FP[56], FP[24], FP[64], FP[32],
                        FP[39], FP[07], FP[47], FP[15], FP[55], FP[23], FP[63], FP[31],
                        FP[38], FP[06], FP[46], FP[14], FP[54], FP[22], FP[62], FP[30],
                        FP[37], FP[05], FP[45], FP[13], FP[53], FP[21], FP[61], FP[29],
                        FP[36], FP[04], FP[44], FP[12], FP[52], FP[20], FP[60], FP[28],
                        FP[35], FP[03], FP[43], FP[11], FP[51], FP[19], FP[59], FP[27],
                        FP[34], FP[02], FP[42], FP[10], FP[50], FP[18], FP[58], FP[26],
                        FP[33], FP[01], FP[41], FP[09], FP[49], FP[17], FP[57], FP[25] };
endmodule
```

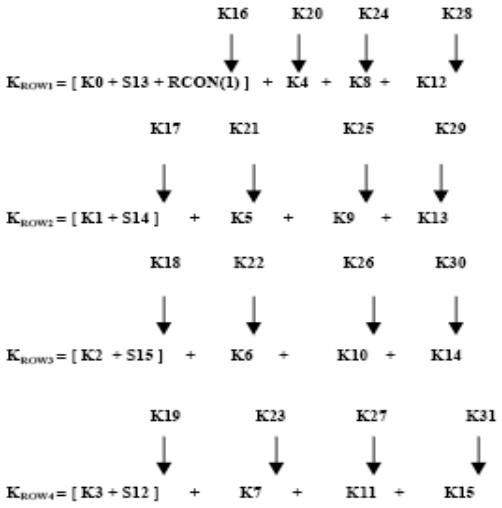
3-Model Sim Altında AES Algoritmasını Gerçeklemek İçin Kaynak Kodları:

3.1.Subbytes Fonksiyonu: S kutusunun olduğu katmandır.

```
Module    aes_sbox(a,d);
input   [7:0]    a;  “8 bit giriş “
output  [7:0]    d;  “8 bit çıkış “
reg     [7:0]    d;

always @(a)
        case(a)          “ 16*16 lık byte matrisin her elemanı S kutusu değeri ile eşleştirilir.”
          8'h00: d=8'h63;
          8'h01: d=8'h7c;
         İle devam eder 256 değer eşleştirilir.

          8'hfe: d=8'hbb;
          8'hff: d=8'h16;
        endcase

endmodule
```

3.2.S kutusundan Geçiş ve Anahtar Üretimi:

İlk olarak round constant fonksiyonu çalıştırılır.
Key Genişletme Algoritması:



1. Sij : Anahtar byte ı Kij nin S kutusu dönüşümüdür

2. RCON=ROUND CONSTANT: S kutusu ile anahtar dönüşümünde rol alan geçiş fonksiyonudur.

RCON(1) = 01H, RCON(2) = 02H,
RCON(3) = 04H, RCON(4) = 08H,
RCON(5) =10H, RCON(6) = 20H,
RCON(7) = 40H, RCON(8) = 80H,
RCON(9) = 1BH, RCON(10) = 36H
RCON(11) =6CH, RCON(12) = D8H,
RCON(13) = ABH, RCON(14) =4DH,
RCON(15) = 9AH,

```
module aes_rcon(clk, kld, out);
input           clk;
input           kld;
output [31:0]   out;
reg    [31:0]   out;
reg    [3:0]    rcnt;
wire   [3:0]    rcnt_next;

always @(posedge clk)
if(kld)         out <= 32'h01_00_00_00;
else            out <= frcon(rcnt_next);
assign rcnt_next = rcnt + 4'h1;
always @(posedge clk)
if(kld)         rcnt <= 4'h0;
else            rcnt <=  rcnt_next;
```

```
function [31:0]          frcon;  " Rcon fonksiyonu çalıştırılır"
input   [3:0]    i;
case(i) // synopsys parallel_case
4'h0: frcon=32'h01_00_00_00;
4'h1: frcon=32'h02_00_00_00;
4'h2: frcon=32'h04_00_00_00;
4'h3: frcon=32'h08_00_00_00;
4'h4: frcon=32'h10_00_00_00;
4'h5: frcon=32'h20_00_00_00;
4'h6: frcon=32'h40_00_00_00;
4'h7: frcon=32'h80_00_00_00;
4'h8: frcon=32'h1b_00_00_00;
4'h9: frcon=32'h36_00_00_00;
default: frcon=32'h00_00_00_00;
endcase
endfunction

endmodule
```

İkinci olarak anahtar üretimi yapılır.

```
module KEY_GENERATE_PROCESS(clk,kld,key,wo_0,wo_1,wo_2,wo_3);
input            clk;
input            kld; "anahtar genişletme modülünü başlatan bayraktır."
input   [127:0]key; "Anahtar uzunluğu 128 bit olarak girilir."
output  [31:0]  wo_0, wo_1, wo_2, wo_3; "Çıkış 32 bit ve 4 değişken tanımlanmıştır."
reg     [31:0]  w[3:0];
wire    [31:0]  tmp_w;"Anahtar çıkış değişkenin atandığı geçici s box parametresidir."
wire    [31:0]  subword;"S box daki byte girişlerinden çıkış üretir. Byte yerdeğişimini sağlar."
wire    [31:0]  rcon;"Anahtar genişletme fonksiyonu xor işlemine dahil edilir."

assign wo_0 = w[0];   "çıkış parametrelerine atama yapılıyor."
assign wo_1 = w[1];
assign wo_2 = w[2];
assign wo_3 = w[3];
always @(posedge clk)        w[0] <= kld ? key[127:096] : w[0]^subword^rcon;
always @(posedge clk)        w[1] <= kld ? key[095:064] : w[0]^w[1]^subword^rcon;
always @(posedge clk)        w[2] <= kld ? key[063:032] : w[0]^w[2]^w[1]^subword^rcon;
always @(posedge clk)        w[3] <= kld ? key[031:000] :
w[0]^w[3]^w[2]^w[1]^subword^rcon;
assign tmp_w = w[3];
"a giriş d çıkış olmak üzere s box geçişleri yapılır ve round constant fonk çalıştırılır."
aes_sbox u0(   .a(tmp_w[23:16]), .d(subword[31:24]));
aes_sbox u1(   .a(tmp_w[15:08]), .d(subword[23:16]));
aes_sbox u2(   .a(tmp_w[07:00]), .d(subword[15:08]));
aes_sbox u3(   .a(tmp_w[31:24]), .d(subword[07:00]));
aes_rcon r0(    .clk(clk), .kld(kld), .out(rcon));
endmodule
```

3.3.Şifreleme İşlemi:

```
module AES_ENCRYPT(clk,
                   rst,
                   ld,
                   done,
                   key,
                   text_in,
                   text_out);

input              clk, rst;"reset 1 bitlik kontrol girişi."
input              ld;"load parametresidir,128 bitlik metni yükler."
output             done;
input   [127:0]key;              "Anahtar uzunluğu,giriş ve çıkış 128 bittir."
input   [127:0]text_in;
output  [127:0]text_out;


wire    [31:0]  w0, w1, w2, w3;"128 bitlik anahtar 32 bitlik parçalara ayrılır."
reg     [127:0]text_in_r;
reg     [127:0]text_out;
reg     [7:0]   sa00, sa01, sa02, sa03;
reg     [7:0]   sa10, sa11, sa12, sa13;
reg     [7:0]   sa20, sa21, sa22, sa23;
reg     [7:0]   sa30, sa31, sa32, sa33;
wire    [7:0]   sa00_next, sa01_next, sa02_next, sa03_next; "Shiftrows fonksiyonu yani
satırların 3 adım ötelenme işlemi gerçekleşir "
wire    [7:0]   sa10_next, sa11_next, sa12_next, sa13_next;
wire    [7:0]   sa20_next, sa21_next, sa22_next, sa23_next;
wire    [7:0]   sa30_next, sa31_next, sa32_next, sa33_next;
wire    [7:0]   sa00_sub, sa01_sub, sa02_sub, sa03_sub;  "Subbytes fonksiyonu yani byte
ların yer değişimi işlemidir "
wire    [7:0]   sa10_sub, sa11_sub, sa12_sub, sa13_sub;
wire    [7:0]   sa20_sub, sa21_sub, sa22_sub, sa23_sub;
wire    [7:0]   sa30_sub, sa31_sub, sa32_sub, sa33_sub;
wire    [7:0]   sa00_sr, sa01_sr, sa02_sr, sa03_sr;
wire    [7:0]   sa10_sr, sa11_sr, sa12_sr, sa13_sr;
wire    [7:0]   sa20_sr, sa21_sr, sa22_sr, sa23_sr;
wire    [7:0]   sa30_sr, sa31_sr, sa32_sr, sa33_sr;
wire    [7:0]   sa00_mc, sa01_mc, sa02_mc, sa03_mc;   "Mixcolumns fonksiyonu yani
sütunların karıştırılması işlemi gerçekleşir "
wire    [7:0]   sa10_mc, sa11_mc, sa12_mc, sa13_mc;
wire    [7:0]   sa20_mc, sa21_mc, sa22_mc, sa23_mc;
wire    [7:0]   sa30_mc, sa31_mc, sa32_mc, sa33_mc;
reg             done, ld_r;
reg     [3:0]   dcnt;
```

```
always @(posedge clk) “Alttaki kısım MISC logic yani kontrol logic kısımdır”
        if(!rst) dcnt <=  4'h0;
        else
        if(ld)          dcnt <=  4'hb;
        else
        if(|dcnt)       dcnt <= dcnt - 4'h1;

always @(posedge clk) done <= !(|dcnt[3:1]) & dcnt[0] & !ld;
always @(posedge clk) if(ld) text_in_r <= text_in;  “ Burada algoritmadaki döngü sayısı için
parametre tanımlanır.”
always @(posedge clk) ld_r <= ld;


“Shiftrows fonksiyonu yani satırların ötelenme işlemi gerçekleşir.next ile adlandırılır.”
always @(posedge clk)       sa33 <= ld_r ? text_in_r[007:000] ^ w3[07:00] : sa33_next;
always @(posedge clk)       sa23 <= ld_r ? text_in_r[015:008] ^ w3[15:08] : sa23_next;
always @(posedge clk)       sa13 <= ld_r ? text_in_r[023:016] ^ w3[23:16] : sa13_next;
always @(posedge clk)       sa03 <= ld_r ? text_in_r[031:024] ^ w3[31:24] : sa03_next;
always @(posedge clk)       sa32 <= ld_r ? text_in_r[039:032] ^ w2[07:00] : sa32_next;
always @(posedge clk)       sa22 <= ld_r ? text_in_r[047:040] ^ w2[15:08] : sa22_next;
always @(posedge clk)       sa12 <= ld_r ? text_in_r[055:048] ^ w2[23:16] : sa12_next;
always @(posedge clk)       sa02 <= ld_r ? text_in_r[063:056] ^ w2[31:24] : sa02_next;
always @(posedge clk)       sa31 <= ld_r ? text_in_r[071:064] ^ w1[07:00] : sa31_next;
always @(posedge clk)       sa21 <= ld_r ? text_in_r[079:072] ^ w1[15:08] : sa21_next;
always @(posedge clk)       sa11 <= ld_r ? text_in_r[087:080] ^ w1[23:16] : sa11_next;
always @(posedge clk)       sa01 <= ld_r ? text_in_r[095:088] ^ w1[31:24] : sa01_next;
always @(posedge clk)       sa30 <= ld_r ? text_in_r[103:096] ^ w0[07:00] : sa30_next;
always @(posedge clk)       sa20 <= ld_r ? text_in_r[111:104] ^ w0[15:08] : sa20_next;
always @(posedge clk)       sa10 <= ld_r ? text_in_r[119:112] ^ w0[23:16] : sa10_next;
always @(posedge clk)       sa00 <= ld_r ? text_in_r[127:120] ^ w0[31:24] : sa00_next;

“Subbytes fonksiyonu çalıştırılır. Her bir byte s kutusundan geçirilir.”
assign sa00_sr = sa00_sub;
assign sa01_sr = sa01_sub;
assign sa02_sr = sa02_sub;
assign sa03_sr = sa03_sub;
assign sa10_sr = sa11_sub;
assign sa11_sr = sa12_sub;
assign sa12_sr = sa13_sub;
assign sa13_sr = sa10_sub;
assign sa20_sr = sa22_sub;
assign sa21_sr = sa23_sub;
assign sa22_sr = sa20_sub;
assign sa23_sr = sa21_sub;
assign sa30_sr = sa33_sub;
assign sa31_sr = sa30_sub;
assign sa32_sr = sa31_sub;
assign sa33_sr = sa32_sub;
```

“Sütunların karıştırılması işlemi gerçekleşir.Yeni değerler mc ile belirtilir.”

```
assign {sa00_mc, sa10_mc, sa20_mc, sa30_mc}  =
mix_col(sa00_sr,sa10_sr,sa20_sr,sa30_sr);
assign {sa01_mc, sa11_mc, sa21_mc, sa31_mc}  =
mix_col(sa01_sr,sa11_sr,sa21_sr,sa31_sr);
assign {sa02_mc, sa12_mc, sa22_mc, sa32_mc}  =
mix_col(sa02_sr,sa12_sr,sa22_sr,sa32_sr);
assign {sa03_mc, sa13_mc, sa23_mc, sa33_mc}  =
mix_col(sa03_sr,sa13_sr,sa23_sr,sa33_sr);
assign sa00_next = sa00_mc ^ w0[31:24];
assign sa01_next = sa01_mc ^ w1[31:24];
assign sa02_next = sa02_mc ^ w2[31:24];
assign sa03_next = sa03_mc ^ w3[31:24];
assign sa10_next = sa10_mc ^ w0[23:16];
assign sa11_next = sa11_mc ^ w1[23:16];
assign sa12_next = sa12_mc ^ w2[23:16];
assign sa13_next = sa13_mc ^ w3[23:16];
assign sa20_next = sa20_mc ^ w0[15:08];
assign sa21_next = sa21_mc ^ w1[15:08];
assign sa22_next = sa22_mc ^ w2[15:08];
assign sa23_next = sa23_mc ^ w3[15:08];
assign sa30_next = sa30_mc ^ w0[07:00];
assign sa31_next = sa31_mc ^ w1[07:00];
assign sa32_next = sa32_mc ^ w2[07:00];
assign sa33_next = sa33_mc ^ w3[07:00];
```

“Anahtar ekleme işlemi için bu parametreler ve döngü anahtarı matrise xor işlemiyle eklenir.”

```
always @(posedge clk) text_out[127:120] <= sa00_sr ^ w0[31:24];
always @(posedge clk) text_out[095:088] <= sa01_sr ^ w1[31:24];
always @(posedge clk) text_out[063:056] <= sa02_sr ^ w2[31:24];
always @(posedge clk) text_out[031:024] <= sa03_sr ^ w3[31:24];
always @(posedge clk) text_out[119:112] <= sa10_sr ^ w0[23:16];
always @(posedge clk) text_out[087:080] <= sa11_sr ^ w1[23:16];
always @(posedge clk) text_out[055:048] <= sa12_sr ^ w2[23:16];
always @(posedge clk) text_out[023:016] <= sa13_sr ^ w3[23:16];
always @(posedge clk) text_out[111:104] <= sa20_sr ^ w0[15:08];
always @(posedge clk) text_out[079:072] <= sa21_sr ^ w1[15:08];
always @(posedge clk) text_out[047:040] <= sa22_sr ^ w2[15:08];
always @(posedge clk) text_out[015:008] <= sa23_sr ^ w3[15:08];
always @(posedge clk) text_out[103:096] <= sa30_sr ^ w0[07:00];
always @(posedge clk) text_out[071:064] <= sa31_sr ^ w1[07:00];
always @(posedge clk) text_out[039:032] <= sa32_sr ^ w2[07:00];
always @(posedge clk) text_out[007:000] <= sa33_sr ^ w3[07:00];
```

“Sütunların karıştırılması fonksiyonu tanımlanır.”

```
function [31:0] mix_col;
input   [7:0]   s0,s1,s2,s3;
reg     [7:0]   s0_o,s1_o,s2_o,s3_o;
```

```
begin
mix_col[31:24]=xtime(s0)^xtime(s1)^s1^s2^s3;
mix_col[23:16]=s0^xtime(s1)^xtime(s2)^s2^s3;
mix_col[15:08]=s0^s1^xtime(s2)^xtime(s3)^s3;
mix_col[07:00]=xtime(s0)^s0^s1^s2^xtime(s3);
end
endfunction

function [7:0] xtime;
input [7:0] b; xtime={b[6:0],1'b0}^(8'h1b&{8{b[7]}});
endfunction
```

ÖZGEÇMİŞ

İlhan CİĞER 1986 yılında İstanbul da doğdu.2004 yılında Kurtuluş Lisesi nden mezun oldu.Aynı yıl İstanbul Üniversitesi Elektrik-Elektronik Mühendisliği bölümünde lisans eğitimine başladı.2008 yılında lisans eğitimini tamamladıktan 2 ay sonra Koç.net te Sistem Ve Network Uzmanı olarak çalışmaya başladı.2010 yılının mart ayında İstanbul Üniversitesi Fen Bilimleri Enstitüsü Elektrik-Elektronik Mühendisliği Anabilim Dalında Elektrik-Elektronik Mühendisliği yüksek lisans programına kabul edildi.Yüksek lisans ders dönemini tamamladıktan sonra 2011 yılının Nisan ayında askerlik görevini yerine getirmek için Koç.net teki görevinden ayrıldı.Aynı yılın Kasım ayında askerlik dönüşü Fox Tv de System and Network Engineer olarak çalışmaya başladı.Aynı zamanda tez aşamasında yarım bıraktığı yüksek lisans öğrenimine devam etti.