T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

RFID OKUYUCUNUN GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Elektrik Elektronik Müh. Zekeriya ALTUN

Enstitü Anabilim Dalı	**:**	**ELK.- ELEKTR. MÜH.**
Enstitü Bilim Dalı	**:**	**ELEKTRONİK**
Tez Danışmanı	**:**	**Yrd. Doç. Dr. Gürsel DÜZENLİ**

Nisan 2010

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

RFID OKUYUCUNUN GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Elektrik Elektronik Müh. Zekeriya ALTUN

Enstitü Anabilim Dalı : ELK.-ELEKTR. MÜH.

Enstitü Bilim Dalı : ELEKTRONİK

Bu tez 29/ 04 /2010 tarihinde aşağıdaki jüri tarafından Oybirliği ile kabul edilmiştir.

Yrd. Doç. Dr.	**Prof. Dr.**	**Yrd. Doç. Dr.**
Gürsel DÜZENLİ	**Etem KÖKLÜKAYA**	**Ahmet Turan ÖZCERİT**
Jüri Başkanı	**Üye**	**Üye**

ÖNSÖZ

RFID (Radyo Frekans Tanımlama) içinde mikroişlemci ve anten taşıyan, üzerinde bulunduğu nesnenin yapısı, hareketleri, kullanım tarihleri, stok ve çıkış depo bilgileri gibi birçok veriyi saklayan radyo frekansıyla haberleşebilen teknolojinin adıdır.

1960'lı yıllarda ABD savunma sanayisinde kullanılmaya başlanılan RFID teknolojisi, artık kuruşlarla ifade edilebilen Tag (Kart) maliyetlerinden dolayı serbest piyasada da rahatça kendine yer bulabilmektedir. 30 KHz'den 2,45 GHz'e kadar olan bir frekans aralığında haberleşebilen Tag ve okuyucu; düşük frekanslarda ucuz ama kısa mesafeli, yüksek frekanslarda ise pahalı ama uzun mesafeli işlem yapabilmektedir. Temassız gerçekleşen bu veri transferleri 5 cm'den 10 m'ye kadar uzaklıkları kapsamaktadır. Yeni tasarım aşamasında olan bazı okuyucular 100m mesafeyi test edebilmişlerdir. Yakın zamana kadar her Tag ve Okuyucunun farklı iletişim protokolleri olduğundan A firmasının Tag'ını B firması okuyamamaktaydı. Kabul edilen evrensel standartlar gelişmesiyle bu sorun ortadan kalkacaktır.

En yaygın uygulama olarak otoban geçişlerinde kullanılan KGS sistemi gösterilse de yakın zaman içerisinde otopark, demirbaş, insan, hayvan, ürün takibinde yaygın olarak kullanılacağı tahmin edilmektedir. Orta vade de barkodun yerini alacağı veya ortak kullanılacağı düşünülmektedir. Üç farklı kategorisi olan Tag'ların (Aktif-enerjili, Yarı Aktif-enerji, Pasif) yaygın kullanılan pasif türü bu tezin konusudur.

Sakarya Üniversitenin 2004 yılında 2009 yılın ortasına kadar kendi geliştirdiği RFID okuyucuyu kullanması üniversite yönetiminin desteğiyle olmuştur. Bu çalışmanın yapılabilmesi ve bu sonuçların elde edilebilmesi en başta Rektörümüz Sayın Prof. Dr. Mehmet Durman, Rektör Yardımcısı Sayın Prof. Dr. Hüseyin Ekiz, Mühendislik Fakültesi Dekanı Mehmet Ali Yalçın, Genel sekreter Doç Dr. Zafer Yılmaz Bey ve Sakarya Büyükşehir Belediyesi Ulaşım Daire Başkanlığına teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

AES	: Advanced Encryption standard
DES	: Data Encryption standard
EM	: Elektromanyetik dalga
HF	: Yüksek Frekans
LF	: Düşük Frekans
MIFARE	: Mikron Bilet Sistemi
RFID	: Radio Frequency Identification
Tag	: Temassız Kart (Etiket)
UHF	: Ultra Yüksek Frekans

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: RFID, Temassız kart, Tag, Mifare, Güvenlik

RFID sistemlerin dayandığı temel güvenlik Tag'ların şifreleme sistemidir. Ancak bu güvenlik, Tag'ların şifrelerinin kırılmasıyla artık anlamını yitirmiştir. Üretici firmalar konusunda gösterilen katı tutumlar bu güne kadar sistemlere fazla müdahale edilememesine neden olmuştur. Ancak, RFID sistemlerinin revaç bulması ve sistemin parasal hareketler üzerine kurgulanmış olması çeşitli saldırılara maruz kalmasını kolaylaştırmıştır. Birçok haberleşme ve Tag şifreleri kırılarak, sistemler kullanılamaz hale getirmiştir. Bu nedenle üretilen Tag'ların güvenliği arttırılırken bir yandan da Tag okuyucuların güvenliği arttırılmalıdır. Standart üretim Tag'lara karşılık hemen her servis sağlayıcı sistemlerinde kendi tasarladıkları okuyucuyu kullanmaktadır. Bu nedenle oluşan güvenlik açığı geliştirilen yeni nesil bir okuyucuyla aşılmaya çalışılmıştır. Sakarya Üniversitesinde geliştirilen bir okuyucu sayesinde veri alma, verme, azaltma, arttırma, bakiye bilgileri gibi transferler birbirlerinden ayrılarak farklı okuyuculara yönlendirilmiştir. Bu parçalı işlemler kopyalama ve şifre kırma işlemlerini oldukça zorlaştırmıştır. Güvenli veri transferi için daha fazla şifreleme, beraberinde uzun süreli işlemleri getirmiştir. Sakarya Üniversitesinde ve Sakarya Büyükşehir Belediyesinin otobüslerinde yapılan denemelerde bu sürenin işlemler için sorun teşkil etmediği görülmüştür. Bu tez de Okuyucu ile Tag arasındaki haberleşme (authentication) bilgileri referans alınarak çeşitli kombinasyon bilgileri karşılaştırılmak suretiyle kopyalama ve korsan sızmaların önüne geçilmiştir. Bunun için iki ardışık haberleşmede değişik sektörlere yazılan tarih, saat, ay ve yazılan bloklar çapraz karşılaştırılmış ve sızıntılar kara listeye alınarak engellenmiştir.

A NEW SECURE LOW COST MIFARE RFID READER DESIGN

SUMMARY

Key Words: RFID, Contactless cards, Tag, Mifare, Security

RFID and contactless smart cards have become pervasive technologies nowadays. Over the last few years, more and more systems adopted this technology as replacement for barcodes, magnetic stripe cards and paper tickets for a variety of applications. Contactless cards consist of a small piece of memory that can be accessed wirelessly, but unlike RFID tags, they also have some computing capabilities. Most of these cards implement some sort of simple symmetric-key cryptography, which makes them suitable for applications that require access control. The MIFARE Classic is the most widely used contactless smart card in the market. Its design and implementation details are kept secret by its manufacturer. Due to a weakness in the pseudo-random generator (CRYPTO1 stream cipher), it is able to crack the Crypto-1 in as little as 0.1 seconds if the attacker can access or eavesdrop the RF communications with the (genuine) reader. MIFARE classic card can be cloned in a much more practical card-only scenario, where the attacker only needs to be in the proximity of the card for a number of minutes, therefore making usurpation of identity through pass cloning feasible at any moment and under any circumstances.

In this thesis we designed a new MIFARE reader which overcomes of all the security vulnerability with a low cost solution. Our new MIFARE reader has been tested with success in two different places in Sakarya, Turkey. One of them was used at the Sakarya University Campus for different applications like cashless payment (refectory, canteen), Automatic loading and information points, card publishing, security (building access, door access). The other was tested at the public transport in Sakarya.

BÖLÜM 1. GİRİŞ

Radyo Frekansı ile Tanımlama (RFID = Radio Frequency Identification) karşılıklı bilgi transfer edebilen bir okuyucu ve bir karttan oluşup temassız olarak çalışan tanıma sistemleridir. Dünyada uzun zamandır yaygın olarak kullanılırken [1, 2, 3] (Wal-Mart, Metro…) ülkemizde yeni yeni uygulama alanları bulmaya başlamıştır. RFID hayatımızın bazı alanlarını basitleştiren, kullanım kolaylığı sağlayan bir teknolojidir. RFID dünya genelinde çok farklı alanlarında uygulanmış ve ekonomik kazançlar sağlamıştır. ABD'nin süpermarket devi Wal-Mart, ABD Savunma Sanayi, Alman süpermarket zinciri METRO sayılabilir. RFID birçok uygulama alanında kullanılmaktadır. RFID ile ürün sevkiyatında hatalı sevkiyatların büyük bir oranda azalmasına ve kaynak tasarrufu sağlanmıştır.

RFID'nin geleceğinin teknolojisi olmasına karşılık tam bir standardının olmaması ve yeterli sayıda uzmanın bulunmaması onun yavaş tanınmasına ve uygulanmasına neden olmuştur. Fakat, benzer teknolojilerle karşılaştırıldığında (Barkod yerine RFID etiket "Wal-Mart", smart kart yerine RFID kredi kartı "Kore", iButon yerine RFID bilet "Ankara", parmak izi yerine ele enjekte edilen RFID hafızalı çip "Meksika", …) kulanım kolaylığı, işlem hızı ve güvenirlikte alternatifinin olmadığı görülür. Bu nedenle gün geçtikçe artan bir ivmeyle hayatımızın farklı alanlarında uygulanabilecektir. Fakat bu uygulamalar, ortak bir çalışmanın sonucunda veya araştırmanın sonucunda olmamaktadır. Bir şirketin geliştirmiş olduğu uygulama başarılı olmasıyla eksik tarafları düşünülmeden benzerleri geliştirilmektedir. Buna en güzel örnek MIFARE kartlardır [4]. MIFARE kartlar dünya genelinde çok kullanılması, başka şirketlerin bunlara benzer kartların geliştirmesine neden olmuştur. MIFARE kartların güvenlik yapısı, güvenli haberleşmenin şirket tarafından çok gizli tutulmasıyla sağlanmıştır. Böylece okuyucu ve Tag arasındaki haberleşmenin ISO14443A [5] standardında olmakta fakat şifrelenme algoritması

şirket tarafından çok gizli tutulmaktadır. Güvenlik algoritmaların temel taşlarını oluşturan August Kerckhoffs [7] 1883 yılında güvenlik algoritmalarının gizli tutulmasıyla verinin gizliliğinin sağlanamayacağını ifade etmiştir.

Güvenlik algoritmalarının herkes tarafından bilinmesi ancak şifrenin gizli tutulması bu bilim sahasının önünü açacaktır. Bu ifadenin doğruluğu MIFARE kartların 10 yıl sonra şifrelerinin saniyeler içinde çözülebilmesiyle bir defa daha doğrulanmıştır. Bunu yanında AES şifreleme algoritmasının herkes tarafından bilinmesine rağmen şifrelenmiş verilerin çözülmesinin çok zor olması ve dünya genelinde şartsız en güvenli algoritmalardan biri olarak kabul edilmesinin August Kerckhoffs ifadesine ne kadar uygun düştüğü görülür. Bu bakımdan güvenlik algoritmalarının yüksek güvenlikli olması için algoritmanın kendisinin değil sadece şifresinin (anahtarın) gizli ve güvenli olması gerekir.

Sakarya Üniversitesi'nde Tag-Okuyucu arasındaki güvenlik sorunları ele alınmış, Tag'lara yapısal müdahalenin mümkün olmamasından dolayı güvenlik sorununun çözülmesi için standart okuyucu tabanlı yeni bir Okuyucu tasarlanmıştır. 2004 yılından itibaren Sakarya Üniversitesi Kampus Otomasyon sisteminde kullanılan standart MIFARE kartlar geliştirilen dinamik şifreleme algoritması ile yüksek güvenlikli RFID sistemi haline getirilmiştir. 2007 yılı sonunda MIFARE kartlarında açığa çıkan güvenlik sorunları (kart içeriği değiştirme, kopyalama ve okuyucuyu kart konusunda yanıltma) mevcut sistemi sadece kopyalama konusunda etkilemiştir.

Geliştirilen Okuyucu bu tezle beraber tekrar gözden geçirilmiş kopyalama açığı giderilerek Eylül 2007'de yeniden tasarlanmıştır. Bu sistem 15 ay boyunca kesintisiz çalışmıştır. Aynı sistem Şubat 2010 tarihinde de Sakarya Büyükşehir Belediyesi toplu taşıma araçlarında yine başarıyla test edilmiştir.

Kullanılan Mifare Classic tag ve okuyucusu arasındaki iletişim protokolündeki bir tasarım hatasından dolayı şifresi kırılabilen demode bir sistem haline dönüşmüştür. Piyasada çok yaygın bu ürünün bir anda devre dışı kalmasını engelleyecek bir donanım ve yazılım geliştirilerek sistem tekrar kullanılabilir bir hale getirilmiştir.

Tezimizde; Bölüm 2'de Tag'ın genel yapısı ve çalışma sistemi üzerinde duruldu, Bölüm 3'te tezin konusu olan MIFARE Tag'larda sağlanan mevcut güvenlik ile geliştirilmiş atak yöntemleri anlatıldı, Bölüm 4'te ise Okuyucu Tag arasındaki haberleşmede giderilememiş güvenlik açıklarının giderilebilmesi için yaptığımız çalışmanın detayları anlatıldı.

BÖLÜM 2. RFID

RFID'nin ilk olarak ne zaman bulunduğunu söylemek tam olarak mümkün değildir. Çalışma mantığı olarak değerlendirilirse, RFID'nin buluşunu radyo frekansın buluşu olarak alınabilir (1846 Micheal Faraday, Işık ve Radyo dalga elektro magnetik enerjinin parçasıdır). Uygulama mantığı olarak değerlendirilirse, İngilizlerin ikinci dünya savaşında dost/düşman uçakları ayırt etmek için kullanması alınabilir (Bavul büyüklüğündeki sistem). Fakat şimdiki uygulamalara göre değerlendirirsek RFID'nin temeli 1948 yılında Henry Stockman tarafından atılmıştır (Communication by Means of Reflected Power).

2.1. RFID

RFID, verileri temassız olarak alan/veren bir elektronik tanımlama teknolojisidir. Bu teknoloji sayesinde kişi ve ürün istenen ayrıntıda tanımlanabilir. Bu temel özellik kullanılarak çok farklı alanlarda uygulamalar başarıyla gerçekleştirilmiştir. RFID'nın kullanıldığı uygulama alanlarını üç gruba ayrılabilir:

1. Bilet uygulaması

Spor müsabakaları

Turizm

Otoyollar

Yemekhaneler

2. Takip uygulamaları

Otoparklar

Kargolar

Hayvan takibi

Kütüphane

Depolama

Lojistik

Geri dönüşüm

2. Güvenlik uygulamaları

Ürün koruma (mağazalardaki ürünler)

Giriş/çıkış

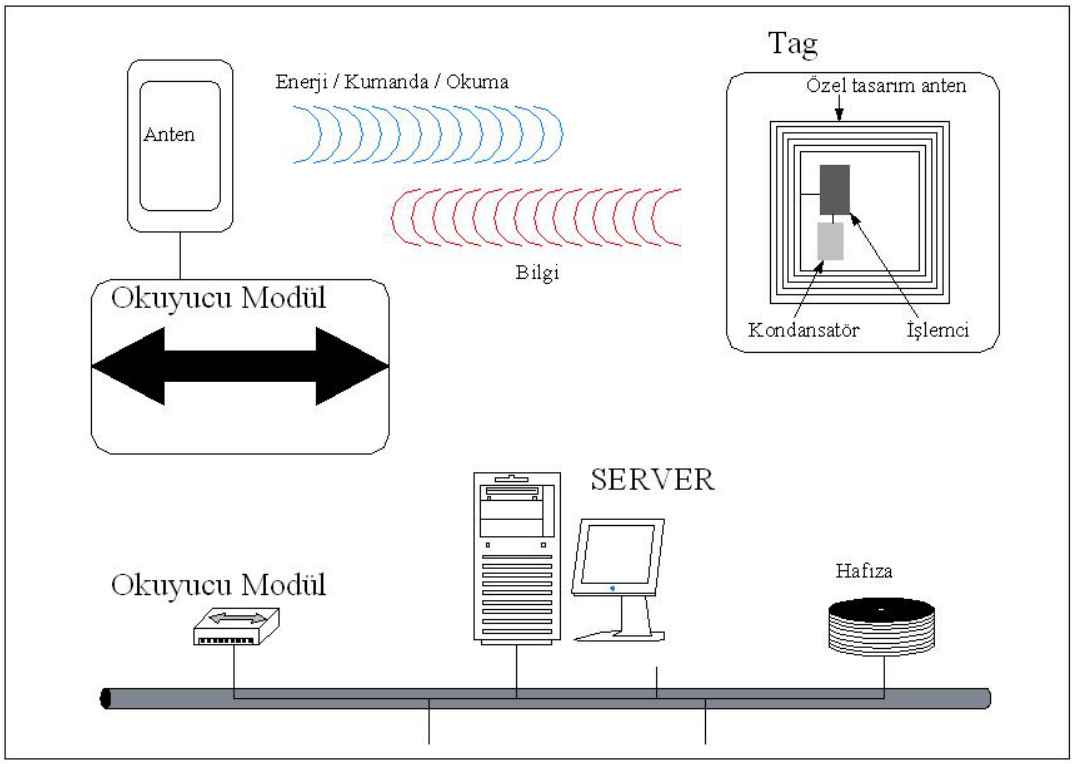
Pasaport

RFID uygulamaları üç parçadan oluşmaktadır;

Birincisi; RFID karttır. Kart yerine daha çok Tag, yaklaşım (proximity) veya Transponder (transmit ve response kelimelerin birleşmesinden oluşmaktadır) denmektedir.

İkincisi; okuyucu/yazıcı modüldür.

Üçüncüsü; sistem uygulamasıdır (Şekil 2.1.)



Şekil 2.1. RFID sisteminin genel yapısı

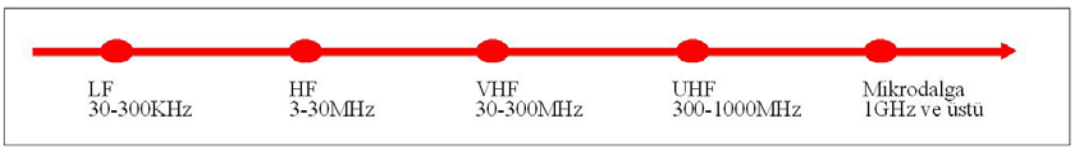
Üç farklı Tag vardır: aktif, pasif ve yarı pasif. Aktif ve yarı pasif Tag'larda pil varken pasif Tag'larda pil yoktur. Aktif Tag'lar devamlı olarak veri gönderirken (Savaş uçaklarının dost/düşman tanımlama sistemleri), yarı pasif Tag'lar okuyucudan işaret aldıklarında veri göndermektedirler (Örneğin: OGS – Otomatik

Geçiş Sistemi). Pasif Tag'ların çalışması için gerekli olan enerji ise okuyucu tarafından yayılan elektromanyetik dalgalardan sağlanır (Örneğin: KGS – Kartlı Geçiş Sistemi). Her uygulamanın diğerine göre avantajları ve dezavantajları olmasına karşılık bu avantaj veya dezavantaj uygulamaya göre değerlendirildiğinde değişebilir. Tag'ların karşılaştırılması Tablo 2.1.'de görülmektedir. Aktif ve pasif Tag'lar kendi aralarında hafıza yapısına göre üç gruba ayrılır. Bunlar sadece okunabilen, okunabilen/yazılabilen ve okunabilen/bir kere yazılabilen Tag'lardır.

Tablo 2.1. Aktif ve pasif Tag'ların karşılaştırılması

Aktif ve yarı pasif Tag		Pasif Tag	
Avantaj	Dezavantaj	Avantaj	Dezavantaj
Uzaktan okuyabilme	Büyük hacim	Küçük	Yakından okuyabilme
Çalışmaya hazır bekleme konumunda (Yarı pasif Tag), Devamlı çalışır konumunda (Aktif Tag)	Pile bağlı çalışma süresi	Sınırsız çalışma süresi	Güçlü okuyucu gereksimi
Büyük hafızalı	Pahalı	Ucuz	Sınırlı Hafızalı

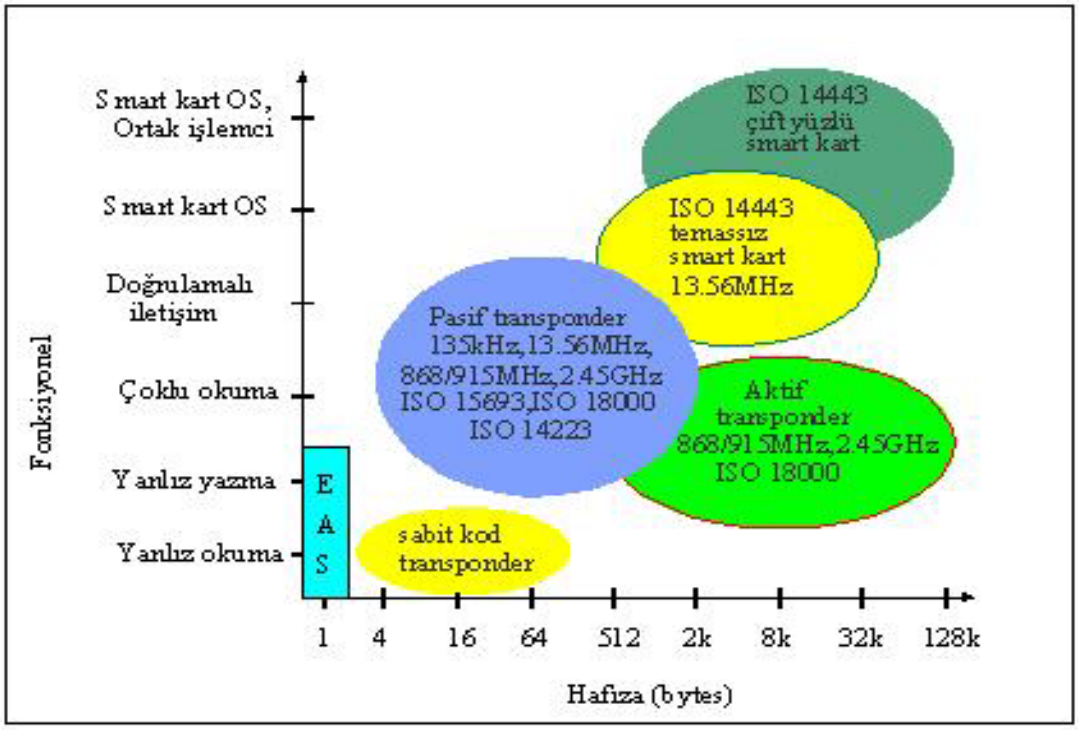
Tag'ların arasında diğer bir fark çalışma frekanslarıdır. Tag'ların çalışma frekansında bir standart olmadığından tüm frekans bant aralığında Tag'lar bulunmaktadır (Şekil 2.2.). En yaygın kullanılan frekanslar ise LF 100-135 kHz, HF 13.56 MHz, UHF 868/915 MHz, MW 2.45 GHz dir. Tablo 2.2.'de okuyucu ile Tag arasındaki mesafeye göre Tag'ların gruplandırılması görülmektedir. Tag'ların çalışma frekanslarının standart olmamasının en büyük nedeni yine uygulamalardan kaynaklanmaktadır (Şekil 2.3.)



Şekil 2.2. Frekans bant aralıkları

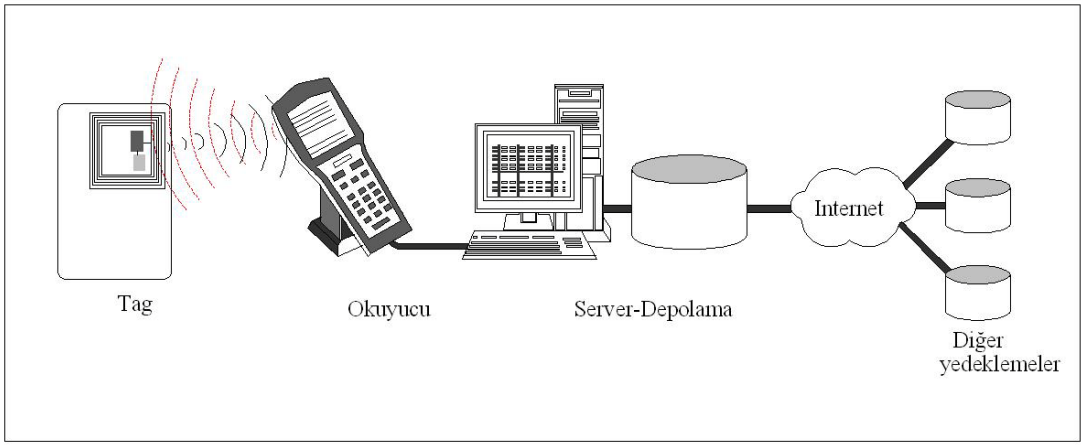
Tablo 2.2. Tag ile okuyucu arasındaki mesafeye göre Tag'ların gruplandırılması

Kısa aralık	Orta aralık	Geniş aralık
<=15 santimetre	<=5 metre	>500 metre
ISO 14443 A+B	ISO 15693	ISO 18000-xx
13.56 MHz, 125-134.2kHz	13.56 MHz, 125-134kHz	860-956 MHz(UHF) 2.4 GHz (Mikrodalga) 5.8 GHz (Mikrodalga
EM-alan,manyetik alan	EM-alan	EM-alan

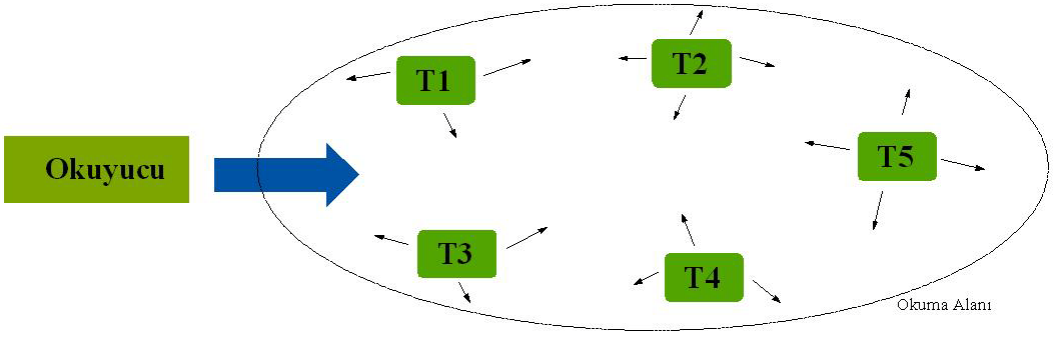


Şekil 2.3. Uygulamaya göre Tag'ların hafıza ve frekans değişimi [8]

Okuyucular Tag'lar gibi çok çeşitli olup uygulama alanına göre seçilirler. Genel olarak online ve offline olmak üzere iki ana gruba ayrılır. Online veri okuma ve belli zaman aralıklarında offline veriler veri okuma Şekil 2.4.'teki gibi veri tabanına iletilir. Okuyucuların birden çok Tag'ı hatasız ve kolayca okuması Şekil 2.5.'te gösterilmiştir. Okuyucular, Tag'larla haberleşme tekniği açısından üç gruba ayrılır: endüktif bağlama, EM dalga yayılımı ve kapasitif dalga (Şekil 2.6.).

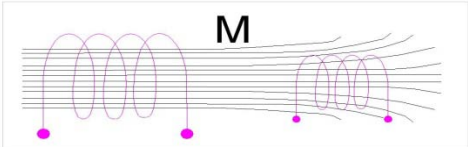
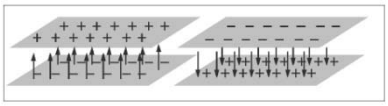


Şekil 2.4. Online ya da offline çalışan okuyucular



Şekil 2.5. Okuyucunun okuma alanı içinde birden fazla Tag okuması

Şekil 2.1. ve Şekil 2.4.'te Veri tabanı olarak adlandırılan yapının aslında okuyucudan alınan ve/veya verilen verilerin işlenmesi, depolanması ve internet üzerinden gerekli kişi ve/veya kurumlara göndermesi anlatılmıştır. Veri tabanı, RFID sistemin güvenliğini denetleyen en önemli parçasıdır. İstek dışı Tag kullanımları, yetkisiz müdahaleler ve geriye doğru incelemeler sonucu ileri dönük strateji geliştirilmesine olanak sağlamaktadır.

Endüktif bağlama 	Manyetik alan Yakın bölge	LF - HF 125 kHz - 13.56 MHz
EM dalga yayılımı 	EM dalga (Radar prensibi) Uzak bölge	UHF 862-956 MHz, 2.45 GHz
Kapasitif bağlama 	Elektrik alan Yakın bölge	LF 125 kHz

Şekil 2.6. Okuyucuların anten çalışma prensibi ve çalışma frekansı

2.2. MIFARE

RFID sistemlerin hızla yaygınlaşması bir standardın kabullenilmesine bağlıdır. Tablo 2.3'de RFID sistemler için hala kullanılmakta olan standartlar verilmiştir. Fakat Tag'ların üst seviye güvenliğinin sağlanabilmesi için şirketler kendilerine göre bir standart belirlemekte ve bunu kimseyle paylaşmamaktadırlar. Bilinmeyen bir sistemin güvenlik açığın bulunması çok zor olmaktadır. Buna ait en büyük örnek NXP şirketinin (Philips'in bir yan kuruluşu) geliştirmiş olduğu ve dünyada en çok kullanılan ve tercih edilen MIFARE kartlar ve okuyuculardır. MIFARE'nin kelime anlamı "Mikron Fare System" yani Mikron Bilet Sistemi. 1994 yılından bu güne dünya genelinde 1 milyardan fazla Tag ve 1 milyon üzerinde okuyucu satılmıştır. 2008 yılı itibarıyla dünya piyasalarının %85'i bu sistemi kullanmaktadır [9].

MIFARE Tag ve okuyucuların bu kadar tercih edilmesindeki sebep hiç kuşkusuz sağladığı güvenlik ve maliyetdir. MIFARE kartların türleri, özellikleri ve standartları Tablo 2.4.'te görülmektedir. Bu Tag ve okuyucularının kriptolama çalışma prensibi gizli tutulduğundan, 2007 yılının sonuna kadar hiçbir açıkları bulunamamıştır. Fakat gelişen teknoloji sayesinde ve MIFARE'e olan yoğun talep sonucunda kriptolama çalışma prensibinin açığı bulunmuştur. Mevcut MIFARE Tag'ların güvenlik açığının giderilmesi zor ve uzun süreç gerektirmesi özellikle uygulama geliştiren şirketleri etkilemiştir. Şekil 2.7.'den de anlaşılacağı üzere RFID sistemlerin güvenliği, maliyet ve Tag'ların enerji tüketimiyle doğru orantılıdır. Yüksek güveli Tag'ların üretilmesi için Tag'ların özel donanımla donatılması gerekmektedir. Özel donanım içeren Tag'ların çalışabilmesi için daha fazla enerji tüketmesi gerekmektedir. Bundan dolayı Tag'ların maliyeti de artmaktadır. Bu Tag'ları okuyacak ekipmanın da çıkış gücünün daha yüksek olması gerekeceğinden bu da ayrıca bir maliyet artışı demektir.

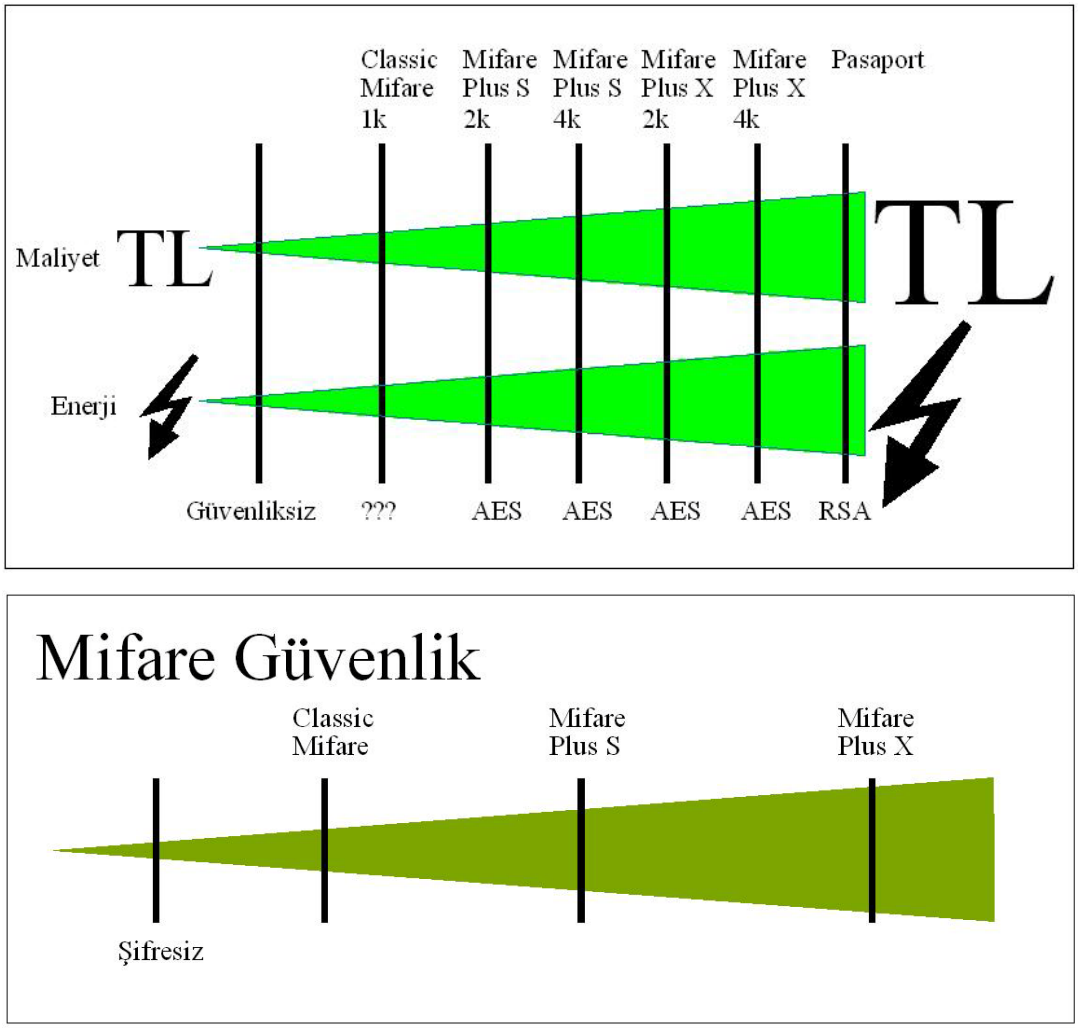
Tablo 2.3. RFID sistemler için standartlar

Standard	Uygulama	Açıklama	Frekans
ISO/IEC 14443	Kimlik Kartları	ISO/IEC 10 cm (3.94 inç) mesafeden temassız olarak kartın tanımlandığı, sinyal alışverişinin yapılabildiği (yazma-okuma) radyo frekans tabanlı haberleşmeli kartları kapsar. (Temassız Kart)	13.56 MHz
ISO/IEC 15693	Kimlik Kartları	ISO/IEC Temassız Kartlar için bir standarttır. 1 metre (3.3 feet) mesafeden temassız olarak kartın tanımlandığı, sinyal alışverişinin yapılabildiği (yazma-okuma) radyo frekans tabanlı haberleşmeli kartları kapsar. (Temassız Kart)	13.56 MHz
ISO 11784/11785	Hayvanlar için Kimlik Kartları	ISO 11784 Hayvanlara ait kimlik tanımlamayı kapsar. ISO 11785 ise hayvanlara ait kimliklerin okunabilmesine dairdir.	134.2 KHz
ISO 17363 DRAFT	Parça yönetimi	ISO toplu malzeme taşıma hakkında standart.	433 MHz
ISO/IEC 18000	Parça yönetimi	An ISO/IEC temassız arayüz standartı	
		• 2. aralık	135 KHz den az
		• 3. aralık	13.56MHz
		• 4. aralık	2.45 GHz
		• 5. aralık	860-960 MHz
		• 6. aralık	433 MHz
ISO/IEC TR24729-2	Geri dönüşüm	ISO/IEC RFID Kartların geri dönüşümünün sağlanması standardı	N/A
EPC Version 1.0/1.1 Şartnamesi	Tedarik zinciri	EPC fiziksel temassız kartlarla ilgili genel şartname ve Tag veri şartnamesi.	
		• 900 MHz Sınıfı 0 RFID Tag Şartnamesi	900 MHz
		• 860 MHz-930 MHz aralığı 1 RFID Tag Radyo Frekans ve lojik iletişim arayüz şartnamesi	860-930 MHz
AIAG B-11	Kimlik Hammadesi	Standart sanayi hammadde şartnamesi	862-928 MHz; 2.45 GHz

Tablo 2.4. MIFARE kartların türleri özellikleri ve ISO standartları

Türü	Kripto algoritma	Hafıza	ISO	Fiyat USD*
MIFARE Ultralight	Yok	64 Byte	14443A	0,46
MIFARE Ultralight C	3DES	192 Byte	14443A	0,54
MIFARE Classic	Crypto 1	4 Byte,	14443A	0,53
MIFARE Classic	Crypto 1	1kByte	14443A	0,56
MIFARE Classic	Crypto 1	4kByte	14443A	0,9
MIFARE PROx, SmartMX	Programlanabilir (Java)	30 kByte	14443A, 7816-4	3,12
MIFARE PROx, SmartMX	Programlanabilir (Java)	72 kByte	14443A, 7816-4	8,22
MIFARE DESfire	DES, 3DES veya AES	2kByte	14443A, 7816-4	1,1
MIFARE DESfire	DES, 3DES veya AES	4kByte	14443A, 7816-4	1,35
MIFARE DESfire	DES, 3DES veya AES	8kByte	14443A, 7816-4	1,6
MIFARE DESfire EV1	AES	2kByte	14443A, 7816-4	0,91
MIFARE DESfire EV1	AES	4kByte	14443A, 7816-4	1,09
MIFARE DESfire EV1	AES	8kByte	14443A, 7816-4	1,25
MIFARE Plus S	AES ve Crypto 1	2kByte	14443A	0,66
MIFARE Plus S	AES ve Crypto 1	4kByte	14443A	1
MIFARE Plus X	AES ve Crypto 1 Proximity check	2kByte	14443A	0,74
MIFARE Plus X	AES ve Crypto 1 Proximity check	4kByte	14443A	1,9

*(50.000 adet üzerinden 1 tane fiyatı- 1.04.2010)



Şekil 2.7. RFID sisteminin güvenliği, Tag'ların maliyeti ile enerji tüketimin karşılaştırılması

2.3. MIFARE Classic

NXP şirketinin dışında Legic, Atmel, Microchip ve Texas Instrumenst gibi Tag ve okuyucu üreten farklı şirketler de bulunmaktadır. Fakat bu ürünler MIFARE kadar yaygın kullanılmamaktadır. Bu nedenle MIFARE'e ait en yaygın kullanılan MF1ICS50 Tag bu çalışmada temel alınmıştır. MF1ICS50 veya MIFARE Classic 1K kullanılan adıyla1 kByte hafızalı, ISO/IEC 14443A haberleşme standardında, kimlik belirlemede mutual three pass authentication ISO/IEC 9798-2 standardında (bu standarda tam uymadıkları görüldü [10]) ve firma tarafından geliştirilmiş CRYPTO1 güvenlik algoritmasına sahiptir. 2007 yılın sonunda CRYPTO1 algoritması çözülerek dünyada büyük bir şaşkınlığa ve endişeye neden olmuştur

[11]. Mifare kartlar dünya genelinde çok kullanıldığından 2009 yılında Mifare Plus S ve Mifare Plus X kartları üretilmiştir. Bu kartlar Mifare Classic Okuyucularında da okunabilecek şekilde tasarlanmıştır. Fakat bu durumda güvenlik seviyesi Mifare Classic kartların seviyesine inmektedir. Yüksek güvenlikli ve AES kriptolamaya sahip bu yeni kartlar Mifare Classic kartına ait tüm güvenlik açıklarını gidermiştir.

MIFARE Classic pasif bir Tag ve 13,56 MHz'lik endüktif bağlamayla çalışmaktadır. Tag ve okuyucu arasındaki mesafeden dolayı buna proximity bağlama da denmektedir. Buna göre karta Tag yerine proximity kart da denmektedir (PICC-Proximity Integrated Circuit Card). Okuyucuya Proximity okuyucusu denmektedir (PCD-Proximity Coupling Device). PCD'nin PICC'yi algılayabildiği en uzak mesafe 10cm dir.

MIFARE Classic 1k ve 4k olarak üretilmekte, fakat 1k çok daha yaygın kullanıldığı için burada sadece onun üzerinde durulacaktır. Tablo 2.5.'de hafıza yapısı görülmektedir. Mifare kartlar aslında sadece hafıza kartlarıdır. Bu nedenle bu kartlar istense de Java kartlara kıyasla hiçbir şekilde üzerinde bir program çalıştırılamazlar. 1k, 16 adet sektöre ve her sektör 4 bloğa ayrılmıştır. Her blok ta 16 Byte uzunluğundadır. Fakat her sektörün iki adet farklı fonksiyona sahip şifreleme anahtarı vardır. Bu iki şifreleme anahtar ve buna bağlı aynı sektördeki diğer blokların (Blok 0,1 ve 2; DATA) kullanma özelliklerinin ayarlanması bir blokta (Blok 3; sector trailer) tutulmaktadır. Bu bloğun şifreleme ve ayarlama özellikleri Tablo 2.6.a., Tablo 2.6.b. ve Tablo 2.6.c.'de görülmektedir. Sektör trailer 001, 011 ve 101 olduğu sürece tüm ayarlar yeniden yapılabilmektedir. Buna karşılık 000 ve 100 durumunda sadece KEY A ve KEY B'ye ait şifreler değiştirilebilir ama Access Bitler değiştirilemez. Buna karşılık Sektör trailer 010, 110 ve 111 durumunda sektöre ait hiçbir şey değiştirilemez. Sonuç olarak her sektörün Blok 3 ile geri kalan blok 0, 1 ve 2'ye ait okuma, yazma, artırma, azaltma, aktarma, geri alma işlemlerin nasıl yapılacağı belirlenebilir. Böylece tek bir Tag ile farklı uygulamalar için kolayca ayarlanabilmektedir.

Üretici blog olan blok 0, sektör 0 entegrenin üretim aşamasında belirlendiğinden bu bilgi sadece okunabilmekte ve değiştirilememektedir. Bu blokta entegrenin

versiyonu ve kart seri numarası (UID) bulunmaktadır. Bu seri numara dünyada tektir. MİFARE Tag'larda ortaya çıkan güvenlik açıklarının önüne geçebilmek için Mifare kartların sahip olduğu arttırma, azaltma, aktarma, geri alma işlemlerinden hiçbiri kullanılmamıştır. Bu işlemler SAÜ'de geliştirilen okuyucu tarafından kullanılan mikroişlemciyle gerçekleştirilmiştir.

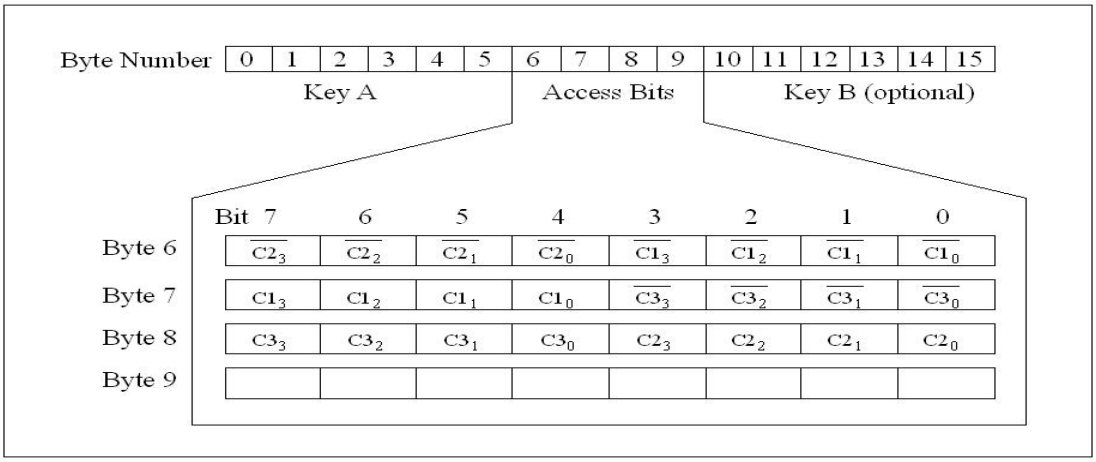
Tablo 2.5. MIFARE Classic'in hafıza yapısı

		Blok bit numaraları														
Sektör	Blok	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Tanımlama
15	3	Key A						Access Bits				Key B						Sector Trailer 15
	2																	Data
	1																	Data
	0																	Data
14	3	Key A						Access Bits				Key B						Sector Trailer 14
	2																	Data
	1																	Data
	0																	Data
:	:																	:
:	:																	:
1	3	Key A						Access Bits				Key B						Sector Trailer 1
	2																	Data
	1																	Data
	0																	Data
0	3	Key A						Access Bits				Key B						Sector Trailer 0
	2																	Data
	1																	Data
	0																	Üretici Bloğu

Tablo 2.6.a. MIFARE Classic Tag'ın her sektörüne ait şifreleme ve özellik ayarları (sector trailer)

Giriş Bilgileri				
Bit No	Geçerli işlem		Blok	Tanım
$C1_3$ $C2_3$ $C3_3$	Okuma, yazma	→	3	sector trailer
$C1_2$ $C2_2$ $C3_2$	Okuma, yazma, arttırma, azaltma transfer geri getirme	→	2	data blok
$C1_1$ $C2_1$ $C3_1$	Okuma, yazma, arttırma, azaltma transfer geri getirme	→	1	data blok
$C1_0$ $C2_0$ $C3_0$	Okuma, yazma, arttırma, azaltma transfer geri getirme	→	0	data blok

Tablo 2.6.b. MIFARE Classic Tag'ın Access Bits'e ait şifreleme ve özellik ayarları (sector trailer)



Tablo 2.6.c. MIFARE Classic Tag'ın sektör detayları (sector trailer)

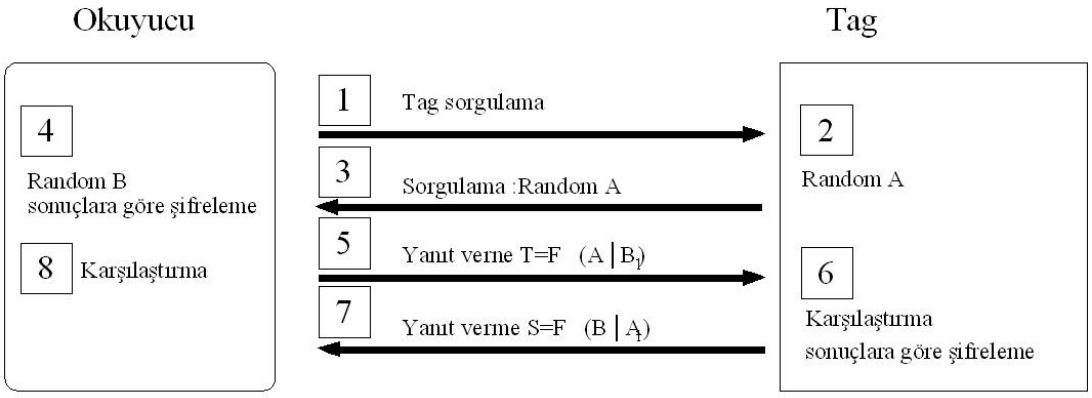
Giriş Bilgileri									
Giriş Bit			Giriş bilgi bloğu						Açıklama
			KEY A		Access Bits		KEY B		
C1	C2	C3	okuma	yazma	okuma	yazma	okuma	yazma	
0	0	0	never	key A	key A	never	key A	key A	key B okunabilir
0	1	0	never	never	key A	never	key A	never	key B okunabilir
1	0	0	never	key B	key A | B	never	never	key B	
1	1	0	never	never	key A | B	never	never	never	
0	0	1	never	key A	key A	key A	key A	key A	key B okunabilir taşınabilir durum
0	1	1	never	key B	key A | B	key B	never	key B	
1	0	1	never	never	key A | B	key B	never	never	
1	1	1	never	never	key A | B	never	never	never	

Giriş Bilgileri							
Giriş Bit			Giriş bilgi bloğu				Açıklama
C1	C2	C3	okuma	okuma	arttırma	yazma	
0	0	0	key A|B	key A|B	key A|B	key A|B	biçim aktarma
0	1	0	key A|B	never	never	never	okuma/yazma bloğu
1	0	0	key A|B	key B	never	never	okuma/yazma bloğu
1	1	0	key A|B	key B	key B	key A|B	değer bloğu
0	0	1	key A|B	never	never	key A|B	değer bloğu
0	1	1	key B	key B	never	never	okuma/yazma bloğu
1	0	1	key B	never	never	never	okuma/yazma bloğu
1	1	1	never	never	never	never	okuma/yazma bloğu

MIFARE Classic'in haberleşme protokolü ISO14443 standardına göredir. Buna göre okuyucu okuma alanında bir Tag seçmesi gerekir. Okuyucu okuma alanında birden fazla Tag arasında hatasız bir tanesini seçebilmesi sahip olduğu "Anticollision" (çarpışma önleyici) protokolünden kaynaklanmaktadır. Okuyucu, okuma alanı içerisindeki tüm Tag'ların seri numaralarını (UID-Unique Identifier) göndermesini

ister. Manchester-Kodlama sayesinde okuyucu okuma alanında birden fazla Tag'ın cevap gönderdiğini anlar. Eğer birden fazla Tag varsa okuma alanında, ikili arama mantığına benzer bir yöntemle, okuyucu aranan Tag'ın seri numarasını bit bit artırarak sadece aranan Tag en son cevap vermesini sağlamaktadır.

Aranan Tag bulunduktan sonra onunla güvenli haberleşmeye geçilmesi Şekil 2.8.'de gösterilmiştir (mutual challenge and response authentication). Güvenli haberleşme geçmek için Tag önce rastgele bir sayı (challenge) üretir ve okuyucuya gönderir. Okuyucu okumak istediği sektörün şifre anahtarını bu rastgele sayı ile şifreleyerek (response) ve kendi ürettiği bir rastgele sayıyla birlikte Tag'a geri gönderir. Tag, bunu çözümleyerek sektörün şifresini elde ederek sektörün şifresiyle karşılaştırır, doğru değilse okuyucudan yeniden seri numarasını okuma talebi gelene kadar Tag haberleşmeyi durdurur. Eğer doğru ise güvenli haberleşmenin sağlanmış olduğunu (authentication) okuyucuya bildirmek için onun gönderdiği rastgele sayı ile cevap verir. Böylece okuyucu ve Tag arasında güvenli haberleşme sağlanmış olmaktadır. Bundan sonra okuyucu mevcut sektördeki bloklardan, şifresinin yetki çerçevesi içinde, işleyebilir (yazma, okuma, artırma, azaltma, çoğaltma, geri alma)



Şekil 2.8. Karşılıklı kimlik doğrulama sorgulaması yöntemi

2.4. Güvenlik

Bir şifreleme algoritmasının sağladığı güvenlik için iki ana görüş vardır. Bunlardan birincisi şifreleme algoritmasını bilen kişi sayısıyla güvenlik ters orantılı olması görüşüdür. Başka bir ifadeyle şifreleme algoritmasının güvenliği, o algoritmayı ne kadar az kişi bilirse o düzeyde güvenlidir. Bu "security-through-obscurity" olarak ifade edilmektedir. MIFARE bu güvenlik anlayışı ile üretilmiştir. İkinci görüş ise

birincisinin tersidir. Bu görüşte ise güvenlik algoritması tam olarak herkes tarafından bilinmesiyle algoritmanın güvenliği tescil edilmiş olur görüşüdür. Dünyaca ünlü AES şifreleme algoritması bu anlayış ile geliştirilmiştir. Bu algoritmanın açık kodları Ek A'da verilmiştir. Fakat her iki görüşün bir ortak noktası vardır. Şifreleme algoritması ne kadar yaygın kullanılıyorsa o oranda çözülebilirliği artmaktadır. Ayrıca, şifre algoritması ne kadar iyi olursa olsun şifreleme için seçilen şifrenin de zor olması gerekmektedir. Burada zor ifadesi tahmin edilme olasılığı olabildiğince düşük olması demektir. Bu nedenle şifreler bilgisayar yardımıyla ve olasılık hesabıyla belirlenmektedir. Şifreler doğrudan sıkça kullanılan karakterlerden oluşabildiği gibi sadece ikili veya heksadesimal de olabilmektedir. Şifre algoritmalarına kriptolama denmektedir. Bu algoritmaları ve/veya ürettiği şifreleri çözme işine ise kripto analiz denmektedir. Kriptolama ve şifre belirleme yöntemlerinin gelişmesine paralel olarak kripto analiz yöntemleri de gelişmektedir. Şu anda en yaygın kullanılan küme kriptolama yöntemi olarak AES, Blow fish, Serpent, Twofish ve sırasal kriptolama yöntem olarak RC4 gösterilebilir. Küme kripto analiz yöntemler olarak; Brute force, Davies, Boomerang, Slide, XSL, Meet in the milde ve sırasal kripto analiz yöntemler olarak; Correlation, Correlation immunity gösterilebilir. MIFARE sıralı kriptolama yöntemi olarak Shift registeri kullandığı bulunmuştur [12].

2.5. Kriptolama

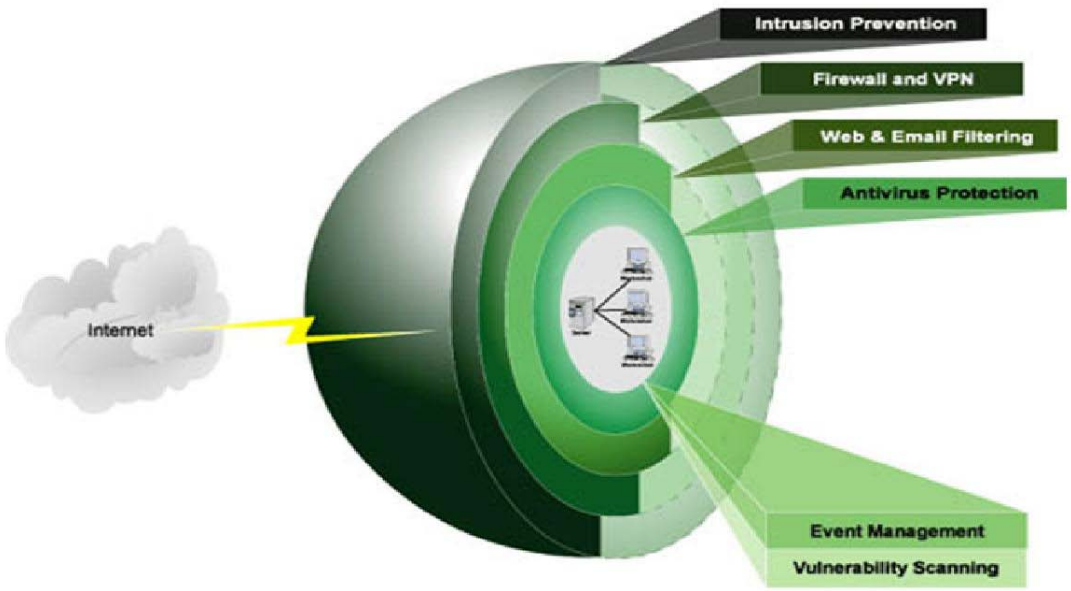
Kriptolamanın amacı, gizli bir belgenin güvenli olarak bir yere iletilebilmesidir. Bunu sağlamak için orijinal belge şifreli belgeye çevrilir. Şifreli belge gönderilmesi istenen yere gönderilir. Alıcı şifreli belgeyi deşifre etmesiyle belge güvenli olarak iletilmiş olur. Bu basit açıklama tüm modern kriptolamalar için geçerlidir. Aralarındaki fark orijinal belgeyi şifrelemek ve deşifre etmekte için kullanılan şifre anahtarların şifrelenmiş belgeden elde edilme zorluğudur. MIFARE kriptolamayı gizli tutmakla şifrelenmiş bilginin elde edilemeyeceğini düşünmüştür. Fakat 2007 yılından itibaren çok sayıda araştırmacı bunu deşifre ederek MIFARE kartların güvenliğini aşmıştır [11, 12, 13, 14, 15, 16, 17, 18]. Buna karşılık kriptolama için en yaygın kullanılan AES'in şifreleme ve şifre çözme algoritması tamamen açık ve herkes tarafından bilinmesine karşılık, halen şifrelenmiş belgeden orijinal belge elde

edilebilmiş değildir. Fakat bu güne kadar birçok çalışma [19, 20, 21, 22, 23] AES'inde çözülebileceğini ifade etmektedir. Bunlardan ilki [19] XSL ile $2^{2^{\circ\circ}}$ işlemle, teoriksel olarak, çözülebilir olduğu söylenmektedir. Fakat bu teoriksel ifadelerin gerçekleştirilebilirliği incelendiğinde AES'in çözülemez olduğu varsayılmaktadır. Buradan da anlaşılmaktadır ki güvenlik; algoritmanın gizlenmesiyle değil (tam tersi olarak algoritma herkesçe bilinir) şifrenin elde edilememesiyle sağlanır.

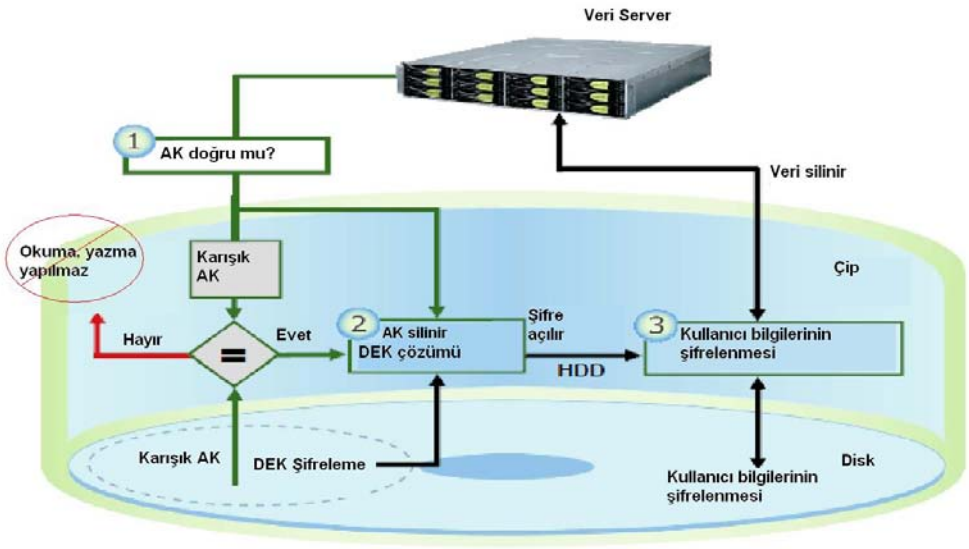
BÖLÜM 3. RFID GÜVENLİK

3.1. Giriş

RFID sistemlerin artarak gelişmesiyle uzaktan otomatik veri işlenmesi esnasında bu verilerin güvenliğinin sağlanması daha da önem kazanmıştır. RFID sisteminin güvenli haberleşmesinin dışında Tag'lardan okunan veya yazılacak verilerin ağ üzerinden iletilmesi, bilgisayar ve/veya sunucudaki verilerin güvenliğinin de sağlanması gerekmektedir. Fakat RFID sistemin dışındaki sistemlerin (verilerin ağ üzerinden gönderilmesi ve depolanması) Şekil 3.1., Şekil 3.2.'de görüldüğü gibi uzun zamandır kullanılıyor olmasından dolayı güvenliğinin sağlanması üzerine çok sayıda çalışmalar yapılmış ve başarıyla uygulanmıştır [7, 31, 34, 35, 36]



Şekil 3.1. Ağ güvenlik yapısı



Şekil 3.2. Veri depolama güvenliği

Ağ güvenliğinin evrenselliğinden dolayı detaylı güvenlik çalışmaları yapılmasına karşılık MIFARE RFID sisteminin çalışma prensibi 2007 yılı sonuna kadar şirket tarafından gizli tutulduğundan Tag güvenliği ve Tag okuyucu haberleşme güvenliği gelişememiştir. MIFARE Tag'ları dünya genelinde çok yaygınlaşması açıkların ortaya çıkmasına neden olmuştur. Bu açıklar 2007 yılı sonu itibarıyla yayınlanmıştır. Bu nedenle mevcut Mifare sistemin güvenli olarak kullanılması mümkün olamamaktadır. Çözüm olarak tüm sistem daha güvenli bir sistemle değiştirilmeli veya mevcut sistemdeki güvenlik açığını oluşturan noktalar lokal çözülerek güvenli hale getirilmelidir. Her iki durumda da zaman, uyumsuzluk ve maliyet gibi olumsuzluklarla karşılaşılmaktadır. Bu çalışmada en düşük maliyet olan ikinci durum ele alınarak mevcut sistem açığı giderildi. Bu durumda tezin konusu olan çalışmada 2004 yılından itibaren Sakarya Üniversitesi Kampus Otomasyon sisteminde kullanılan standart MIFARE Tag'larının okunması dinamik şifreleme algoritması ile yüksek güvenlikli RFID sistemi geliştirilmiştir [32]. Fakat bu sistem 2007 yılı sonunda MIFARE'in Tag ve okuyucu haberleşmesinin açığının bulunmasıyla yetersiz kalmıştır. MIFARE'nin Tag okuyucu arasındaki haberleşme açığı uygulanan yarı-online sistemle giderilmeye çalışılmıştır.

3.2. RFID Doğrulaması

RFID sistemlerinde Tag ve okuyucu arasındaki doğruluğun kanıtlanmasının (authentication) sağlanmasıyla güvenli haberleşme başlamaktadır. Doğruluk sağlanması işlemi gerçekleşene kadarki aşamaların güvenli olmasıyla sistem güvenli olmaktadır. MIFARE'nin en zayıf olduğu nokta doğruluk sağlanması aşamalarından kaynaklanmaktadır. Doğruluk sağlanması işleminin yüksek güvenlikli olması Tag'ın maliyetini artırır. Aşağıda belli başlı Doğruluk sağlanması yöntemleri Tag maliyet artışına göre verilmiştir.

3.2.1. Seri numara doğrulaması

Bu doğrulama yönteminde okuyucu Tag'ın seri numarasını okuyarak kendi hafızasındaki seri numaralarla karşılaştırır. Kendisinde kayıtlı ise işleme onay vermektedir. Bu tür doğrulama yöntemler giriş/çıkış sistemlerine benzer uygulamaklarında kullanılmaktadır. Fakat Tag'ların seri numarası kopyalanabildiğinden bunların güvenlik düzeyi çok düşüktür.

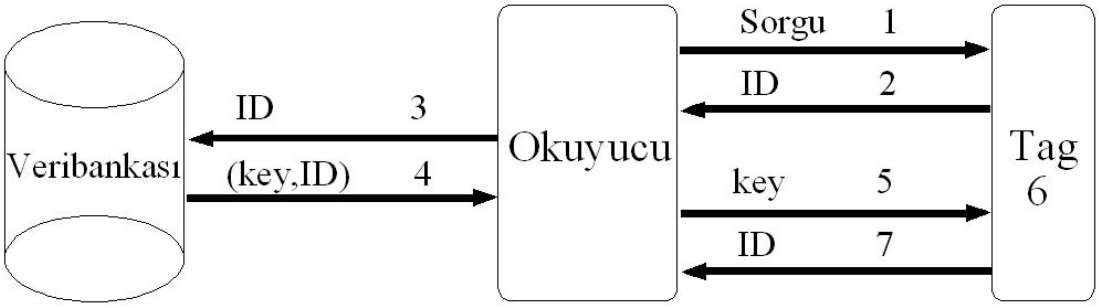
3.2.2. Açık şifreli doğrulama

Bu doğrulama yönteminde şifre okuyucu ve Tag'da saklanmaktadır. Okuyucu Tag'a şifreyi gönderir o da aldığı şifreyi kendi şifresiyle kontrol ederek doğru ise verilerin okuyucu tarafından okunmasına izin verir. Fakat şifreler Tag'a doğrudan gönderildiğinden başka okuyucular da şifreleri elde edebilir. Bunun için değişken şifreleme kullanılmaktadır. Bu da yazılabilen Tag gereksinimini doğurmaktadır. Bu tür uygulamalarda okuma yapılacak sayı kadar tek kullanımlı şifreler Tag'a yüklenir ve bu şifrelerde okuyucu tarafından da bilinir. Böylece tek kullanımlı şifrenin açıkta gitmesiyle güvenlik sağlanmış olur. Bu tür uygulamalar futbol, sinema, konser kombine biletlerde kullanılmaktadır. Bu mantık şu anda Internet üzerinden banka işlemlerini yapabilmek için kullanılan tek kullanımlık şifrematik olarak adlandırılan cihazlarda da kullanılmaktadır. Bu yöntemin tek farkı şifrematik kullanılsın veya

kullanılmasın her 5 saniye içinde yeni bir şifre üretmesidir. Böylece kişiye özel olan bu şifrematikler her 5 saniye içinde yeni bir şifre ürettiklerinden kullanım ömürleri ortalama 3 yıldır. Bu sürenin sonunda şifrematik yeniden programlanmalıdır. Aynı şifreyi Bankadaki sunucunun de üretiyor olması çok üst düzeyde bir güvenliğe neden olmaktadır. Çünkü bir sonraki şifreyi sadece şifrematik ve sunucu bilmektedir.

3.2.3. Hash-Lock doğrulama

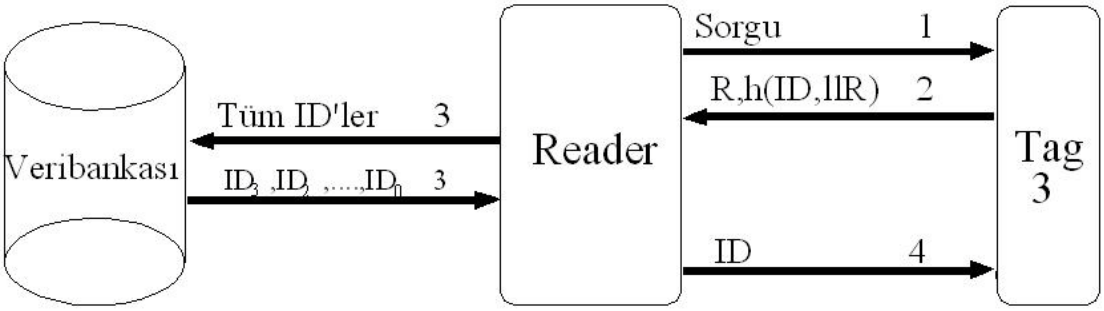
Bu doğrulama yönteminde tek yön fonksiyonlar kullanılmaktadır. Tek yön fonksiyonlar ileri doğru çok kolay hesaplanırken geriye doğru çok zor hesaplanmaktadır. Buna Hash hesaplama denir ve en yaygın kullanılan algoritma MD5 VE SHA-4 dır. Tag'ın ilk kullanımından önce bir çift üretilir bunlar şifre ve buna bağlı Hash değeridir. Bunlar okuyucuya kayıt edilir. Ayrıca bu Hash değeri Şekil 3.3.'teki gibi Tag'a da kayıt edilir. Doğrulama işleminde (1) Tag ilk olarak Hash değerini okuyucuya gönderir (2). Okuyucu bu Hash değerini hafızasında arayarak (3) şifreyi bulur (4) ve Tag'a gönderir (5). Tag bu şifreyi tek yön fonksiyonla hesaplayarak ilk gönderdiği Hash değeri ile karşılaştırır (6). Eşitse verilerini okumaya izin verir (7). Bunun içinde Tag tek yönlü fonksiyonları hesaplayabilecek donanıma sahip olmalıdır.



Şekil 3.3. Hash-Lock doğrulama

3.2.4. Rastgele Hash-Lock doğrulama

Önceki Hash-Lock doğrulama işleminde okuyucu Tag arasındaki haberleşme kayıt edilerek sahte bir okuyucu ile Tag'a işlem yapılabilir. Bunun önüne geçmek için Şekil 3.4.'te gösterilen rastgele Hash-Lock doğrulama yöntemi kullanılır. Bu yöntemde farklı olarak Doğruluk sağlanması işleminde (1) Tag rastgele bir sayı okuyucuya gönderir (2). Okuyucu ve Tag bu rastgele sayıyı kullanarak tek yönlü fonksiyonla Hash değeri hesaplarlar (3). Okuyucu hesaplama sonucu olan Hash değerini Tag'a gönderir (4) ve o da kendi hesapladığı Hash değeri ile karşılaştırır. Eşitse okuyucuya verilerini okuma izni verir. Hash değerini Tag tarafından üretilen rastgele sayı ile yapıldığından okuyucu ile Tag arasında haberleşmenin dinlenmesiyle bir sonuç elde edilememektedir. MIFARE'in zayıf noktası bu rastgele sayının çok tekrar etmesinden kaynaklanmaktadır. Bu işlemin gerçeklememesi için Tag tek yönlü fonksiyonun hesaplamasının yanında rastgele sayıyı da üretebilme kabiliyetinde de olmalıdır.

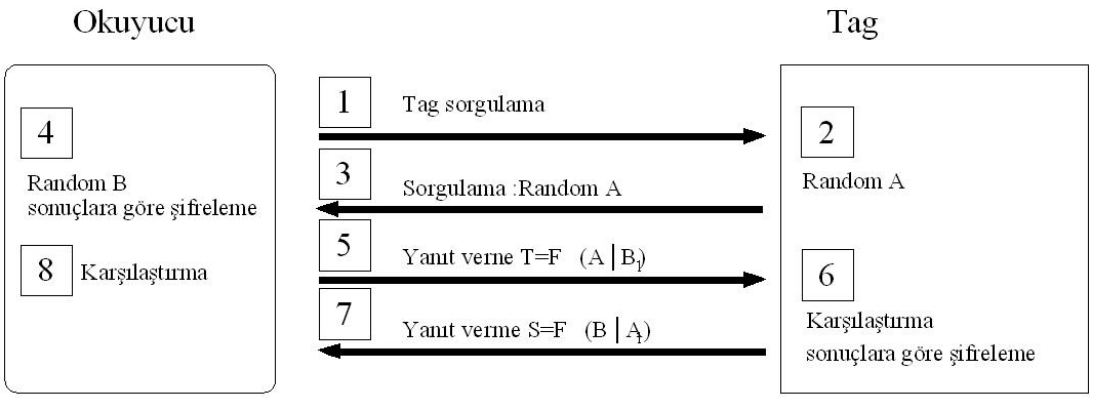


Şekil 3.4. Rastgele Hash-Lock doğrulama

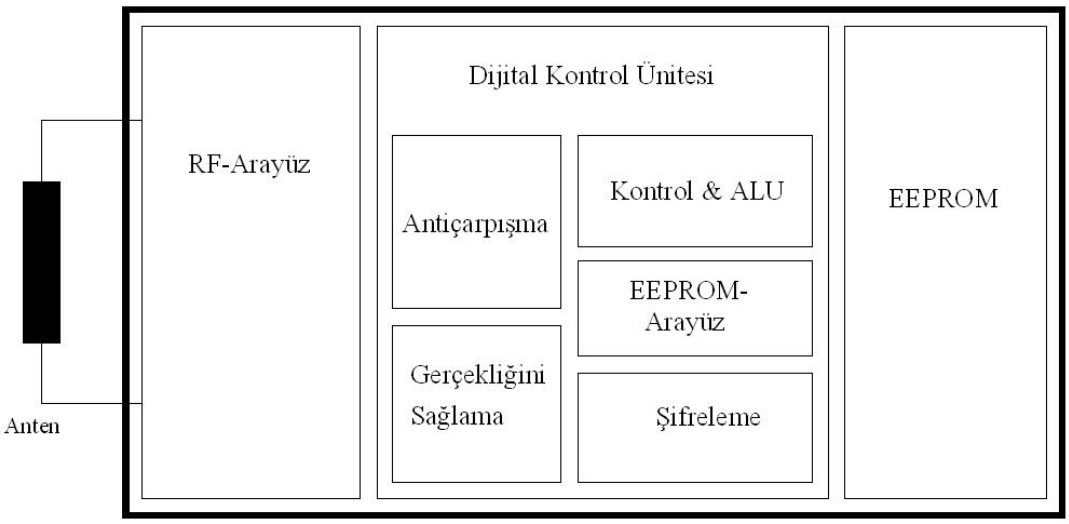
3.2.5. Meydan okuma-Yanıt verme doğrulaması

Şekil 3.5.'te gösterildiği üzere bu yöntemde okuyucu Tag'a bağlanmak isteğini gönderirken (1) Tag ona rastgele (2) bir sayı A göndermektedir (3). Okuyucu aldığı A sayısını ve kendi ürettiği rastgele B sayısı ile kriptolama algoritmasıyla ve gizli bir K şifre anahtarıyla bir sayı üretmektedir (4). Bu sayıyı Tag'a gönderir (5) o da aynı kriptolama algoritması ve K şifre anahtarıyla sayıyı çözer ve buradaki A sayıyı kendi A sayısıyla karşılaştırır (6). Eşitse yeni bir rastgele sayı üreterek çözdüğü B sayısı ile

kriptolama algoritmasıyla ve K şifre anahtarıyla bir sayı üretmekte ve okuyucuya bunu göndermektedir (7). Okuyucuda bunu çözerek kendi gönderdiği B anahtarıyla Tag'dan aldığı B anahtarını karşılaştırır (8). Aynı ise okuyucu ile Tag arasında doğrulama sağlanmış olur. MIFARE Şekil 3.6.'de detayları gösterilen bu yöntemi kullanmaktadır. Fakat rastgele sayıyı iyi üretemediği için haberleşme kayıt edilerek şifre çözülebilmektedir. Önceki yöntemlerde haberleşme şifresiz yapılırken bu yöntemde haberleşme şifreli yapılmaktadır. Bunun için Tag ilave olarak kriptolama yapabilecek yeteneğe sahip olmalıdır.



Şekil 3.5. Meydan okuma-Yanıt verme doğrulaması



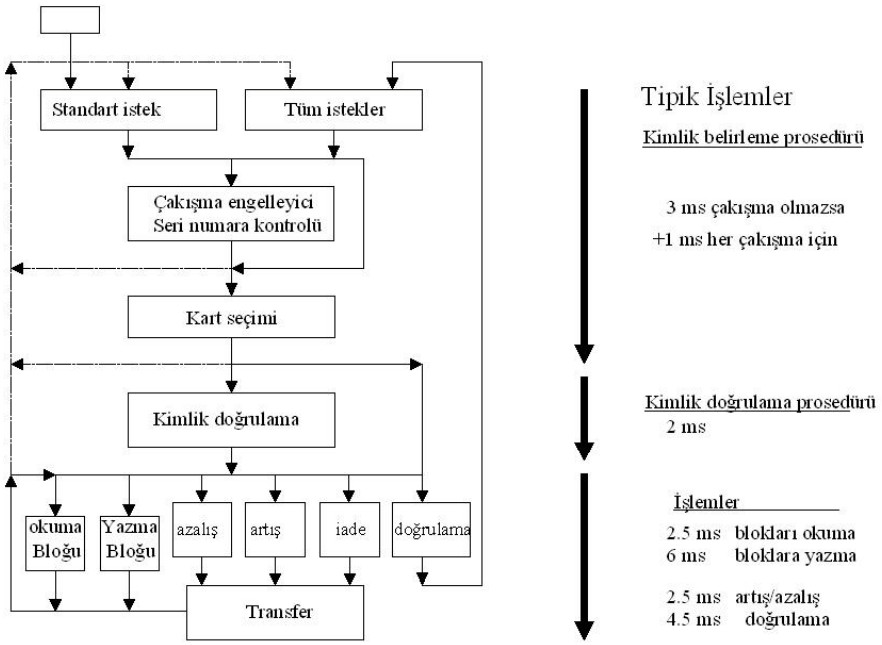
Şekil 3.6. MIFARE Classic Tag'ın blok şeması

3.3. MIFARE'e Saldırı

MIFARE Tag'ların güvenlikleri sürekli saldırılara maruz kalmaktadır. Bu saldırıların bir kısmı yazılımla yapılmaktayken bir kısmı da fiziksel ve elektroniksel saldırılardır.

3.3.1. Kaba kuvvet saldırı yöntemi

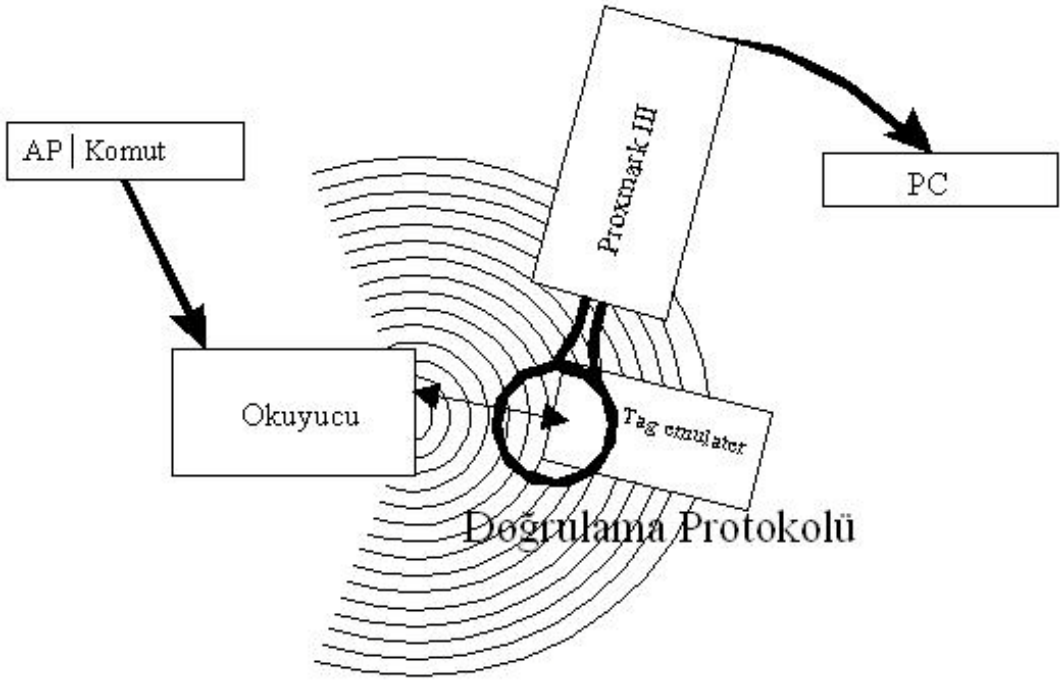
MIFARE Tag'a yapılacak en kolay atak yöntemi Tag'ın şifresinin bulunmasıdır. MIFARE Classic 48-bitlik bir şifreye sahiptir. Normal şartlarda tek tek şifrelerin denenmesi 2^{48} farklı olasılık vardır. Şifrelerin tek tek denemesine kaba kuvvet saldırısı denmektedir. RSA laboratories [29] tarafından geliştirilen RC5 algoritmasının 56bitlik şifresi 250 günde ve 64-bitlik şifresi 1757 günde kaba kuvvet saldırı yöntemiyle [30] kırılabildi. Fakat MIFARE Tagların her şifre denemesi Şekil 3.7.'de görüldüğü gibi yaklaşık 6ms sürdüğünden tek bir okuyucuyla toplam deneme yaklaşık 54297 yıl sürecektir. Bu nedenle MIFARE Tag'ların şifrelerin çözülmesi imkânsız gibi gözükmektedir. Fakat MIFARE rastgele sayı üretme hatasından dolayı birkaç saat ve hatta 1-2 saniye içinde de şifre çözülebilmektedir [37].



Şekil 3.7. MIFARE Classic Tag'ın doğrulama akışı ve süreleri

3.3.2. Okuyucuya saldırı yöntemi

Bu yöntemde Şekil 3.8.'de görüldüğü gibi Tag yerine bilgisayara bağlı Tag simule eden bir elektronik devre (Tag emülatör) kullanılarak hatalı doğrulama oluşturularak okuyucudan gelen cevaplar kayıt edilir. Okuyucu toplam olarak 2^{48} adet farklı cevap oluşturabilmesine karşılık bundan çok daha az kayıt sağlanmasıyla şifrenin çözülmesi sağlanabilmektedir [11]. Yapılan çalışmada en fazla 2^{36} adet kayıtla bunun kolayca sağlanabildiği görülmüştür. Fakat tüm 48-bit şifrenin hepsini bu yöntem yerine bir önceki yöntemin ortak kullanılmasıyla çok daha hızlı 48-bitlik şifre elde edilebilmektedir.

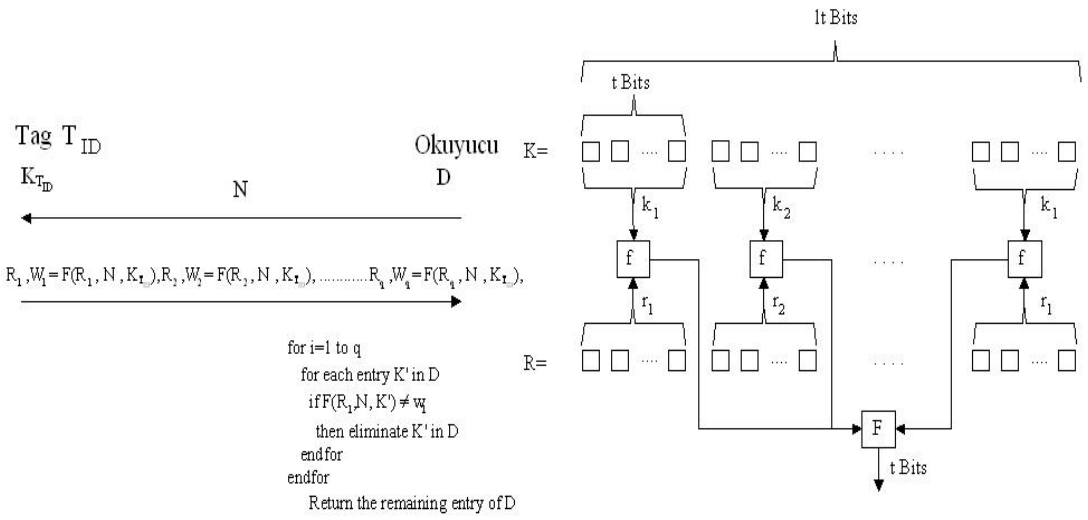


Şekil 3.8. Okuyucuya saldırı yöntemi

3.3.3. Cebirsel saldırı yöntemi

Bu yöntem adından da anlaşıldığı üzere cebirsel bir saldırı yöntemidir. Bu yöntemde MIFARE Tag ile doğrudan bir etkileşime gerek kalmamaktadır. Sadece okuyucu ile Tag arasında en az 50bitlik şifreli haberleşme verisine ihtiyaç vardır. Bundan sonra büyük bir polinom denklem seti ile şifre, ortalama 200 saniyede normal bir

bilgisayarla çözülebilmektedir [13]. Bu denklem çözümü "Satisfiability problem of propositional logic" veya kısa adıyla SAT algoritmasıyla çözülmektedir. Bu algoritmanın temeli, probleme ait tanımlanmış değişkenleri teker teker eski elemanlara dönüştürerek kaldırmasına dayanmaktadır. Bu işlem, problemin iyi tanımlanmasına ve bilgisayarın işlem hızına bağlıdır. Şekil 3.9.'da herhangi bir RFID sistemin doğrulama işleminin matematiksel ifadesi görülmektedir [28]. Bu ifade SAT algoritmasıyla çözülerek Tag'ın şifresi bulunabilmektedir. Bu yöntemle tüm tanımsız sistemlerin belli sayıda giriş ve/veya çıkış bilgisinin bilinmesi çözüm için yeterlidir.



Şekil 3.9. RFID sistemin doğrulama işlemin matematiksel ifadeleri [40]

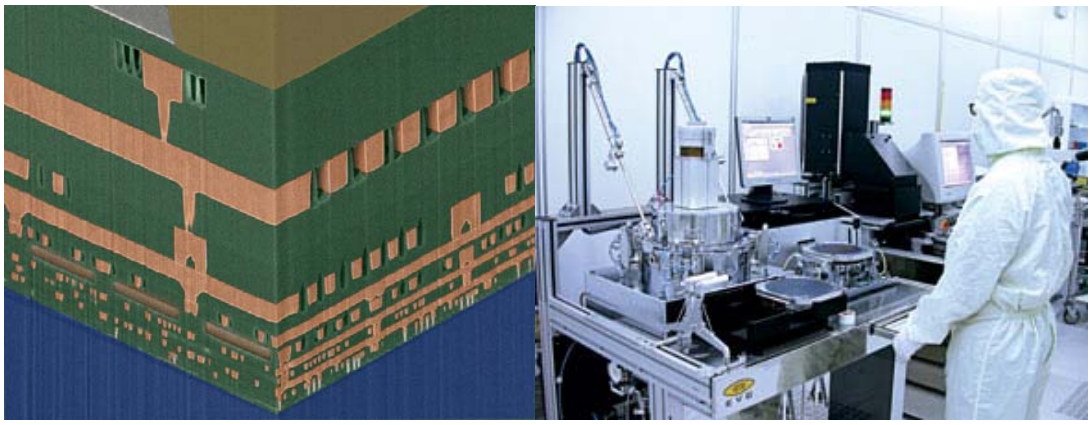
3.3.4. Kandırma saldırı yöntemi

Bu yöntemde okuyucuya sahte Tag'lar gerçekmiş gibi algılandırılır. Bu yöntem nöbetleşe saldırı (relay attack), ortadaki adam saldırısı (Man-in-the-Middle Attack) ve Klonlama (cloning Attack) yöntemleri için de geçerlidir. Her üç yöntem yaklaşık olarak aynı mantıkta çalışmaktadır. Sadece okuyucuya sahte Tag'ı doğruymuş gibi algılanmasını sağlayan uygulama farklıdır. Örneğin nöbetleşe saldırı yönteminde okuyucu ile Tag arasındaki tüm haberleşme kayıt edilir. Kayıt edilen bu haberleşme verileri okuyucuya sanki Tag varmış gibi tekrar tekrar aktarılarak kullanılır. Böylece kayıt edilen veri bir toplu taşıma aracına ait Tag ise sınırsız bir bilet elde edilmiş

olunur veya giriş/çıkış sistemine ait ise kayıt edilen kişi olarak giriş/çıkış sağlanmış olunur [27]. Bu yöntemde en önemli kriter, okuyucuya uygun cevabın tam zamanında verilmesidir.

3.3.5. Ters Mühendislik yöntemi

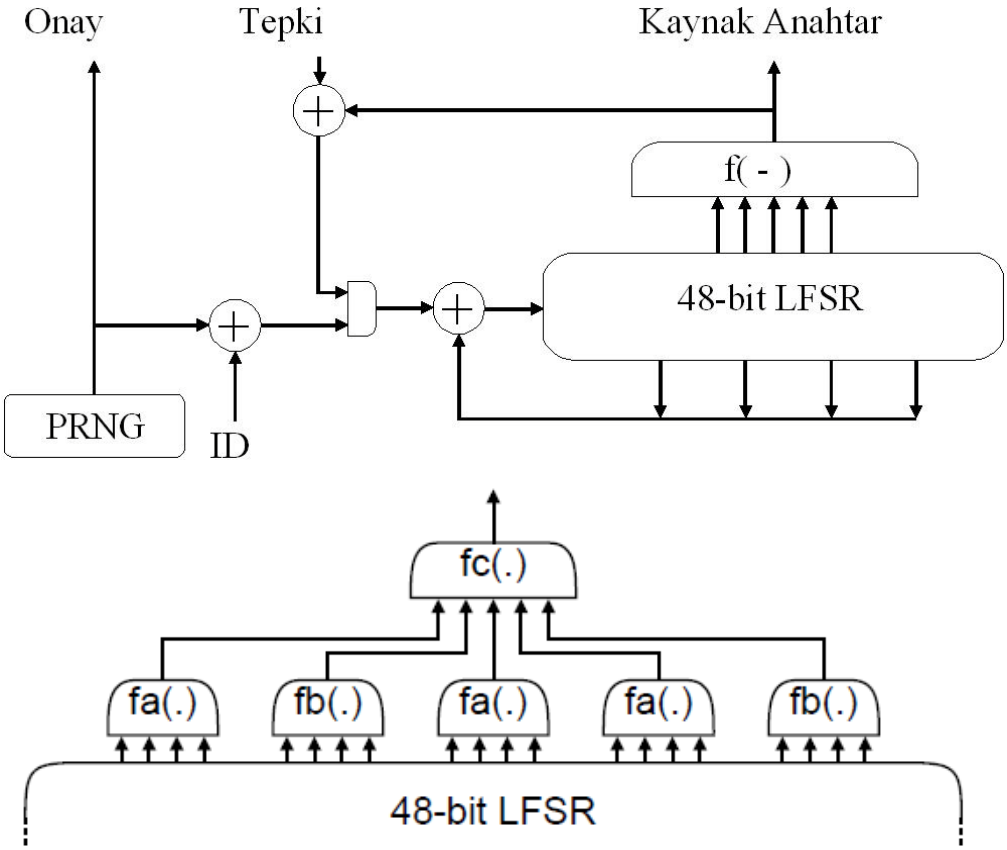
Bu yöntemde entegrenin katmansal yüzeyi aşama aşma kazınarak ve fotoğrafı çekilerek lojik yapının yeniden elde edilmesine dayanmaktadır. Bu yöntem kullanılarak MIFARE Tag'ların gizli olan yapısı tamamen deşifre edilerek 2007 sonunda yayınlanmıştır [12] Bu çalışmada Tag'ın entegresi 0.04µm zımpara ile zımparalanarak 500 kat büyüten mikroskopla fotoğrafları çekilmiş ve her zımparalama sonucunda farklı açı ve yönlerde en az 20 adet fotoğraf çekilerek panorama photo stitcher [11] programı yardımıyla birleştirilerek incelenmiştir. Bu incelemenin sonucunda MIFARE Tag'ların altı katmanlı olduğu ve yaklaşık 10.000 adet ve 70 farklı lojik kapıdan oluştuğu görülmüştür (Şekil 3.10.).



Şekil 3.10. MIFARE Tag'ların entegresi zımparalanarak lojik kapı yapısı elde edilmesi [41]

Bunun sonucunda kriptolama işlemini yapan Crypto-1 adındaki kriptolamanın sadece lineer kaydırmalı (lineer feedback shift register-LFSR) olduğu görülmüştür. Diyagram Şekil 3.11.'de görülmektedir [26]. Bu şifreleme yöntemi, NXP şirketinin HiTag2'de kullanılan şifreleme algoritmasına çok benzer olduğundan ve bunun

çalışma yöntemi bilindiğinden buna bağlı analizler çok daha hızlı sonuçlandırılarak sistem başarılı bir şekilde deşifre edilmiştir. Şekil 3.11.'de kriptolama işlemindeki en büyük hata gösterilmiştir. Rastgele sayı üretecinin (Pseudo Random Number Generator-PRNG Hash fonksiyonu) doğru çalışmamasından dolayı çok zayıf bir kriptolama yapılmaktadır.



Şekil 3.11. MIFARE Classic Tag'ların şifreleme blok diyagramı

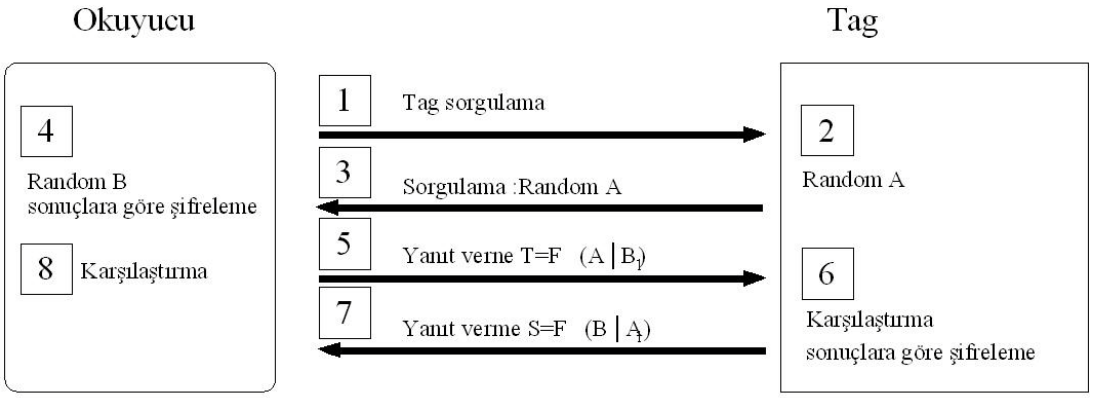
MIFARE Tag'ların Meydan okuma-Yanıt verme doğrulaması yöntemiyle çalıştığını ve bu yöntemde rastgele sayıların üretilmesiyle yüksek düzeyde güvenlik sağlanıldığı iddia edilir. Fakat gerçekte ise PRNG'nin rastgele sayı oluşturmasının Tag'ın ne kadar süre okuyucudan enerji almasına bağlı olarak "rastgele" bir sayı ürettiği tespit edilmiştir. Bunun anlamı, Tag'ın okuyucudan aldığı ilk enerjiden sonra herhangi bir t anında üreteceği PRNG sayısı daima sabittir. Böylece, Meydan okuma-Yanıt verme doğrulaması yöntemi bu hatadan dolayı Hash-Lock doğrulama gibi çalışmaktadır.

BÖLÜM 4. MİFARE GÜVENLİK AÇIĞININ İYİLEŞTİRİLMESİ

4.1. Mifare güvenlik açıkları

Bir önceki bölümde MİFARE Classic Tag'ların güvenli olmadığını ve saniyeler içinde çözülebileceği anlatılmıştı. Mifare Tag'ların dünya genelinde 1 milyardan [27] fazla satıldığı ve bu tercihte güvenlik ve ekonomikliğin öne çıktığı göz önüne alınırsa yeni bir güvenlik sisteminin oluşturulması (örneğin Mifare Plus'ın kullanılması gibi) sürecine kadar mevcut sistemin güvenli hale getirilmesi için ilave tedbirlerin alınması kaçınılmazdır. Fakat bu güvenlik açığın kapatılması sırasında Tag üzerinde ve haberleşmesinde herhangi bir değişiklik yapılması, mevcut Tag'a yapısal müdahaleyi gerektirdiğinden dolayı, mümkün olmadığı açıktır. Bu nedenle güvenlik açığının okuyucuda ve doğrulamadan sonra yapılması gerekmektedir. Tag'ın içindeki bilgilerini yetkili okuyucuya açması üç ana aşamadan sonra olmaktadır. Birinci adım, Tag'ın kendisini tanıtmasıdır. İkinci adım, doğrulamanın sağlanmasıdır. Son adım ise belli aralıklarla authentication'nun kesilmediğine dair bilgilendirme iletişiminin sağlanmasıdır. Tag'ın kendisini tanıtmasının en basit yolu sadece kendisinin sahip olduğu seri numarasını göndermektir. Doğrulamanın sağlanması ise mevcut Tag ile onunla haberleşen okuyucunun birbirlerini onaylanmasıyla mümkündür Bu durum Şekil 4.1.'de gösterilmiştir. Doğrulama işlemi sırasında onaylaşma kriptolanarak gönderilen şifrelerin aynı olmasıyla mümkündür.

Burada da görüldüğü gibi Tag'ların yüksek güvenlikli olabilmesi için haberleşmenin olabildiğince güvenli olması gerekir. Fakat haberleşmenin yüksek güvenli olması Tag'ın maliyetini artırmaktadır. Bu nedenle Tag'ların haberleşmesinin yüksek güvenlikli ve düşük maliyetli olması gerekir. Bu tezde önerilen sistem bunu sağlamaktadır. Çünkü önerilen sistemde Tag'ların ve haberleşmenin çalışmasında herhangi bir değişiklik yapılmamıştır.



Şekil 4.1. Meydan okuma-Yanıt verme doğrulaması

Mifare Tag'ların güvenlik açıkları iki başlık altında toplanabilir.

I) Tag içeriğinin değiştirilmesi

II) Tag'ı kopyalamak

Tag içeriğinin değiştirilmesini engellemek için bazı önlemlerden bahsedilebiliyorsa da Tag kopyalama için bir güvenlik mevcut değildir. Bu çalışmada özellikle Tag kopyalama sorunu üzerinde duruldu. Dolaylı yoldan bir güvenlik yöntemi geliştirildi. Bu yöntemin yazılım akış diyagramı Ek B'de gösterilmiştir. Geliştirilen bu güvenlik yöntemi 15 ay süreyle SAÜ'de üç farklı uygulama alanında (manuel para yükleme, otomatik para yükleme ve içerik izleme, turnike kontrolü) başarıyla uygulandı. SAÜ'deki uygulama kapalı bir çevrim uygulaması olduğundan (tüm sistem online internet üzerinden takip edilebilmektedir) uygulaması ve takibi kolaylaştırmaktadır. Bu nedenle aynı uygulamayı Adapazarı Büyükşehir Belediyesinin mevcut Mifare bilet sisteminde ilave sistem olarak 10 gün süreyle uygulanabilir olduğu başarıyla görüldü.

Tag içeriğinin değiştirilmesindeki ana mantık mevcut bir Tag'ın şifresi kırılarak içeriğinin yedeklenmesi ve gerektiğinde bu yedeklenen içeriğin Tag'a geri yüklenmesidir. Böylece yedeklenen Tag bir otobüs biletine ait ise Tag'a para yüklendikten sonra yedeğinin alınmasıyla sonsuza kadar bu bilet bir daha para yüklenmeden kopyası sürekli Tag'a yüklenerek kullanılması anlamına gelmektedir.

Bunun yanında Tag'daki bilgiler kriptolu değilse istenen para değeri hiçbir yükleme yapılmadan da kolayca istenen değerle değiştirilebilir. Mifare Tag'ların özellikle parasal işlemler için tasarlandığı düşünülürse kripto kullanmadan artma, azaltma ve geri alma işlemleri yapılamaz. Bu nedenle Tag'ın şifresinin kırılmasıyla istenen değer Tag'a yüklenebilir. Mifare Tag'ların bu özellikleri kullanılmadığı takdirde Tag'lardaki bilgi kriptolanabilir. Ancak bu durumda parasal işlemler için artma, azaltma ve geri alma işlemleri Tag'da değil okuyucuda yapılmalıdır. Bu da Tag'ın daha fazla okuma işlemine tabi tutulması demektir. Mifare Tag'ların parasal işlemleri için sahip olduğu bu özelliklerin kullanılabilmesi için ilave güvenlik işlemleri geliştirilmiştir.

Bu güvenlik işlemlerin ana mantığı Tag'ların sahip olduğu Key A ve Key B'nin farklı okuyucularda olması ve şifrenin Tag'dan değil okuyucuların Tag'larla haberleşmesinin dinlenmesiyle elde edildiği varsayılarak geliştirilmiştir. Böylece Key A'ya sahip okuyucular sadece Tag'lardaki bakiyeyi azaltma yetkisine sahip olduğundan şifre kırmaya karşı yüksek güvenlikli korumaya gerek kalmamıştır. Toplu taşıma araçların içinde veya yemekhane turnike sisteminde olduğu gibi.

Fakat Key B'ye sahip okuyucular Tag'lardaki bakiyeyi istediği gibi değiştirme yetkisine sahip olduğundan bu okuyucu sisteminde şifre kırmaya karşı yüksek güvenlik gerekmektedir. Örneğin: Tag'lara para yükleme noktalarındaki haberleşme yüksek güvenlikli olmalıdır. Fakat Mifare Tag'ların tasarım hatasından dolayı azaltma yetkisi olan şifrenin bilinmesiyle de Tag bakiyesi arttırılabilmektedir [24]. Bu tasarım hatası ise Tag okuyucusuyla haberleşirken işlem bitmeden Tag'ın uzaklaştırılmasıyla eğer bakiye azaltma esnasında bu yapılırsa rastgele bir değer bakiyede oluşabilmektedir. Böylece azaltma yetkisine sahip şifre bilinmesiyle Tag'ın okuyucuyla haberleşmeye geçtikten ne kadar sonra uzaklaştırıldığında istenen bakiyenin oluştuğu deneme yanılma yöntemiyle elde edilebilir.

Tag içeriğinin değiştirilmesi ile Tag'ın kopyalanması genel olarak aynı işlemlerdir. Aralarındaki tek fark Tag'a ait her iki şifrenin elde edilmesi sonucunda mevcut olan

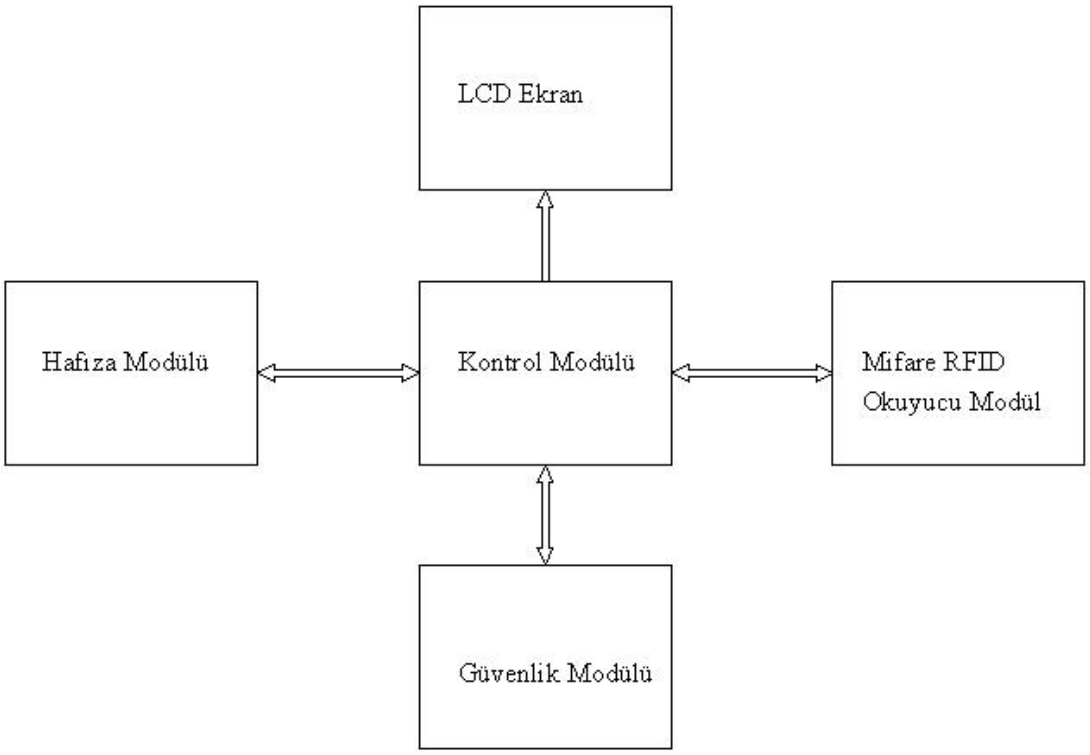
Tag'ın birebir kopyasının yeni boş bir Tag'a kopyalanmasıdır [25]. Böylece orijinal Tag hiç kullanılmadan sadece içeriğinin boş Tag'a kopyalanması sonucunda sonsuza kadar Tag kullanılabilir olmaktadır. Bunun önlenebilmesi için harici bir sistemin Tag'ı denetlemesi gerekmektedir. Burada önerilen sistem online (SAÜ yemekhane otomasyon sistemi) veya yarıonline (Sakarya Büyükşehir Belediyesi toplu taşıma araçları) sistemdir.

Kopyalanan Tag'ların orijinallerinden bazı farklılıkları vardır. Bunlar sıfırıncı blokta bulunan seri numarası (UID) ve okuyucuyla haberleşme esnasındaki Tag'ın okuyucuya cevap verme zamanı ve elektrik alan şiddetinin farklı olmasıdır. İlk farklılık Tag emule eden bir cihaz [4] kullanıldığında anlaşılamaz. Ayrıca Çin taklidi Mifare Tag'lar üretildiği bilinmektedir [18, 24]. Bu Tag'lara istenen UID değerleri atanabilmektedir. Bu durumda da Tag ayırt edilemez olabilmektedir. Diğer durumlar ise okuyucunun okuma işlemini yapan entegresine bağlıdır (firmware). Standart Mifare okuyucu entegrelerin hiçbirinde Tag'ın cevap verme zamanı ve/veya elektrik alan şiddetini ölçen bir özellik mevcut değildir. Bu özellikler Mifare entegresine eklenebilir ancak bunun için Mifare okuyucularının tasarımları değiştirilmelidir. Bu ise maliyet artışı demektir. Ayrıca bu son durumlar Tag'ların üretim kalitesine çok bağlı olacağından orijinal Tag'larında bu bariyere takılma ihtimali vardır. Bu nedenle kopya Tag'ların %100 ayırt edilebilmeleri mümkün değildir. Ancak burada önerilen online ve yarıonline sistemle kopyalanmış Tag'lar rahatlıkla tespit edilebilmektedir.

4.2. Geliştirilen güvenli Mifare sistemi

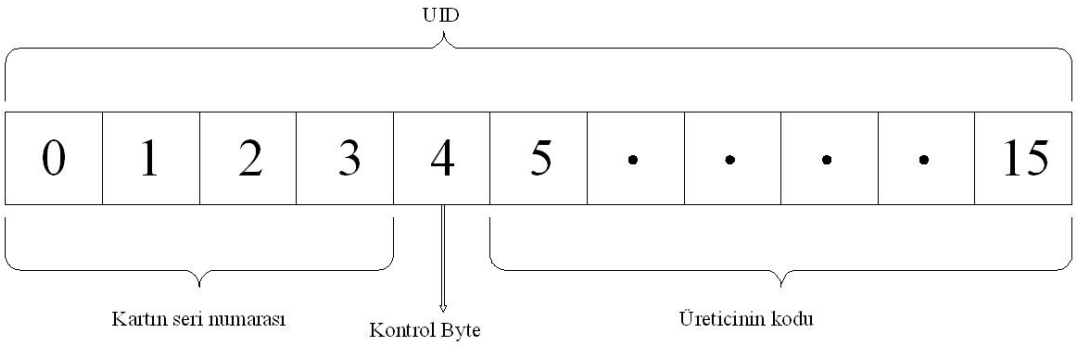
Mifare Tag'ların bu güvensizliğine rağmen bu güvenlik açıklıklarının kullanılarak içeriklerinin değiştirilmesi ve kopyalanması herkes tarafından yapılabilecek kolay bir şey değildir. Bu işlemlerin yapılabilmesi öncelikle RFID ve haberleşme sisteminde bilgili olunması, pratik uygulama ve parasal desteğe ihtiyaç vardır. Fakat Mifare Tag'ların içerik değiştirme veya kopyalama işinin herkes tarafından kolayca yapılacak düzeye inmesinin ne zaman olacağı tahmin edilemeyeceğinden güvenli ve kullanılabilir bir sistem geliştirildi. Burada geliştirilen sistem 2000–2008 yılları

arasında geliştirilen ve uluslararası fuarlarda sergilenen sistemin [32]. 2007 yılı sonuna kadarki Tag'ların güvenliği kesinlikle yetersiz kaldığından sistem de güvensiz olmuştur. Mifare Tag'ların güvenliğinin üstünde ektra bir güvenlik sağlamanın dezavantajı daha uzun bir okuma süresidir. Bunun anlamı Tag'ın okuyucu da daha uzun süreli tutulmasıdır. Normal şartlar altında bir Tag'ın okunması ve bakiyesinin değiştirilmesi 85 ms sürmektedir. Geliştirilecek harici güvenliğin hem daha uzun süreli bir okuma yapması hem de harici güvenlik için kendi hesaplama zamanını da eklenmesiyle süre daha da artacaktır. Bu sebepten harici bir güvenlik sisteminin geliştirilmesi için öncelikle uygulamaya bağlı bir Tag'ın Okuyucu üzerinde en az ne kadar durması gerektiği araştırıldı. Sakarya Üniversitesindeki Personel ve öğrenci yemekhane turnikelerinde yapılan çalışmada, pratikte Tag okutanların sırada bekleme zamanlarına bağlı olarak bu sürenin en az 1,5-2 sn ve en çok 105 sn olduğunu tespit etmiştir. Bu sürelerin bir kişinin Tag'ını turnikedeki okuyucuya koyması ile geçtikten sonra arkadaki diğer kişinin Tag'ını turnikeye koyması arasında geçen süre olarak belirlenmiştir. Yemekhanede sıra olmadığı zaman elde edilmiş süre en az ve yemekhane sırasında yoğunluk olduğu zaman en çok süredir. Aynı ölçüm Sakarya Büyükşehir Belediyesinin toplu ulaşım araçlarında denendiğinde en az 1-1,4 sn ve en çok 15 sn olarak belirlenmiştir. Toplu taşıma araçlarında ilk otobüse binen kişi ile ikincisinin arasındaki süre buradaki en az süreyi ifade etmektedir. Fakat bu en az süre sadece otobüse giren ilk iki kişi için geçerlidir. Bu yüzden bu değer optimum değer değildir. İkinci-üçüncü kişilerde bu süre 1,3–1,6 saniye iken üçüncü-dördüncü kişilerde bu süre 1,5–2,1 saniye olmaktadır. Sakarya Üniversitesinde geliştirilen Mifare Güvenlik Sistemi Şekil 4.2.'de görülmektedir.



Şekil 4.2. Mifare Güvenlik Sistemi Blok Şeması

Kontrol Modülü Microchip'in PIC 18F452 mikroişlemcisi ile tasarlanmıştır. Mifare RFID okuyucu modül Mifare'nin standart entegresini içermektedir. Hafıza modülü micro SD 1 GByte (yemekhane sistemi) veya 8 GByte (toplu taşıma sistemi) kullanıldı. Güvenlik modülü için PIC16F84 mikroişlemcisi kullanıldı. Mifare Tag'ların Blok 0 dışındaki tüm bloklar belli bir algoritmaya bağlı Dinamik olarak şifrelenmiştir. Daha önce geliştirilen bu algoritma [10] sayesinde her Mifare Tag'ın kendine özgü 15 adet şifresi vardır. Böylece herhangi bir Tag'ın şifresi bulunsa dahi diğer Tag'ların bilgisine ulaşılamaz. Geliştirilen bu algoritma sayesinde her Mifare Tag'ın kendine özgü her bloğun farklı olmak üzere 16 adet şifresi vardır. Böylece herhangi bir Tag'ın şifresi bulunsa dahi diğer Tag'ların bilgisine ulaşılamaz. Geliştirmiş algoritmanın ana mantığı bir sonraki alt başlıkta açıklanacaktır. Mikroişlemciye ait akış diyagramı Ek A'da verilmiştir. Akış diyagramına göre sistemin çalışması özetle şu şekildedir: Kontrol ünitesi yaklaşık 350 msn'de bir okuyucu alanında bir Tag'ın olup olmadığını sorgular. Okuma alanına bir Tag konulduğunda buna bağlanır ve bu bağlanmada Tag'a Şekil 4.3.'deki gibi UID gönderir.



Şekil 4.3. UID'ye ait 16-Byte'lik veri şekli

UID, mikro SD'deki yasaklı tablo ile karşılaştırılır ki eğer bu UID yasaklı ise sistem işlemi durdurur. Yasaklı değilse geliştirilen algoritma ve UID ile Blok 3'e ait şifre hesaplanır. Bu şifre her Tag için ayrı olduğundan buna Dinamik Şifre denir. Hesaplanan bu şifre ile Blok 3 Sektör 0 okunarak 16 Byte AES'le şifrelenmiş bilgi güvenlik modülüne iletilir. AES'in açık kodu Ek-B'de verilmiştir. Güvenlik modülü bunu çözene kadar Blok 3, Sektör 1 ve Sektör 2'de okunur. Sektör 0 bilgisi doğru olarak açıldığında buradaki tarih bilgisi ile şu andaki tarih bilgisi karşılaştırılır. Bu tarihler farklı ise en son okuma bir önceki ay yapılmış demektir. Sektör 1'de tek aylar Sektör 2'de çift aylara ait bakiye işlemleri tutulduğundan bir önceki aya ait bilgi diğer sektöre aktarılmalıdır. Örnek olarak en son Tag'a kayıt Mart ayında yapılmış ve şu anki okuma Nisan ayında olduğu düşünülürse, Mart'a ait bilgi tek ay olduğundan Sektör 1'de saklanmaktaydı ve bu bilgi Sektör 2'ye aktarılmalıdır. Böylece şu anki aya bağlı olarak Sektör 1 veya Sektör 2 okunur. Buradaki bilgi güvenlik modülüne gönderilerek AES şifrelemesini çözerek bilgiyi geri gönderir. Buradaki bakiye bilgisi ile Sektör 0'daki bakiye bilgisi karşılaştırılır. Eşitse, geçiş ücretinden büyük veya eşit olup olmadığına bakılır. Bu şartı da sağlıyorsa geçişe izin verilir. İzin verilmezden hemen önce yeni bakiye ve şu anki tarih bilgisi AES ile şifrelenerek Sektör 0'a yazılır. Yazılma işleminden 50 msn sonra yazılan bilgi tekrar okunarak doğru yazım yapılıp yapılmadığı kontrol edilir.

Mevcut denemelerde görüldü ki birinci yazmadaki başarı %84 ve ikinci yazmadaki başarı %100 olarak gerçekleşmiştir. Buradaki başarısız yazmaların sistemden kaynaklanmadığı kullanıcıdan kaynaklandığı bilinmelidir. Aynı şekilde şu anki aya bağlı bakiye bilgisi de AES ile şifrelenerek Sektör 1 veya Sektör 2'ye yazılır ve yazımın doğruluğu da yine yukarıda anlatıldığı gibi yapılır.

Mikroişlemcinin mikro SD'ye kaydettiği bilgiler sunucuya gönderilerek orada işlenir. Sunucuda geliştirdiğimiz bir yazılım ile bir Tag'ın kopyalandığını veya içeriğinin değiştirildiğini kontrol etmekteyiz. AES'den dolayı içerik değiştirilemeyeceğinden sadece Tag'ın kopyalanıp kopyalanmadığına bakmaktayız. Böyle bir sonuçla karşılaşıldığında Tag'ın UID'si mikro SD'nin yasaklı Tag tablosuna kaydedilir ve kullanılması önlenir. Geliştirilen bu sistem bilgiyi AES ile şifrelediğinden Tag aynı Tag'a kopyalandığında Blok 3'e ait bilgi sistemde iki defa gözükeceğinden kopyalandığı anlaşılmış olacaktır. Başka bir Tag'a kopyalandığında UID'si farklı olacağından mikroişlemcili sistem Dinamik Şifrelemeden dolayı bunu tespit ederek kullanımı engeller.

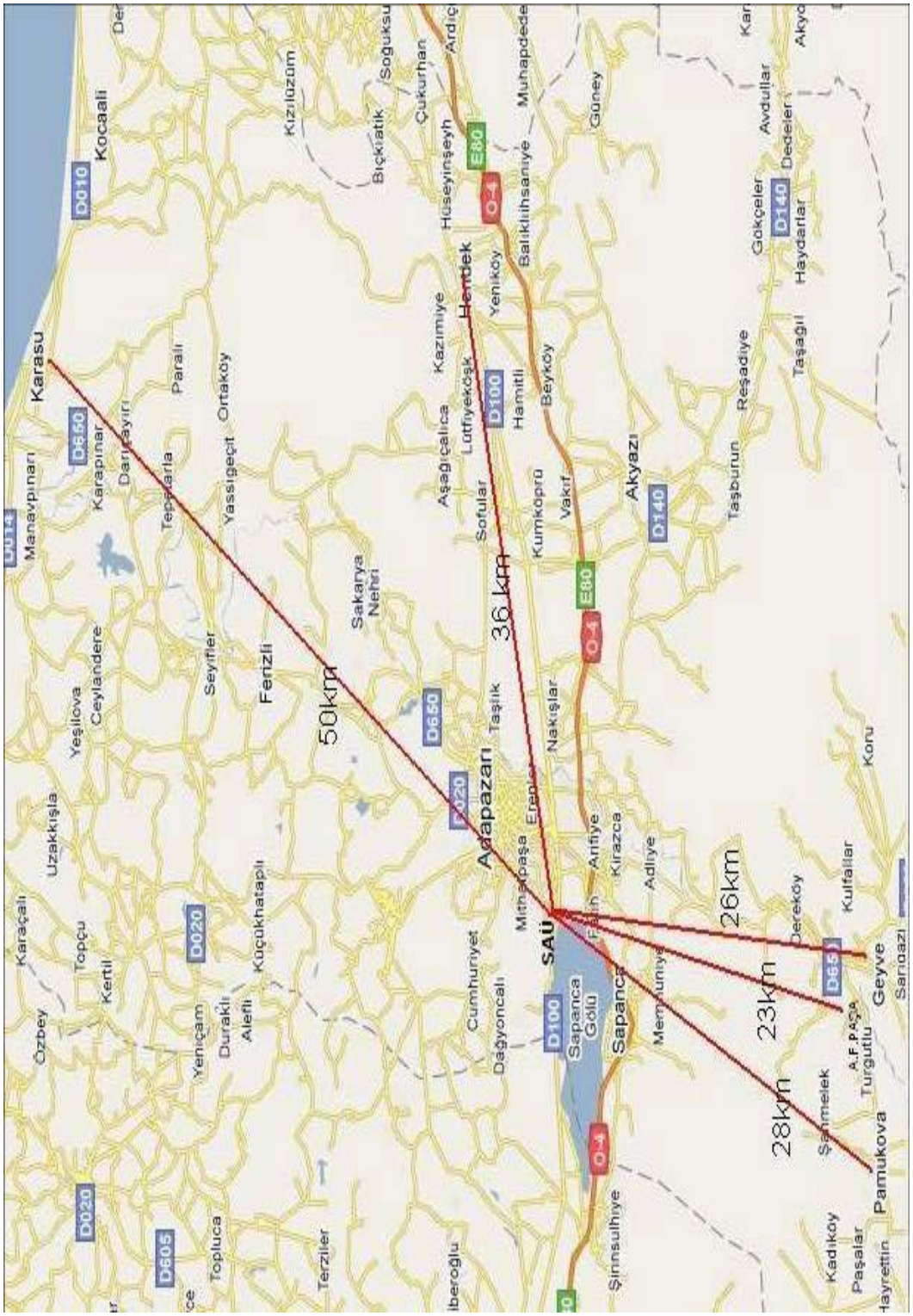
Geliştirilen bu sistem AES ile bilgiyi şifrelediğinden bilginin yetkisiz kişiler tarafından değiştirilmesi bugün için mümkün değildir. Tag'ın başka Tag'a kopyalanması mikroişlemci tarafından algılanarak engellenmektedir. Fakat Tag dolumdan hemen sonra Tag bilgisi yedeklenir ve aynı Tag'a geri yüklenirse mikroişlemci ile sunucunun bilgi alışverişinden sonra ancak anlaşılmaktadır. Bu süre ne kadar kısa olursa aynı Tag'a yapılan kopyalama işlemi o kadar kısa sürede anlaşılacaktır.

Sakarya Üniversitesinin yemekhane otomasyon sistemi online veri aktarımıyla çalıştığından aynı Tag'a yapılan kopyalama işlemi en fazla 15 sn'de bulunabilecek yapıdadır. Bu önerilen sistem Sakarya Üniversitesi yemekhanesinde 15 ay süreyle uygulandı. Ana sunucu ile okuma işlemini yapan istasyonların mesafesi Tablo 4.1 ve Şekil 4.4'de görülmektedir. Buna karşılık Sakarya Büyükşehir Belediyesinin toplu

taşıma araçlarındaki sistem yarı online çalıştığından veri aktarımı yapıldıktan en fazla 183 sn sonra bulunabilmektedir. Buradaki sürenin daha uzun olmasının nedeni veri aktarımının sadece tek bir durakta yapılıyor olmasındandır. Veri aktarımı birden fazla durakta yapıldığı takdirde sürenin yemekhane otomasyon sistemindekine yakın olacağı açıktır.

Tablo 4.1. Geliştirilen Yüksek güvenlikli RFID sistemin 6 ana bölgenin merkez kontrol sistemine uzaklıkları

NEREDEN	**NEREYE**	**MESAFE(KUŞUÇUŞU)**
KAMPÜS	SALON	6 km
KAMPÜS	KARASU	50 km
KAMPÜS	PAMUKOVA	28 km
KAMPÜS	GEYVE	26 km
KAMPÜS	HENDEK	36 km
KAMPÜS	ALİFUATPAŞA	23 km



Şekil 4.4. Geliştirilen yüksek güvenlikli Mifare sistemi (Sakarya ili ile ilçeleri arasında)

4.3. Dinamik şifreleme

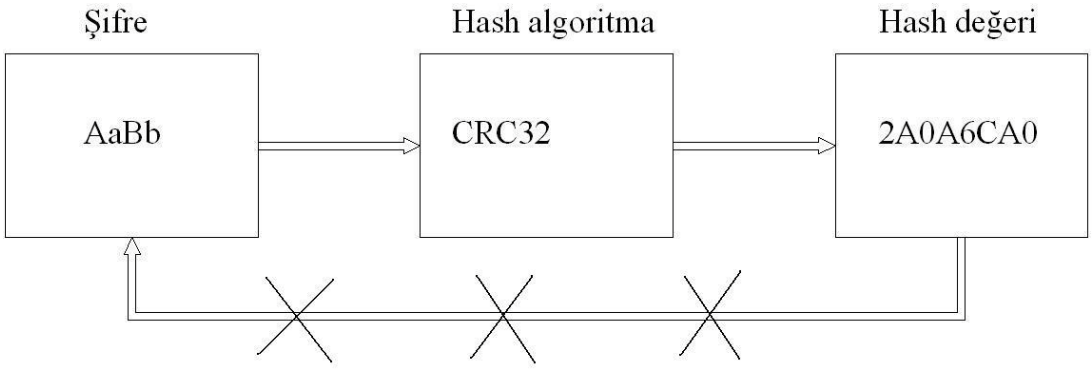
Geliştirilen dinamik şifreleme, ana mantığını Hash algoritmasından almıştır. Hash algoritması elektronik ortamda bulunan verinin bir hexadesimal karşılığını üretmektedir. Bunu bir imza gibi de düşünebiliriz. Başka bir ifadeyle elektronik ortamda bulunan verinin izinsiz değiştirilmesi durumunda Hash algoritması da değişmektedir. İmza bu şekilde taklit edilmiş olmaktadır. Gerçek hayatta bu durum şuna benzemektedir: Herhangi bir ürün alındığında henüz ambalajı ve güvenlik etiketi açılmamışsa bu ürünün hiç kullanılmadığını ve açılmadığını, içeriğinin değiştirilmediğini düşünürüz. Buradaki ürünün kapalı ambalajı veya güvenlik etiketi Hash algoritmasının hesapladığı hexadesimal sayıya karşılık gelmektedir. Hash algoritmalar giriş değerinin büyüklüğünden bağımsız daima sabit uzunlukta bir hexadesimal sayı üretmektedir. En yaygın üç adet hash algoritması; MD5, SHA4-512 ve CRC32 ait örnek Tablo 4.2' te görülmektedir.

Tablo 4.2. Hash algoritmalarına ait örnekler

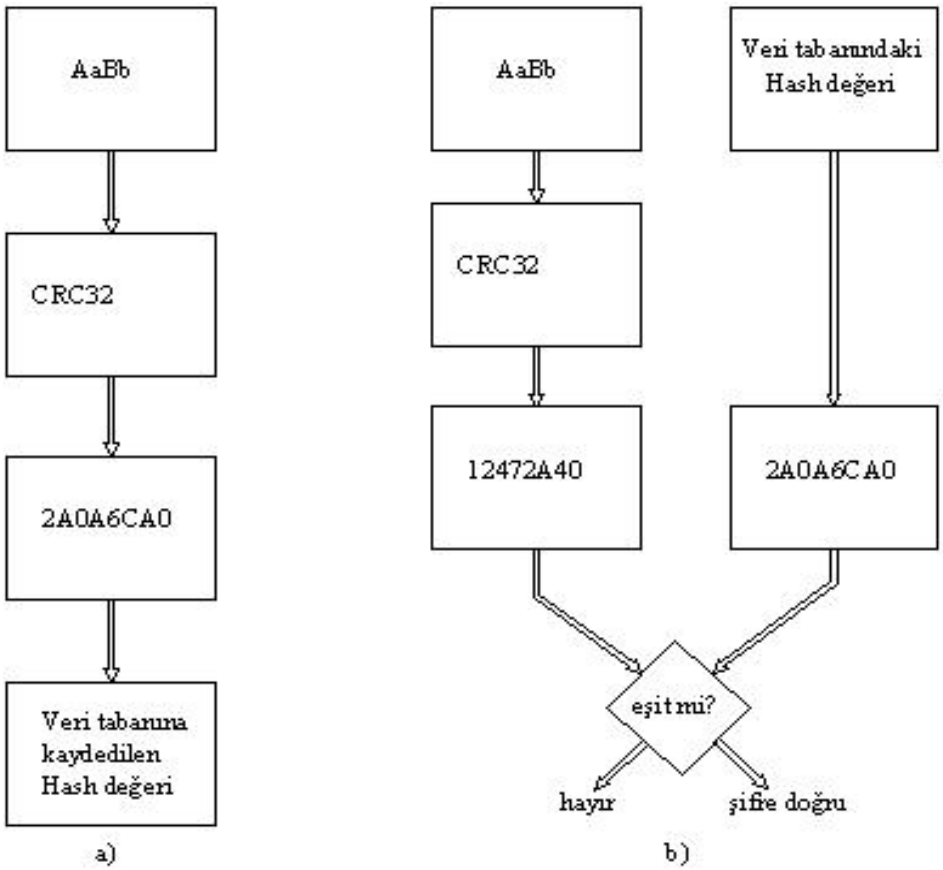
	Giriş	Hash sonucu
MD5	1	C4CA4238A0B923820DCC509A6F75849B
	Z	21C2E59531C8710156D34A3C30AC81D5
	Z	FBADE9E36A3F36D3D676C1B808451DD7
	1234567890*qwertyuıopğü asdfghjklşizxcvbnmöç	754E15511B00FB80ABAA922C0CED537C
SHA4-512	1	4DFF4EA340F0A823F15D3F4F01AB62EAE0E5DA579CCB851F8DB9 DFE84C58B2B37B89903A740E1EE172DA793A6E79D560E5F7F9BD 058A12A280433ED6FA46510A
	Z	3225DFF071CD0CCFF736B0B159CC722963310C008472BE814669 451A062C25C5F7654F079D3E0AE1CF2FDA1551A5A0B1F5E98838 3BE7D383D57F73D4012C4024
	Z	5AE625665F3E0BD0A065ED07A41989E4025B79D13930A2A8C57 D6B4325226707D956A082D1E91B4D96A793562DF98FD03C9DCF 743C9C7B4E3055D4F9F09BA015
	1234567890*qwertyuıopğü asdfghjklşizxcvbnmöç	02D7794CDED7554FC283AEFCEB6184BD147EE9F3B18256F316D4 66F94D3501F655ADD163765580E0AD351200C5C72799E54F4C53 CF8EA069D84A87659C44CDE7
CRC32	1	83DCEFB7
	Z	59BC5767
	Z	62D277AF
	1234567890*qwertyuıopğü asdfghjklşizxcvbnmöç	73C83B30

Hash algoritmaları Şekil 4.5.'te görüldüğü gibi tek yönlü çalışmaktadır. Bunun anlamı Hash değerini bilen kişi şifreye ulaşamaz. Bu nedenle web uygulamalarında

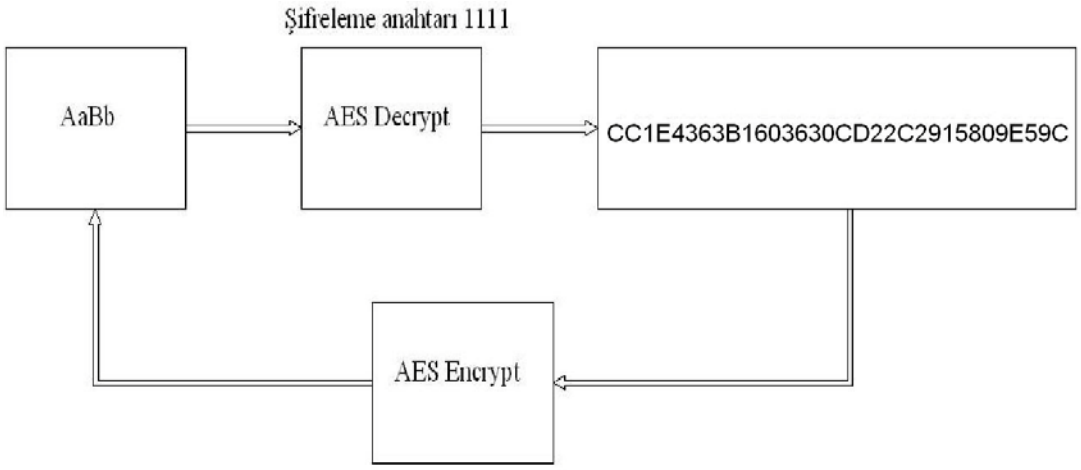
şifrelerin kendileri değil Şekil 4.6.a.'da tanımlandığı gibi hash değerleri saklanmaktadır. Böylece veri tabanına bakmakla yetkili olanlar sadece bu hash değerini görebilmektedir. Fakat bununla herhangi bir işlem yapamazlar. Web uygulamalarında sisteme şifreli giriş yapan kişinin girdiği şifrenin hash değeri hesaplanarak Şekil 4.6.b.'de gösterildiği gibi veri tabanındaki hash değeri ile karşılaştırılır, eşitse şifre doğru girilmiş olur. Ancak her güvenlik yöntemine karşı bir karşı çözüm üretilmeye çalışılmaktadır. Hash algoritmaları şifreleme algoritması değildir. Şifreleme algoritmaları Şekil 4.7.'de görüldüğü gibi çift yönlü çalışırlar. Hash değerleri tek yönlü ve sabit basamaklı olduğundan asıl şifreye ulaşılamasa da "Gökkuşağı Tablosu" diye adlandırılan tablolar yöntemiyle aynı hash değerine sahip şifreler bulunabilir. Bu Gökkuşağı Tablosunda milyonlarca şifre için hash değeri bulunduğundan aranan hash değerine bir tanesi denk gelebilir. Hatta Şekil 4.8.'de aynı hash değerine sahip farklı şifrelerin olabileceği gösterilmiştir. Çünkü hash değerinin basamak sayısından büyük veya küçük hesaplamalarda farklı şifrelere karşı aynı hash değeri çıkabilmektedir. MD5 için 1024 bitlik bir şifrenin hash değerinden yola çıkarak bir şifre üretilebilmektedir [38]. Fakat yine de hash algoritmalarında bu ihtimal düşük sayılmaktadır. Bu nedenle yaygın kullanımı devam etmektedir. Hash değerinden hiçbir zaman şifreye ulaşılmaz. Hash değerinin gökkuşağı tablosundan bulunamaması için şifre olabildiğince uzun ve özel karakterlerden oluşmalıdır.



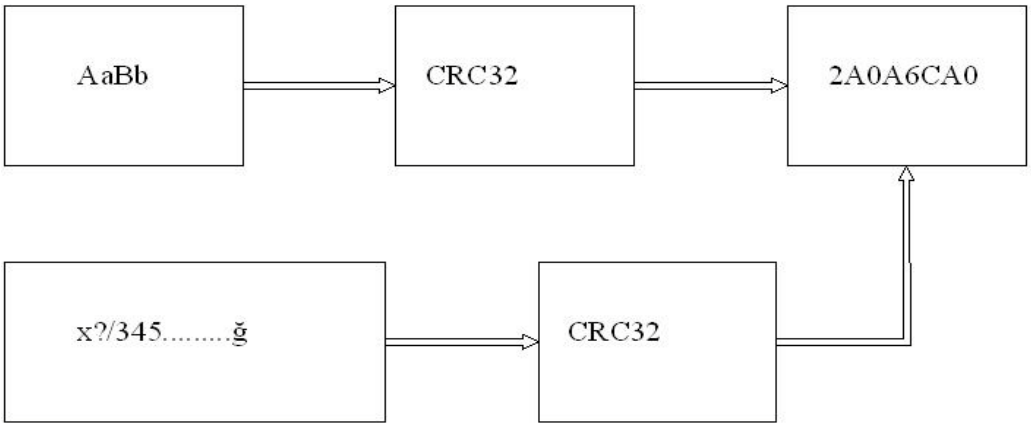
Şekil 4.5. Hash algoritmalar sadece tek yönlü çalışmaktadır



Şekil 4.6.a. Web uygulamalarında şifrenin veritabanına kayıt edilme yöntemi
b. Web uygulamalarında şifreyle besleme giriş yöntemi



Şekil 4.7. Şifreleme algoritmaları iki yönlü çalışır



Şekil 4.8. Hash algoritmalarda farklı şifreler aynı Hash değerine karşılık gelebilir

Geliştirilen algoritmanın ana mantığı buradaki hash algoritmalamaya dayanmaktadır. Aralarındaki fark, mikroişlemcilerde kolay, hızlı hesaplanabilir ve güvenli olmasıdır. Hash algoritmaların nasıl çalıştığına burada değinilmeyecektir. Fakat iki örnek üzerinde nasıl çalıştığı gösterilecektir.

Örnek 1:

Her hexadesimal sayıya karşılık bir sayı tablosu oluşturulur. Bu sayı tablosu uygulamadan uygulamaya göre değiştirilir (uygulamaya özgü). Bu hexadesimal sayılar Tablo 4.3.'deki karşılarındaki değerlerle değiştirilir.

Tablo 4.3. Hexadesimal sayıların başka bir değere çevrilmesi

Hexadesimal	Karşılık
0	101
1	119
2	140
3	
.	
.	
E	100
F	199

Hexadesimal değer : 1EF2

Hash hesabı için dönüşümü : 119 100 199 140

Sonrasında bu değerler üstel bir fonksiyonda hesaplanır. Bu fonksiyon da yine her uygulama için farklıdır.

$h = s[0] \cdot 19^{n-1} + s[1] \cdot 19^{n-2} + \ldots\ldots\ldots + s[n-1]$

$h = 119 \cdot 19^3 + 100 \cdot 19^2 + 199 \cdot 19 + 140$

$h = 854451$

Hash değeri sabit basamaklı olduğundan sonuç değerin basamak sayısı az ise sabit bir sayı ile çarpılır veya basamak eklenir. Burada 128 bitlik yani 16 byte (8 basamaklı) olduğunu varsayarsak bunu da sabit bir sayıyla çarparsak:

h = 854451 x 3 = 2563353 x 3 = 7690059 x 3 = 23070177

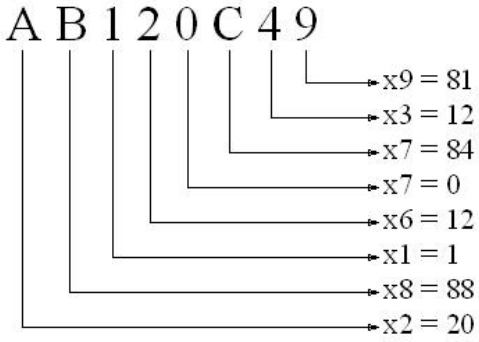
sonucuna ulaşılmış olur.

Örnek 2:

Giriş hexadesimal sayı sabit basamaklı ise, örneğin 8 basamaklı (32 bit) olduğunu varsayalım. Her basamağı farklı sabit bir çarpanla çarpılır. Bu çarpanlar yine uygulamaya özgüdür.

1. basamak 9 ile	4. basamak 7 ile	7.basamak 8 ile
2. basamak 3 ile	5. basamak 6 ile	8.basamak 2 ile
3. basamak 7 ile	6. basamak 1 ile	

Hexadesimal sayımız: AB120C49 olsun.



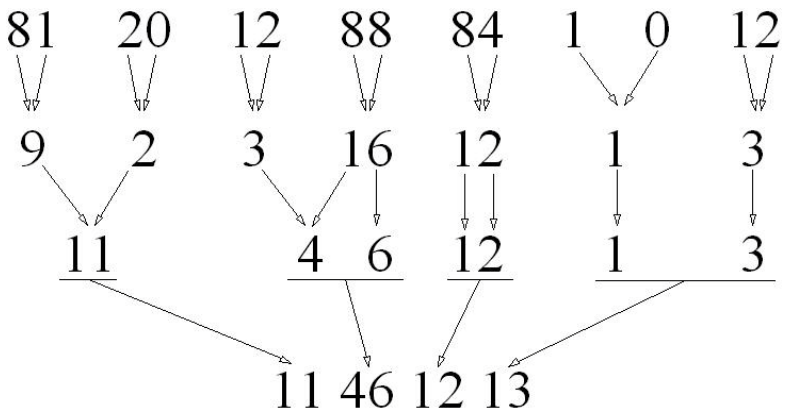
Elde edilen her basamağa ait sayılar belli bir mantığa göre sıralanır. Örneğin üç farklı sıralama mantığı şu şekilde olabilir:

a) 81 20 12 88 84 1 0 12

b) 12 0 1 84 88 12 20 81

c) 81 84 12 88 20 1 0 12

Elde edilen bu sayı sabit basamağa indirgemek için kendi aralarında belli bir mantığa göre toplanır. Bu toplama işlemi yine uygulamaya özgüdür. a) için :



Böylece gerekli hash değeri (11461213) bulunmuş olur.

Sonuç olarak Hash algoritmalar sadece tek yönlü çalışmaktadır. Sonuç değerinden asıl değere ulaşılamaz. Her uygulama için çok farklı sonuçlar veren Hash algoritmalar oluşturulabilir. Fakat MD5 gibi algoritmalar dünya genelinde ve

özellikle yazılım sektöründe orijinal ile sahte yazılımların ayırt edilmesinde kullanılan en yaygın Hash algoritmalarıdır. Bu algoritmalarla sadece orijinal ile sahte yazılımların birbirinden ayırt edilebilmesi için kullanılabilir yoksa sistem güvenliği sağlanamaz. Bu nedenle her uygulama için kendine özgü güvenli Hash algoritmalar geliştirilmektedir.

SONUÇLAR ve ÖNERİLER

Mifare Tag'ların kendi sınıfındaki Tag'lara göre 11 yıldır çok yaygın olarak kullanılıyor olması sistemlerde telafisi çok zor güvenlik açıkları vermesine neden olmuştur. Mifare Tag ve okuyucularının bu güvenlik açığı giderilmeden uygulamalarda kullanılması uzun vadede imkânsız hale gelmiştir [39]. Mifare Classic Tag yerine Mifare AES Tag'ın kullanılması zaman ve maliyet açısında dezavantajlıdır. Bu yüzden mevcut sisteme düşük maliyetli ve Tag'lara müdahale etmeksizin bu dezavantajı giderecek bir sisteme ihtiyaç duyulmaktadır. Bu tezde sunulan çalışma hem maliyet hem de uygulanabilirlik açısında mevcut Mifare güvenlik açığını kapatmaktadır.

Mifare Tag'lardaki güvenlik açığı uygulamalara göre farklı sonuçlara neden olduğundan tezde sunulan çözüm uygulamadan bağımsız olaraktüm RFID sistemlerinde kullanılabilecek şekilde geliştirilmiştir. Mifare Tag ve okuyucuların güvenlik açığının kapatılması için geliştirilen sistemin bazı olumsuzluklarının da olmaması düşünülemez. Güvenlik açığı olan Mifare Tag ve okuyucularının ilk okuma, okuma ve yazma işlemlerini ortalama 10-100ms'de yapmasına karşılık geliştirilen sistem aynı işlemi 800-900ms sürede yapabilmektedir. Fakat bu sürelerin kendi aralarında kıyaslanarak incelenmesi doğru sonuç vermemektedir. Bu sürelerin insanların işlem yapma süreleri ile kıyaslanması gerekmektedir. Bunun dışında bu tezde laboratuar çalışmasıyla okuma işlemini yapan okuyucu yerine yüksek güvenlikli, düşük maliyetli mikroişlemci yazılımıyla donatılmış elektronik devre geliştirilmiştir. Bu sayede arttırma, azaltma, aktarma ve geri alma işlemleri Tag'ın kendi standart alanı üzerinde işlem yapma yerine bu cihazla yapılmıştır.

Geliştirilen sistem ile standart Mifare sistemi farklı uygulamalarda kıyaslandı. Bunun için RFID sistemlerin en yaygın uygulamalarına benzer uygulamalar oluşturularak 15 ay süreyle Sakarya Üniversitesinde deneme yapıldı. Denemelerde kullanılan uygulamalar şunlardır: Tag basımı, giriş/çıkış, sadece para yükleme, sadece burs yükleme, kioslarda para yükleme, burs yükleme, bakiye inceleme ve yemek işlemleri. Bu uygulamalar; isimlerinin farklı olmasının dışında piyasada kullanılan RFID sistemlerin hemen hemen hepsini kapsamaktadır. Ayrıca bu uygulama 2000km^2 alan içinde ve yaklaşık 45000 kişi ile yapıldığından her tür uygulama için uygulanabilir olduğu ispatlanmıştır. Ayrıca Sakarya Büyükşehir Belediyesi toplu taşıma araçlarında aynı sistem başarıyla test edilmiştir. Bu uygulamanın sonucunda görüldü ki farklı atak senaryolarında (Tag kopyalama, bilgi değiştirme, bakiye değiştirme) geliştirilen sistem standart MIFARE sistemine göre istenen güvenliği sağlamıştır. Bu işlemler PC kullanılmadan, online olmadan sadece okuyucuya bir mikroişlemci ilavesi ile sağlanmıştır.

Tag içinde mevcut bulunan 16 sektördeki 4 ayrı blokta bulunan AES ile şifrelenmiş datalar (tarih, saat, bakiye bilgisi, tek ay – çift ay) okuyucu tarafından kaydedildikten sonra şayet korsan kopyalama işlemi yapılırsa aynı değer okuyucu tarafından tekrar sunucuya bildirileceğinden dolayı sistem uyarı verecektir. Sadece tarih değeri bile alınsa; aynı değerli iki tarih (saat, dakika ve saniye bilgileriyle beraber) olamayacağından kart sistemden ihraç edilecektir.

Güvenlik için gerekli görülen bariyerlerin dışına çıkılarak hem Hash Fonksiyonu hem de AES şifreleme kullanılarak sistem online ve yarıonline olarak sürekli kontrolle de birlikte tespit edilen bir hataya rastlanmamıştır. Yemek tedarik firmasına ödenen ücret ile tahsil edilen ücretin de kıyaslaması yapıldığında sistemin düzgün çalıştığı ayrıca denetlenmiştir.

KAYNAKLAR

[1] COLLINS, J, Dexit Turn RFID Cards into Cash, http://www.rfidjournal.com/article/view/673 2010

[2] Mastercard PayPass, http://www.paypass.com 2010

[3] LOMAS, N, RFID could be in all cell phones by 2010, http://www.zdnet.com/news/rfid-could-be-in-all-cell-phones-by-2010/315292, 2009 2010

[4] NXP Semiconductors, http://www.mifare.net/products/smartcardics 2010

[5] ISO 14443, Identification cards-Contactless integrated circuit cards-Proximity cards, http://www.iso.org 2010

[6] ISO 15693, Identification cards-Contactless integrated circuit cards-Vicinity cards, http://www.iso.org 2010

[7] La Cryptographie Militaire, Journal des Science Militaires, 1883

[8] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45020 JTC 1/SC 17 2010

[9] WOLSCH, R, Technische Universitat Dresden, Fakultat Informatik, Seminararbeit, Almanya, 2008

[10] WANG, X., FENG, D., LAİ, X., YU, H., Collisions for Hash Functions MD4, MD5,HAVAL-128 and RIPEMD, Cryptology ePrint Archive, Report 2004/199, http://eprint.iacr.org, 2010.

[11] GANS, GK., HEOPMAN, JH., GARCIA, FD., A Practical Attack on the MIFARE Classic, London, UK, September 8-11, 2008, ISBN:978-3-540-85892-8

[12] NOHL, K., EVANS, D., PLOTZ, H., Reversr-Engineering a Cryptographic RFID Tag, 17th USENIX Security Symposium. San Jose, CA. July 28-August 1, 2008 Page 185

[13] COURTOIS, NT., NOHL, K., O'NEIL, S., Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards, http://eprint.iacr.org/2008/166 2010

[14] HENRICI, D., FLEUREN, T., MULLER, P., Sicherheit und Privatsphäre in RFID-Systemen: Ein Blick hinter die Kulissen, 11. Deutscher IT-Sicherheitscongress, 12-14 May 2009, Bonn, Germany Pages:1-13

[15] GARCIA, FD., ROSSUM, P., VERDULT, R., SCHREUR, W., Wirelessly Pickpocketing a Mifare Classic Card, Proceedings of the 2009 30th IEEE Symposium on Security and Privacy, 17-20 May 2009, Oakland, California, USA, Pages: 3-15

[16] Dismantling contactless smartcards, Radboud University Nijmagen, http://www2.ru.nl/media/pressrelease.pdf, 2010

[17] VERDULT, R., Radboud University Nijmagen, Digital Security group, http://www2.ru.nl/ds/research, 2010

[18] COURTOIS, NT., The Dark Side of Security by Obscurity – and Cloning MiFare, SECRYPT 2009, Milan, Italy, July 7-10, 2009, Page: 331-338

[19] COURTUIS, NT., PIEPRZYK, J., Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, 8th Criptology and Information Security Conference, December 01-05, 2002, Queenstown, New Zealand, Pages:267

[20] FERGUSON N., KELSEY, J., LUCKS, S., SCHNEIER, B., STAY, M., WAGNER, D., WHITING, D., Improved Cryptanalysis of Rijndael, Fast Software Encryption, Volume 1978/2001, Springer Berlin/heidelberg, Page:136-141

[21] BERNSTEIN, D, Cache-timing attacks on AES, http://cr.yp.to/papers.html#cachetiming, 2010

[22] BIRYUKOV, A., KHOVRATOVICH, D., Related-Key Cryptanalysis of the Full AES-192 and AES-256, Advances in Cryptology-ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, vol.5912, Springer, 2009, Pages:1-18

[23] YOUSSEF, M., TAVARES, S.E., Affine equivalence in the AES round function, Volume 148, Issue 2 (May 2005) Pages:161-170

[24] TEEPE, w., Making the Best of Mifare Classic, Radboud University Nijmagen, Digital Security group, http://www2.ru.nl/ds/research/rfid, 2010

[25] GARCIA, F.D., and 6 friends, Dismantling MIFARE Classic, 13th European Symposium on Research in Computer Security (ESORICS 2008). Edited by S. Jajodia and J. Lopez, 2008, Pages: 97-114

[26] NOHL, K., Cryptanalysis of Crypto-1, University of Virginia http://www.cs.virginia.edu/~kn5f/mifare.cryptanalysis.htm 2010

[27] HANCKE, G., A Practical Relay Attack on ISO 14443 Proximity Cards, University of Cambridge, 2005, http://www.rfidblog.org.uk/hancke-rfidrelay.pdf 2010

[28] BLASS, E.O., KURMUS, A., MOLVA, R., NOUBIR, G., SHIFKA, A., The Family of Protocols for RFID-Privacy and Autentication, 2010

[29] http://www.rsa.com/rsalabs/node.asp?id=2251 2010

[30] http://www.distributed.net 2010

[31] http://www.microsoft.com/security/default.aspx 2010

[32] 2007 Almanya Hannover Messe fuarında sergilenmiş "Dinamik Şifrelemeli Kampüs Otomasyon Sistemi"

[33] http://www.easypano.com, http://www.ptgui.com 2010

[34] GEER, D., Malicious bots threaten network security, Computer, Volume 38, Issue 1, Jan. 2005 Page(s): 18 – 20, Digital Object Identifier 10.1109/MC.2005.26

[35] CHEUNG, H., HAMLYN, A., CUNGANG, Y., Network security authentication of power system operations, , Ryerson Univ., Toronto, ON; This paper appears in: Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on,Publication Date: 4-7 May 2008, On page(s): 001687-001692, Location: Niagara Falls, ON, INSPEC Accession Number: 10103654, Digital Object Identifier: 10.1109/CCECE.2008.4564831, Current Version Published: 2008-07-15

[36] Network Security: Current Status and Future Directions, Christos Douligeris, Dimitrios N. Serpanos, Hardcover, 592 pages, June 2007, Wiley-IEEE Press

[37] NOHL, K., Cryptanalysis of Crypto-1 / University of Virginia. 2008. – Forschungsbericht

[38] http://www.md5encryption.com 2010

[39] Security Analysis of the Dutch OV-Chipkaart, TNO Information and Communication Technology, TNO report 34643, Netherlands

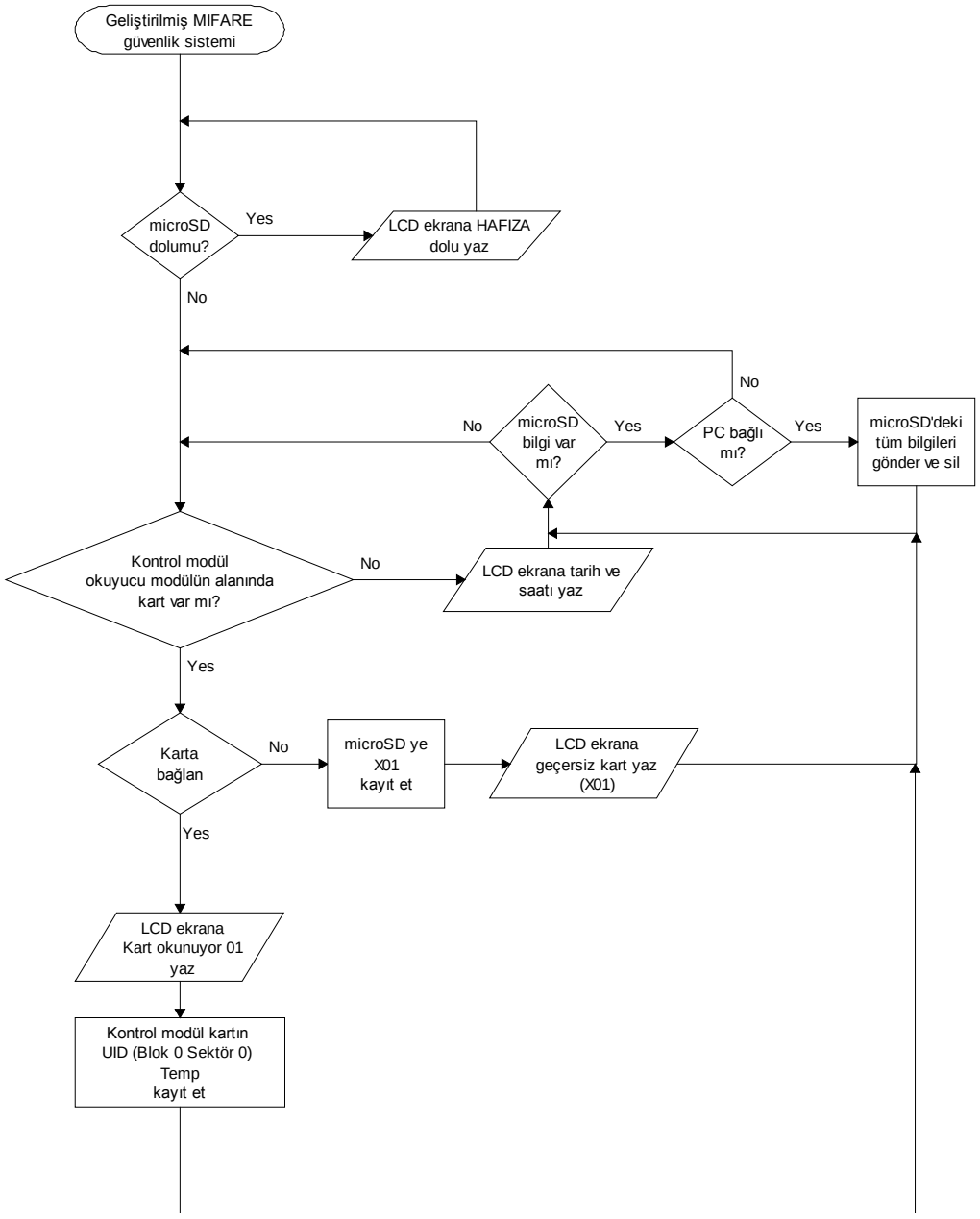
[40] http://www.computer.org/portal/web/csdl/doi/10.1109/MDM.2009.71 2010

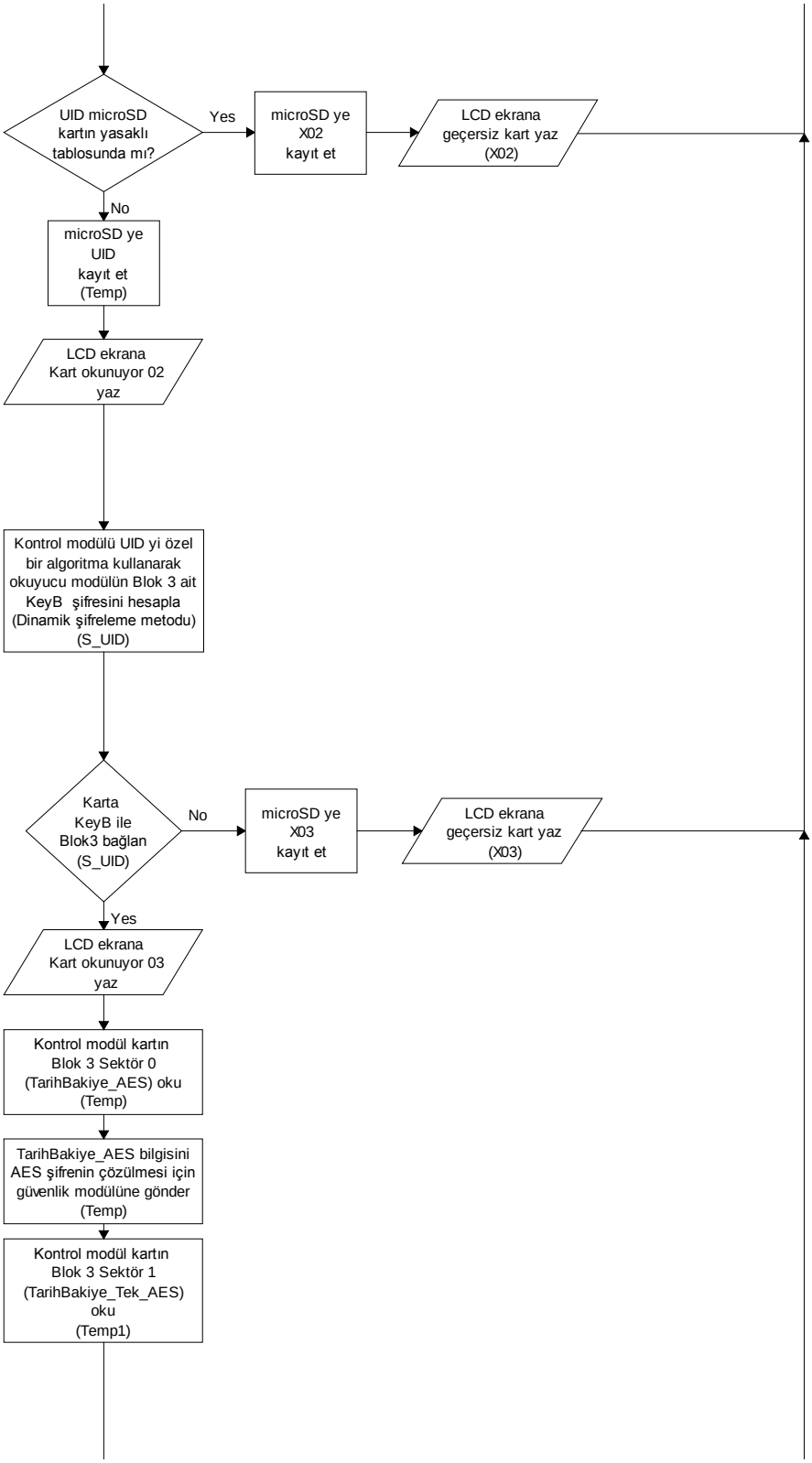
[41] http://www.computer.org/portal/web/csdl/doi/10.1109/CIS.2009.134 2010

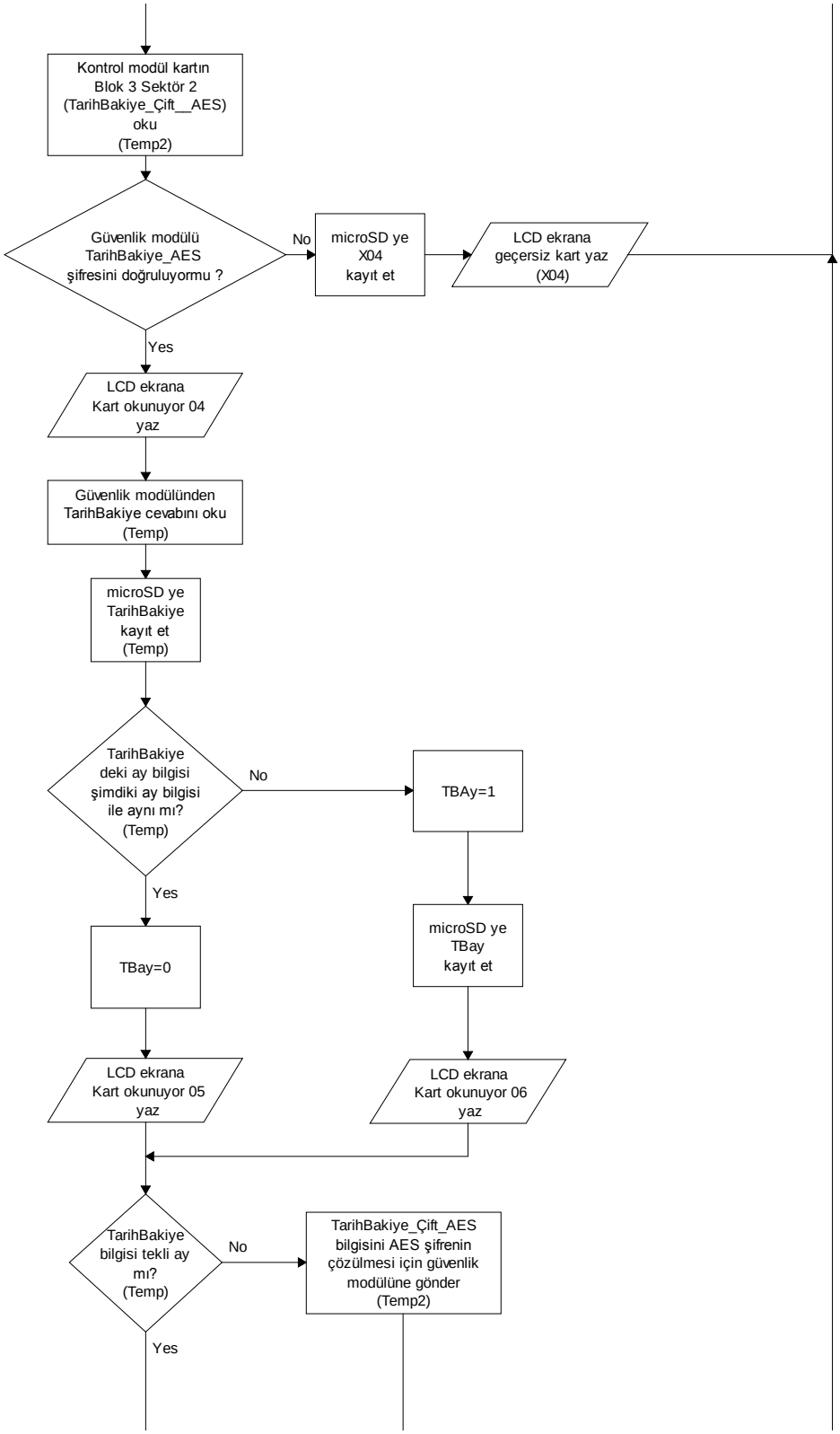
EKLER

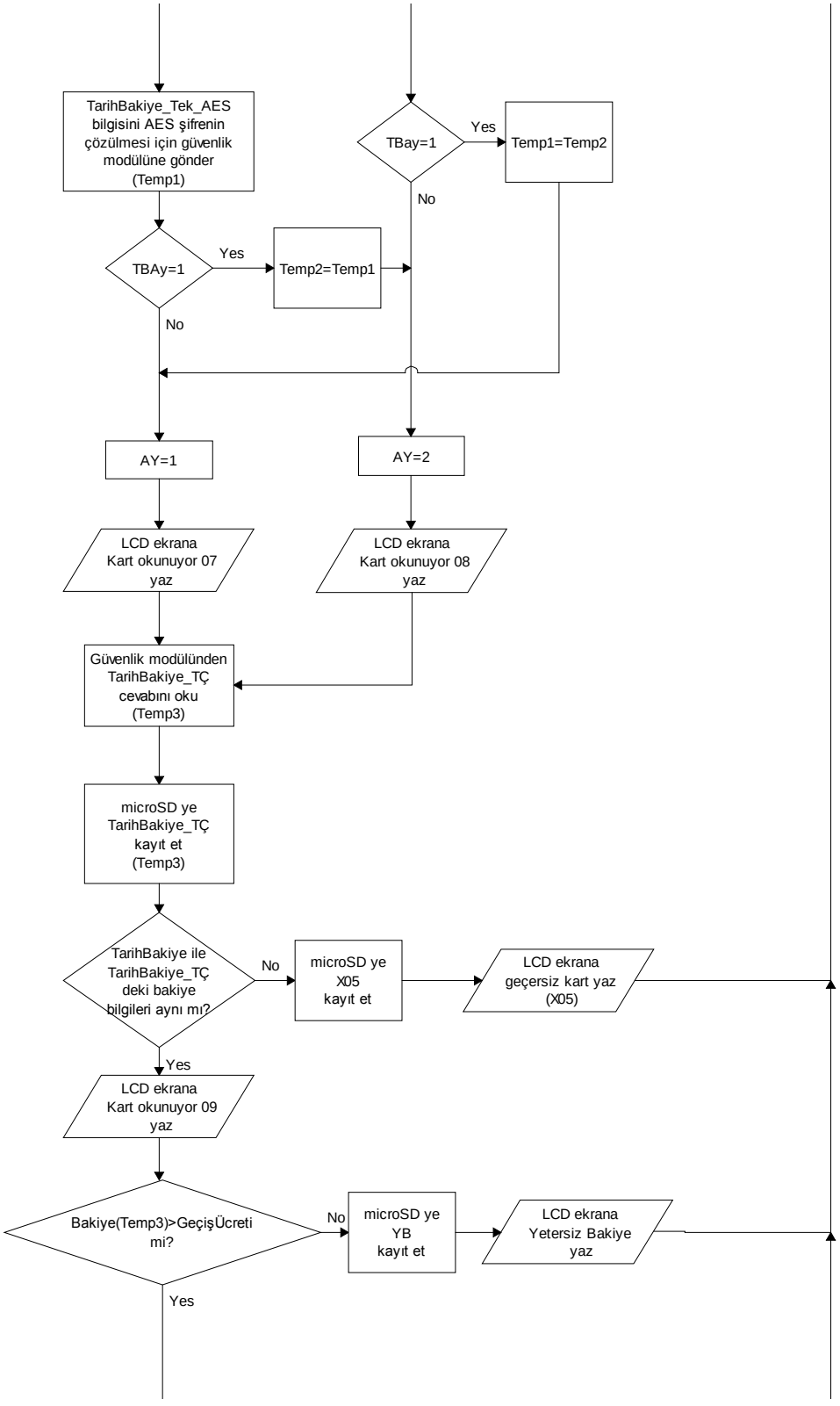
EK A

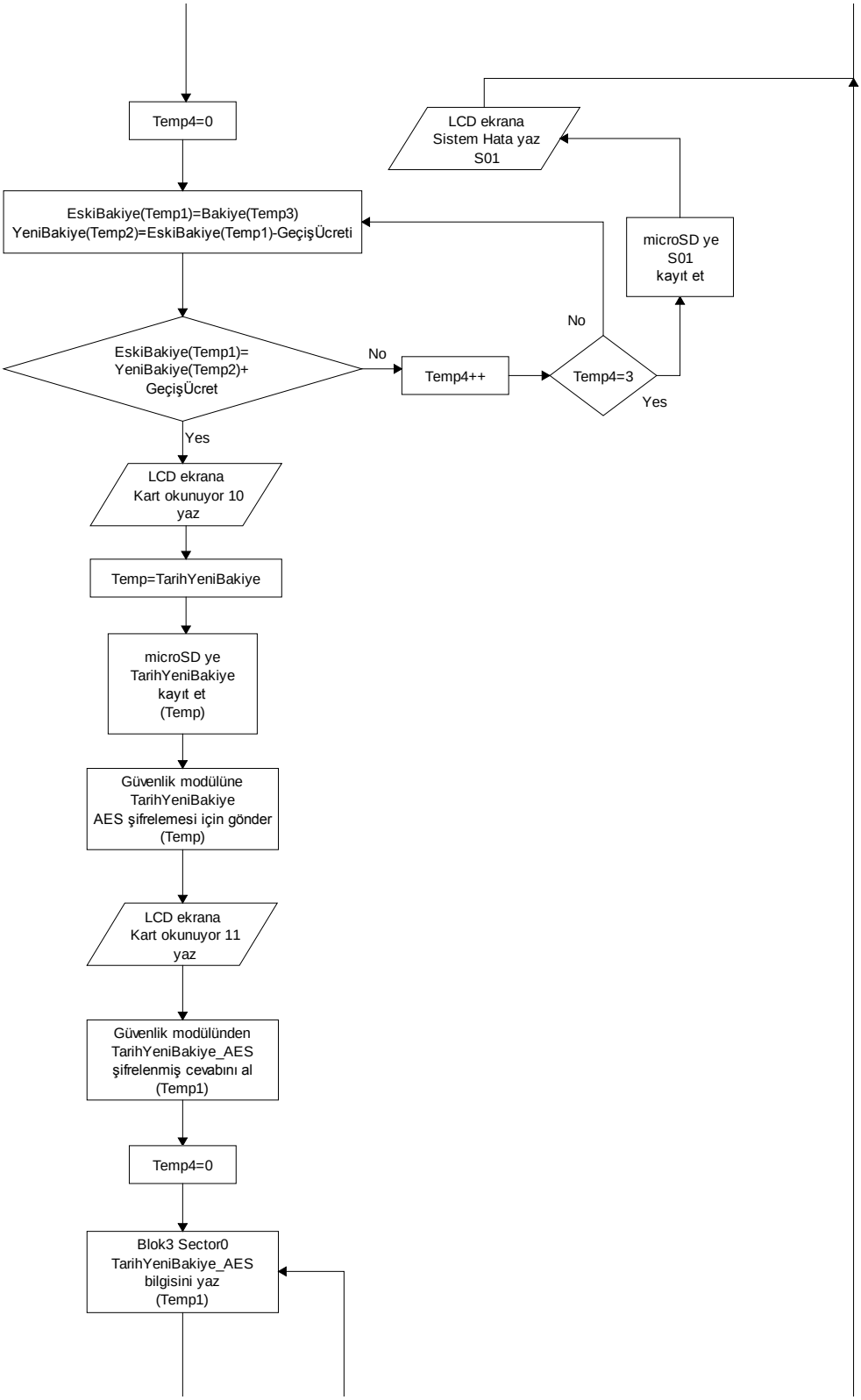
Akış Diyagramı

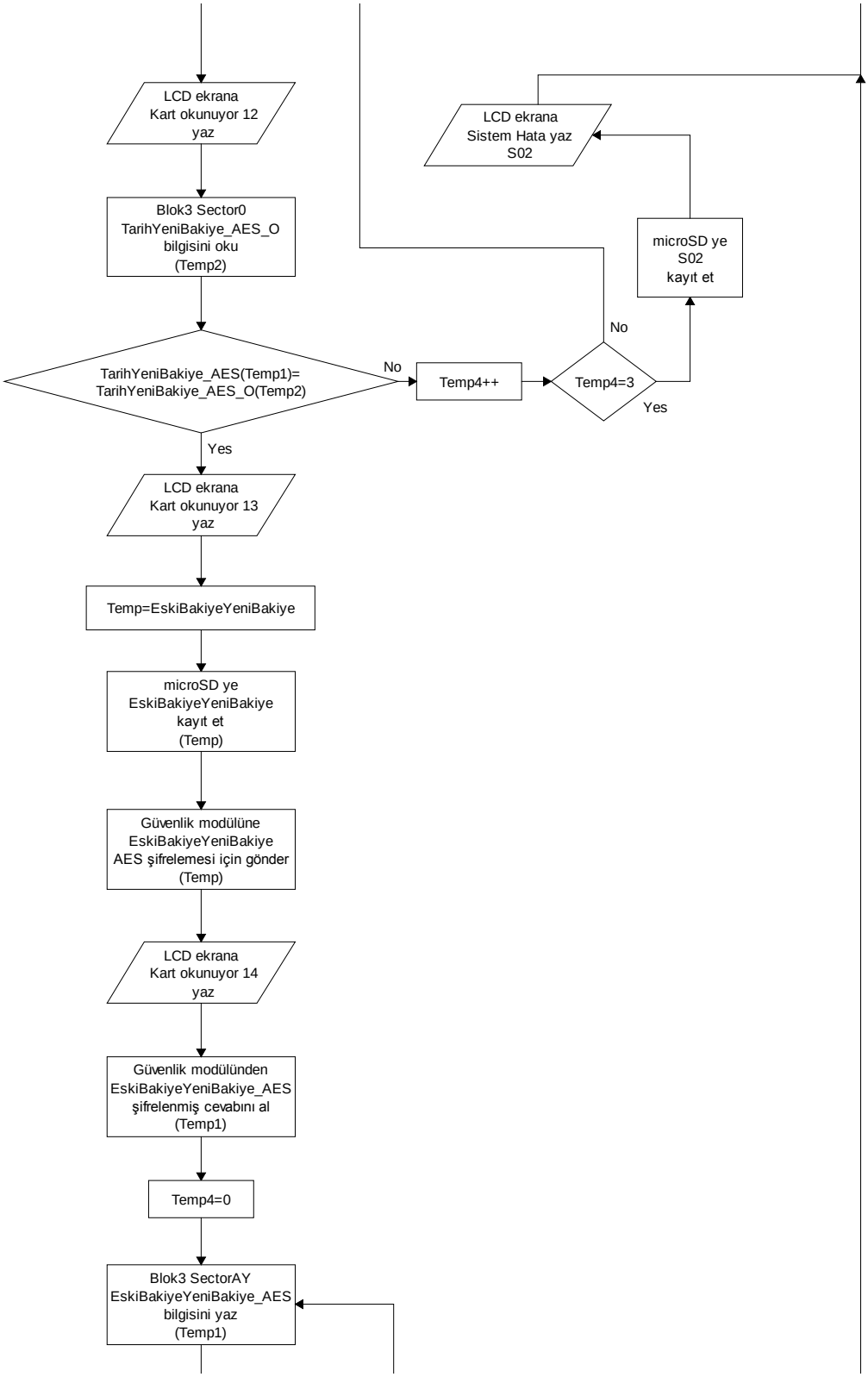


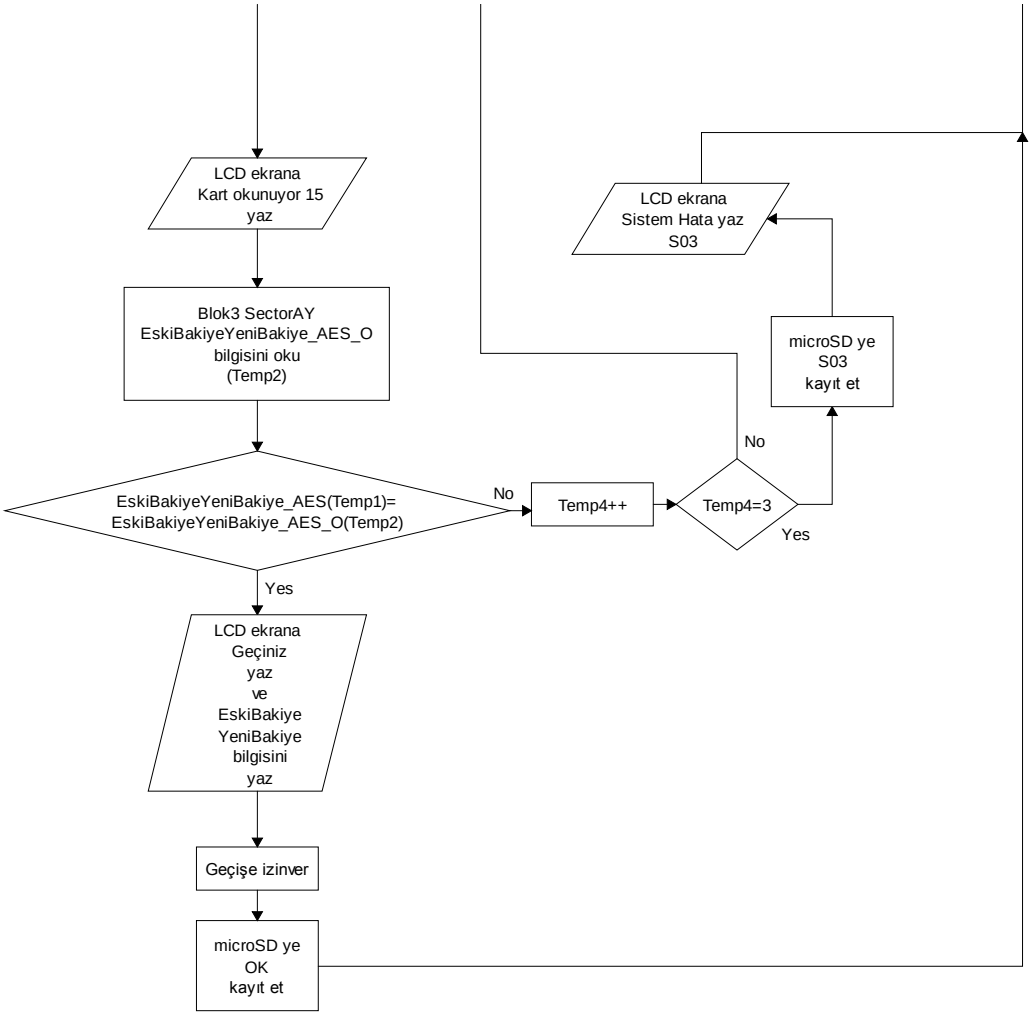

UID microSD kartın yasaklı tablosunda mı?
Yes
microSD ye X02 kayıt et
LCD ekrana geçersiz kart yaz (X02)
No
microSD ye UID kayıt et (Temp)
LCD ekrana Kart okunuyor 02 yaz
Kontrol modülü UID yi özel bir algoritma kullanarak okuyucu modülün Blok 3 ait KeyB şifresini hesapla (Dinamik şifreleme metodu) (S_UID)
Karta KeyB ile Blok3 bağlan (S_UID)
No
microSD ye X03 kayıt et
LCD ekrana geçersiz kart yaz (X03)
Yes
LCD ekrana Kart okunuyor 03 yaz
Kontrol modül kartın Blok 3 Sektör 0 (TarihBakiye_AES) oku (Temp)
TarihBakiye_AES bilgisini AES şifrenin çözülmesi için güvenlik modülüne gönder (Temp)
Kontrol modül kartın Blok 3 Sektör 1 (TarihBakiye_Tek_AES) oku (Temp1)


Kontrol modül kartın Blok 3 Sektör 2 (TarihBakiye_Çift__AES) oku (Temp2)
Güvenlik modülü TarihBakiye_AES şifresini doğruluyormu ?
No
microSD ye X04 kayıt et
LCD ekrana geçersiz kart yaz (X04)
Yes
LCD ekrana Kart okunuyor 04 yaz
Güvenlik modülünden TarihBakiye cevabını oku (Temp)
microSD ye TarihBakiye kayıt et (Temp)
TarihBakiye deki ay bilgisi şimdiki ay bilgisi ile aynı mı? (Temp)
No
TBAy=1
Yes
TBay=0
microSD ye TBay kayıt et
LCD ekrana Kart okunuyor 05 yaz
LCD ekrana Kart okunuyor 06 yaz
TarihBakiye bilgisi tekli ay mı? (Temp)
No
TarihBakiye_Çift_AES bilgisini AES şifrenin çözülmesi için güvenlik modülüne gönder (Temp2)
Yes


TarihBakiye_Tek_AES bilgisini AES şifrenin çözülmesi için güvenlik modülüne gönder (Temp1)
TBay=1
Yes
Temp1=Temp2
No
TBAy=1
Yes
Temp2=Temp1
No
AY=1
AY=2
LCD ekrana Kart okunuyor 07 yaz
LCD ekrana Kart okunuyor 08 yaz
Güvenlik modülünden TarihBakiye_TÇ cevabını oku (Temp3)
microSD ye TarihBakiye_TÇ kayıt et (Temp3)
TarihBakiye ile TarihBakiye_TÇ deki bakiye bilgileri aynı mı?
No
microSD ye X05 kayıt et
LCD ekrana geçersiz kart yaz (X05)
Yes
LCD ekrana Kart okunuyor 09 yaz
Bakiye(Temp3)>GeçişÜcreti mi?
No
microSD ye YB kayıt et
LCD ekrana Yetersiz Bakiye yaz
Yes


Temp4=0
LCD ekrana
Sistem Hata yaz
S01
EskiBakiye(Temp1)=Bakiye(Temp3)
YeniBakiye(Temp2)=EskiBakiye(Temp1)-GeçişÜcreti
microSD ye
S01
kayıt et
No
EskiBakiye(Temp1)=
YeniBakiye(Temp2)+
GeçişÜcret
No
Temp4++
Temp4=3
Yes
Yes
LCD ekrana
Kart okunuyor 10
yaz
Temp=TarihYeniBakiye
microSD ye
TarihYeniBakiye
kayıt et
(Temp)
Güvenlik modülüne
TarihYeniBakiye
AES şifrelemesi için gönder
(Temp)
LCD ekrana
Kart okunuyor 11
yaz
Güvenlik modülünden
TarihYeniBakiye_AES
şifrelenmiş cevabını al
(Temp1)
Temp4=0
Blok3 Sector0
TarihYeniBakiye_AES
bilgisini yaz
(Temp1)

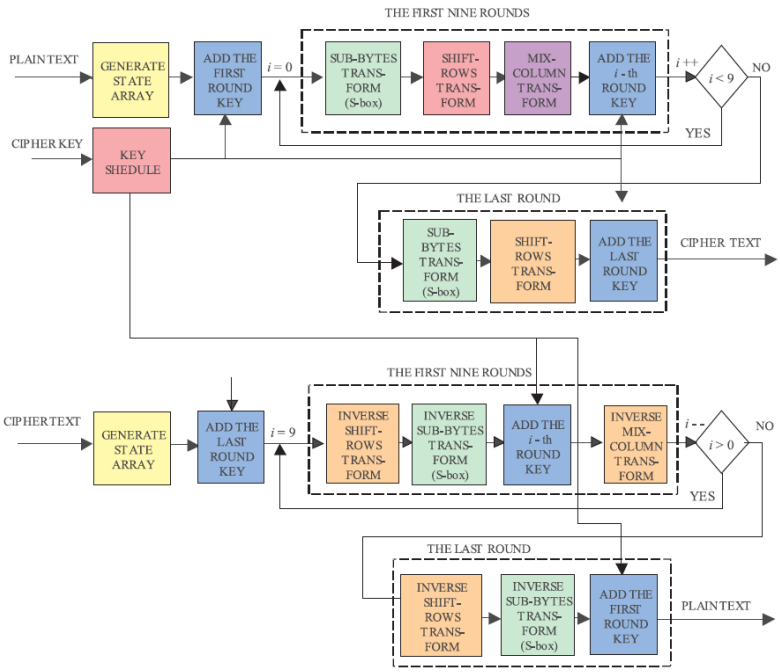

LCD ekrana
Kart okunuyor 12
yaz
Blok3 Sector0
TarihYeniBakiye_AES_O
bilgisini oku
(Temp2)
TarihYeniBakiye_AES(Temp1)=
TarihYeniBakiye_AES_O(Temp2)
No
Temp4++
Temp4=3
No
Yes
microSD ye
S02
kayıt et
LCD ekrana
Sistem Hata yaz
S02
Yes
LCD ekrana
Kart okunuyor 13
yaz
Temp=EskiBakiyeYeniBakiye
microSD ye
EskiBakiyeYeniBakiye
kayıt et
(Temp)
Güvenlik modülüne
EskiBakiyeYeniBakiye
AES şifrelemesi için gönder
(Temp)
LCD ekrana
Kart okunuyor 14
yaz
Güvenlik modülünden
EskiBakiyeYeniBakiye_AES
şifrelenmiş cevabını al
(Temp1)
Temp4=0
Blok3 SectorAY
EskiBakiyeYeniBakiye_AES
bilgisini yaz
(Temp1)


LCD ekrana
Kart okunuyor 15
yaz
Blok3 SectorAY
EskiBakiyeYeniBakiye_AES_O
bilgisini oku
(Temp2)
EskiBakiyeYeniBakiye_AES(Temp1)=
EskiBakiyeYeniBakiye_AES_O(Temp2)
No
Temp4++
Temp4=3
No
Yes
microSD ye
S03
kayıt et
LCD ekrana
Sistem Hata yaz
S03
Yes
LCD ekrana
Geçiniz
yaz
ve
EskiBakiye
YeniBakiye
bilgisini
yaz
Geçişe izinver
microSD ye
OK
kayıt et

Ek-B

AES (Advanced Encryption Standart)

AES kullanılmadan önce 1976 yılında geliştirilen DES (Data Encryption Standart) kullanılmaktaydı. DES 56 Bitlik bir şifre anahtar uzunluğuna sahiptir. 1990 yılında yeterli güvenliği sağlayamamasından dolayı 3DES geliştirildi. Fakat bu 112 Bitlik bir şifre anahtara sahipti ve 1997 yılında çok yavaş çalışmaktaydı. Bundan dolayı Amerikan National Institute of Standart and Technology (NIST) dünya genelinde güvenli, hızlı ve basit uygulanabilir bir algoritmanın geliştirilmesi için yarışma düzenledi. Uluslar arası bir komite birçok adayın algoritmasını 2000 yılının sonuna kadar farklı ülkeler ve platformlarda tartışarak AES'i seçtiler. AES'e yakın diğer algoritmalar sırasıyla MARS, Serpent ve Twofish'tir. AES'in tüm algoritması açık ve ücretsiz olarak herkese sağlanmaktadır. AES'in blok yapısı Şekil Ek-B.1'deki gibidir.



Şekil Ek-B.1. AES Blok yapısı

Programı da aşağıdaki gibidir. AES ile şifrelenenin çözülememesi için sadece şifrenin güçlü olması yeterlidir. Bu çalışmada AES çalışma mantığı durulmamıştır. Doğrudan C## kodu PIC asm'ye aktarıldı. PIC Asm kodunda sadece AES 128 bit ile çalışmaktadır. Yazılım kodu aşağıdaki gibidir.

```
//Rijndael.h

#ifndef __RIJNDAEL_H__
#define __RIJNDAEL_H__

#include <exception>
#include <cstring>

using namespace std;

//Rijndael (pronounced Reindaal) is a block cipher, designed by Joan Daemen and Vincent Rijmen as a candidate algorithm for the AES.
//The cipher has a variable block length and key length. The authors currently specify how to use keys with a length
//of 128, 192, or 256 bits to encrypt blocks with al length of 128, 192 or 256 bits (all nine combinations of
//key length and block length are possible). Both block length and key length can be extended very easily to
// multiples of 32 bits.
//Rijndael can be implemented very efficiently on a wide range of processors and in hardware.
//This implementation is based on the Java Implementation used with the Cryptix toolkit found at:
//http://www.esat.kuleuven.ac.be/~rijmen/rijndael/rijndael.zip
//Java code authors: Raif S. Naffah, Paulo S. L. M. Barreto
//This Implementation was tested against KAT test published by the authors of the method and the
//results were identical.
class CRijndael
{
public:
	//Operation Modes
	//The Electronic Code Book (ECB), Cipher Block Chaining (CBC) and Cipher Feedback Block (CFB) modes
	//are implemented.
	//In ECB mode if the same block is encrypted twice with the same key, the resulting
	//ciphertext blocks are the same.
	//In CBC Mode a ciphertext block is obtained by first xoring the
	//plaintext block with the previous ciphertext block, and encrypting the resulting value.
```

```
	//In CFB mode a ciphertext block is obtained by encrypting the previous
ciphertext block
	//and xoring the resulting value with the plaintext.
	enum { ECB=0, CBC=1, CFB=2 };

private:
	enum { DEFAULT_BLOCK_SIZE=16 };
	enum { MAX_BLOCK_SIZE=32, MAX_ROUNDS=14, MAX_KC=8,
MAX_BC=8 };

	//Auxiliary Functions
	//Multiply two elements of GF(2^m)
	static int Mul(int a, int b)
	{
		return (a != 0 && b != 0) ? sm_alog[(sm_log[a & 0xFF] + sm_log[b
& 0xFF]) % 255] : 0;
	}

	//Convenience method used in generating Transposition Boxes
	static int Mul4(int a, char b[])
	{
		if(a == 0)
			return 0;
		a = sm_log[a & 0xFF];
		int a0 = (b[0] != 0) ? sm_alog[(a + sm_log[b[0] & 0xFF]) % 255] &
0xFF : 0;
		int a1 = (b[1] != 0) ? sm_alog[(a + sm_log[b[1] & 0xFF]) % 255] &
0xFF : 0;
		int a2 = (b[2] != 0) ? sm_alog[(a + sm_log[b[2] & 0xFF]) % 255] &
0xFF : 0;
		int a3 = (b[3] != 0) ? sm_alog[(a + sm_log[b[3] & 0xFF]) % 255] &
0xFF : 0;
		return a0 << 24 | a1 << 16 | a2 << 8 | a3;
	}

public:
	//CONSTRUCTOR
	CRijndael();

	//DESTRUCTOR
	virtual ~CRijndael();

	//Expand a user-supplied key material into a session key.
	// key		- The 128/192/256-bit user-key to use.
	// chain	- initial chain block for CBC and CFB modes.
	// keylength  - 16, 24 or 32 bytes
	// blockSize  - The block size in bytes of this Rijndael (16, 24 or 32 bytes).
	void MakeKey(char const* key, char const* chain, int
keylength=DEFAULT_BLOCK_SIZE, int blockSize=DEFAULT_BLOCK_SIZE);
```

```
private:
        //Auxiliary Function
        void Xor(char* buff, char const* chain)
        {
                if(false==m_bKeyInit)
                        throw exception(sm_szErrorMsg1);
                for(int i=0; i<m_blockSize; i++)
                        *(buff++) ^= *(chain++);
        }

        //Convenience method to encrypt exactly one block of plaintext, assuming
        //Rijndael's default block size (128-bit).
        // in         - The plaintext
        // result     - The ciphertext generated from a plaintext using the key
        void DefEncryptBlock(char const* in, char* result);

        //Convenience method to decrypt exactly one block of plaintext, assuming
        //Rijndael's default block size (128-bit).
        // in         - The ciphertext.
        // result     - The plaintext generated from a ciphertext using the session key.
        void DefDecryptBlock(char const* in, char* result);

public:
        //Encrypt exactly one block of plaintext.
        // in          - The plaintext.
    // result      - The ciphertext generated from a plaintext using the key.
    void EncryptBlock(char const* in, char* result);

        //Decrypt exactly one block of ciphertext.
        // in         - The ciphertext.
        // result     - The plaintext generated from a ciphertext using the session key.
        void DecryptBlock(char const* in, char* result);

        void Encrypt(char const* in, char* result, size_t n, int iMode=ECB);

        void Decrypt(char const* in, char* result, size_t n, int iMode=ECB);

        //Get Key Length
        int GetKeyLength()
        {
                if(false==m_bKeyInit)
                        throw exception(sm_szErrorMsg1);
                return m_keylength;
        }

        //Block Size
        int       GetBlockSize()
        {
```

```
            if(false==m_bKeyInit)
                throw exception(sm_szErrorMsg1);
            return m_blockSize;
        }

        //Number of Rounds
        int GetRounds()
        {
            if(false==m_bKeyInit)
                throw exception(sm_szErrorMsg1);
            return m_iROUNDS;
        }

        void ResetChain()
        {
            memcpy(m_chain, m_chain0, m_blockSize);
        }

public:
        //Null chain
        static char const* sm_chain0;

private:
        static const int sm_alog[256];
        static const int sm_log[256];
        static const char sm_S[256];
    static const char sm_Si[256];
    static const int sm_T1[256];
    static const int sm_T2[256];
    static const int sm_T3[256];
    static const int sm_T4[256];
    static const int sm_T5[256];
    static const int sm_T6[256];
    static const int sm_T7[256];
    static const int sm_T8[256];
    static const int sm_U1[256];
    static const int sm_U2[256];
    static const int sm_U3[256];
    static const int sm_U4[256];
    static const char sm_rcon[30];
    static const int sm_shifts[3][4][2];
        //Error Messages
        static char const* sm_szErrorMsg1;
        static char const* sm_szErrorMsg2;
        //Key Initialization Flag
        bool m_bKeyInit;
        //Encryption (m_Ke) round key
        int m_Ke[MAX_ROUNDS+1][MAX_BC];
        //Decryption (m_Kd) round key
```

```
   int m_Kd[MAX_ROUNDS+1][MAX_BC];
        //Key Length
        int m_keylength;
        //Block Size
        int       m_blockSize;
        //Number of Rounds
        int m_iROUNDS;
        //Chain Block
        char m_chain0[MAX_BLOCK_SIZE];
        char m_chain[MAX_BLOCK_SIZE];
        //Auxiliary private use buffers
        int tk[MAX_KC];
        int a[MAX_BC];
        int t[MAX_BC];
};

#endif // __RIJNDAEL_H__
```

```
//Rijndael.cpp

#include <cstring>
#include <exception>
#include "Rijndael.h"

const int CRijndael::sm_alog[256] =
{
        1, 3, 5, 15, 17, 51, 85, 255, 26, 46, 114, 150, 161, 248, 19, 53,
        95, 225, 56, 72, 216, 115, 149, 164, 247, 2, 6, 10, 30, 34, 102, 170,
        229, 52, 92, 228, 55, 89, 235, 38, 106, 190, 217, 112, 144, 171, 230, 49,
        83, 245, 4, 12, 20, 60, 68, 204, 79, 209, 104, 184, 211, 110, 178, 205,
        76, 212, 103, 169, 224, 59, 77, 215, 98, 166, 241, 8, 24, 40, 120, 136,
        131, 158, 185, 208, 107, 189, 220, 127, 129, 152, 179, 206, 73, 219, 118,
154,
        181, 196, 87, 249, 16, 48, 80, 240, 11, 29, 39, 105, 187, 214, 97, 163,
        254, 25, 43, 125, 135, 146, 173, 236, 47, 113, 147, 174, 233, 32, 96, 160,
        251, 22, 58, 78, 210, 109, 183, 194, 93, 231, 50, 86, 250, 21, 63, 65,
        195, 94, 226, 61, 71, 201, 64, 192, 91, 237, 44, 116, 156, 191, 218, 117,
        159, 186, 213, 100, 172, 239, 42, 126, 130, 157, 188, 223, 122, 142, 137,
128,
        155, 182, 193, 88, 232, 35, 101, 175, 234, 37, 111, 177, 200, 67, 197, 84,
        252, 31, 33, 99, 165, 244, 7, 9, 27, 45, 119, 153, 176, 203, 70, 202,
        69, 207, 74, 222, 121, 139, 134, 145, 168, 227, 62, 66, 198, 81, 243, 14,
        18, 54, 90, 238, 41, 123, 141, 140, 143, 138, 133, 148, 167, 242, 13, 23,
        57, 75, 221, 124, 132, 151, 162, 253, 28, 36, 108, 180, 199, 82, 246, 1
};

const int CRijndael::sm_log[256] =
{
```

```
	0, 0, 25, 1, 50, 2, 26, 198, 75, 199, 27, 104, 51, 238, 223, 3,
	100, 4, 224, 14, 52, 141, 129, 239, 76, 113, 8, 200, 248, 105, 28, 193,
	125, 194, 29, 181, 249, 185, 39, 106, 77, 228, 166, 114, 154, 201, 9, 120,
	101, 47, 138, 5, 33, 15, 225, 36, 18, 240, 130, 69, 53, 147, 218, 142,
	150, 143, 219, 189, 54, 208, 206, 148, 19, 92, 210, 241, 64, 70, 131, 56,
	102, 221, 253, 48, 191, 6, 139, 98, 179, 37, 226, 152, 34, 136, 145, 16,
	126, 110, 72, 195, 163, 182, 30, 66, 58, 107, 40, 84, 250, 133, 61, 186,
	43, 121, 10, 21, 155, 159, 94, 202, 78, 212, 172, 229, 243, 115, 167, 87,
	175, 88, 168, 80, 244, 234, 214, 116, 79, 174, 233, 213, 231, 230, 173, 232,
	44, 215, 117, 122, 235, 22, 11, 245, 89, 203, 95, 176, 156, 169, 81, 160,
	127, 12, 246, 111, 23, 196, 73, 236, 216, 67, 31, 45, 164, 118, 123, 183,
	204, 187, 62, 90, 251, 96, 177, 134, 59, 82, 161, 108, 170, 85, 41, 157,
	151, 178, 135, 144, 97, 190, 220, 252, 188, 149, 207, 205, 55, 63, 91, 209,
	83, 57, 132, 60, 65, 162, 109, 71, 20, 42, 158, 93, 86, 242, 211, 171,
	68, 17, 146, 217, 35, 32, 46, 137, 180, 124, 184, 38, 119, 153, 227, 165,
	103, 74, 237, 222, 197, 49, 254, 24, 13, 99, 140, 128, 192, 247, 112, 7
};

const char CRijndael::sm_S[256] =
{
	99, 124, 119, 123, -14, 107, 111, -59, 48, 1, 103, 43, -2, -41, -85, 118,
	-54, -126, -55, 125, -6, 89, 71, -16, -83, -44, -94, -81, -100, -92, 114, -64,
	-73, -3, -109, 38, 54, 63, -9, -52, 52, -91, -27, -15, 113, -40, 49, 21,
	4, -57, 35, -61, 24, -106, 5, -102, 7, 18, -128, -30, -21, 39, -78, 117,
	9, -125, 44, 26, 27, 110, 90, -96, 82, 59, -42, -77, 41, -29, 47, -124,
	83, -47, 0, -19, 32, -4, -79, 91, 106, -53, -66, 57, 74, 76, 88, -49,
	-48, -17, -86, -5, 67, 77, 51, -123, 69, -7, 2, 127, 80, 60, -97, -88,
	81, -93, 64, -113, -110, -99, 56, -11, -68, -74, -38, 33, 16, -1, -13, -46,
	-51, 12, 19, -20, 95, -105, 68, 23, -60, -89, 126, 61, 100, 93, 25, 115,
	96, -127, 79, -36, 34, 42, -112, -120, 70, -18, -72, 20, -34, 94, 11, -37,
	-32, 50, 58, 10, 73, 6, 36, 92, -62, -45, -84, 98, -111, -107, -28, 121,
	-25, -56, 55, 109, -115, -43, 78, -87, 108, 86, -12, -22, 101, 122, -82, 8,
	-70, 120, 37, 46, 28, -90, -76, -58, -24, -35, 116, 31, 75, -67, -117, -118,
	112, 62, -75, 102, 72, 3, -10, 14, 97, 53, 87, -71, -122, -63, 29, -98,
	-31, -8, -104, 17, 105, -39, -114, -108, -101, 30, -121, -23, -50, 85, 40, -33,
	-116, -95, -119, 13, -65, -26, 66, 104, 65, -103, 45, 15, -80, 84, -69, 22
};

const char CRijndael::sm_Si[256] =
{
	82, 9, 106, -43, 48, 54, -91, 56, -65, 64, -93, -98, -127, -13, -41, -5,
	124, -29, 57, -126, -101, 47, -1, -121, 52, -114, 67, 68, -60, -34, -23, -53,
	84, 123, -108, 50, -90, -62, 35, 61, -18, 76, -107, 11, 66, -6, -61, 78,
	8, 46, -95, 102, 40, -39, 36, -78, 118, 91, -94, 73, 109, -117, -47, 37,
	114, -8, -10, 100, -122, 104, -104, 22, -44, -92, 92, -52, 93, 101, -74, -110,
	108, 112, 72, 80, -3, -19, -71, -38, 94, 21, 70, 87, -89, -115, -99, -124,
	-112, -40, -85, 0, -116, -68, -45, 10, -9, -28, 88, 5, -72, -77, 69, 6,
	-48, 44, 30, -113, -54, 63, 15, 2, -63, -81, -67, 3, 1, 19, -118, 107,
	58, -111, 17, 65, 79, 103, -36, -22, -105, -14, -49, -50, -16, -76, -26, 115,
```

```
        -106, -84, 116, 34, -25, -83, 53, -123, -30, -7, 55, -24, 28, 117, -33, 110,
        71, -15, 26, 113, 29, 41, -59, -119, 111, -73, 98, 14, -86, 24, -66, 27,
        -4, 86, 62, 75, -58, -46, 121, 32, -102, -37, -64, -2, 120, -51, 90, -12,
        31, -35, -88, 51, -120, 7, -57, 49, -79, 18, 16, 89, 39, -128, -20, 95,
        96, 81, 127, -87, 25, -75, 74, 13, 45, -27, 122, -97, -109, -55, -100, -17,
        -96, -32, 59, 77, -82, 42, -11, -80, -56, -21, -69, 60, -125, 83, -103, 97,
        23, 43, 4, 126, -70, 119, -42, 38, -31, 105, 20, 99, 85, 33, 12, 125
};

const int CRijndael::sm_T1[256] =
{
        -966564955, -126059388, -294160487, -159679603,
        -855539, -697603139, -563122255, -1849309868,
        1613770832, 33620227, -832084055, 1445669757,
        -402719207, -1244145822, 1303096294, -327780710,
        -1882535355, 528646813, -1983264448, -92439161,
        -268764651, -1302767125, -1907931191, -68095989,
        1101901292, -1277897625, 1604494077, 1169141738,
        597466303, 1403299063, -462261610, -1681866661,
        1974974402, -503448292, 1033081774, 1277568618,
        1815492186, 2118074177, -168298750, -2083730353,
        1748251740, 1369810420, -773462732, -101584632,
        -495881837, -1411852173, 1647391059, 706024767,
        134480908, -1782069422, 1176707941, -1648114850,
        806885416, 932615841, 168101135, 798661301,
        235341577, 605164086, 461406363, -538779075,
        -840176858, 1311188841, 2142417613, -361400929,
        302582043, 495158174, 1479289972, 874125870,
        907746093, -596742478, -1269146898, 1537253627,
        -1538108682, 1983593293, -1210657183, 2108928974,
        1378429307, -572267714, 1580150641, 327451799,
        -1504488459, -1177431704, 0, -1041371860,
        1075847264, -469959649, 2041688520, -1235526675,
        -731223362, -1916023994, 1740553945, 1916352843,
        -1807070498, -1739830060, -1336387352, -2049978550,
        -1143943061, -974131414, 1336584933, -302253290,
        -2042412091, -1706209833, 1714631509, 293963156,
        -1975171633, -369493744, 67240454, -25198719,
        -1605349136, 2017213508, 631218106, 1269344483,
        -1571728909, 1571005438, -2143272768, 93294474,
        1066570413, 563977660, 1882732616, -235539196,
        1673313503, 2008463041, -1344611723, 1109467491,
        537923632, -436207846, -34344178, -1076702611,
        -2117218996, 403442708, 638784309, -1007883217,
        -1101045791, 899127202, -2008791860, 773265209,
        -1815821225, 1437050866, -58818942, 2050833735,
        -932944724, -1168286233, 840505643, -428641387,
        -1067425632, 427917720, -1638969391, -1545806721,
        1143087718, 1412049534, 999329963, 193497219,
```

```
        -1941551414, -940642775, 1807268051, 672404540,
        -1478566279, -1134666014, 369822493, -1378100362,
        -606019525, 1681011286, 1949973070, 336202270,
        -1840690725, 201721354, 1210328172, -1201906460,
        -1614626211, -1110191250, 1135389935, -1000185178,
        965841320, 831886756, -739974089, -226920053,
        -706222286, -1949775805, 1849112409, -630362697,
        26054028, -1311386268, -1672589614, 1235855840,
        -663982924, -1403627782, -202050553, -806688219,
        -899324497, -193299826, 1202630377, 268961816,
        1874508501, -260540280, 1243948399, 1546530418,
        941366308, 1470539505, 1941222599, -1748580783,
        -873928669, -1579295364, -395021156, 1042226977,
        -1773450275, 1639824860, 227249030, 260737669,
        -529502064, 2084453954, 1907733956, -865704278,
        -1874310952, 100860677, -134810111, 470683154,
        -1033805405, 1781871967, -1370007559, 1773779408,
        394692241, -1715355304, 974986535, 664706745,
        -639508168, -336005101, 731420851, 571543859,
        -764843589, -1445340816, 126783113, 865375399,
        765172662, 1008606754, 361203602, -907417312,
        -2016489911, -1437248001, 1344809080, -1512054918,
        59542671, 1503764984, 160008576, 437062935,
        1707065306, -672733647, -2076032314, -798463816,
        -2109652541, 697932208, 1512910199, 504303377,
        2075177163, -1470868228, 1841019862, 739644986
};

const int CRijndael::sm_T2[256] =
{
        -1513725085, -2064089988, -1712425097, -1913226373,
        234877682, -1110021269, -1310822545, 1418839493,
        1348481072, 50462977, -1446090905, 2102799147,
        434634494, 1656084439, -431117397, -1695779210,
        1167051466, -1658879358, 1082771913, -2013627011,
        368048890, -340633255, -913422521, 201060592,
        -331240019, 1739838676, -44064094, -364531793,
        -1088185188, -145513308, -1763413390, 1536934080,
        -1032472649, 484572669, -1371696237, 1783375398,
        1517041206, 1098792767, 49674231, 1334037708,
        1550332980, -195975771, 886171109, 150598129,
        -1813876367, 1940642008, 1398944049, 1059722517,
        201851908, 1385547719, 1699095331, 1587397571,
        674240536, -1590192490, 252314885, -1255171430,
        151914247, 908333586, -1692696448, 1038082786,
        651029483, 1766729511, -847269198, -1612024459,
        454166793, -1642232957, 1951935532, 775166490,
        758520603, -1294176658, -290170278, -77881184,
        -157003182, 1299594043, 1639438038, -830622797,
```

```
        2068982057, 1054729187, 1901997871, -1760328572,
        -173649069, 1757008337, 0, 750906861,
        1614815264, 535035132, -931548751, -306816165,
        -1093375382, 1183697867, -647512386, 1265776953,
        -560706998, -728216500, -391096232, 1250283471,
        1807470800, 717615087, -447763798, 384695291,
        -981056701, -677753523, 1432761139, -1810791035,
        -813021883, 283769337, 100925954, -2114027649,
        -257929136, 1148730428, -1171939425, -481580888,
        -207466159, -27417693, -1065336768, -1979347057,
        -1388342638, -1138647651, 1215313976, 82966005,
        -547111748, -1049119050, 1974459098, 1665278241,
        807407632, 451280895, 251524083, 1841287890,
        1283575245, 337120268, 891687699, 801369324,
        -507617441, -1573546089, -863484860, 959321879,
        1469301956, -229267545, -2097381762, 1199193405,
        -1396153244, -407216803, 724703513, -1780059277,
        -1598005152, -1743158911, -778154161, 2141445340,
        1715741218, 2119445034, -1422159728, -2096396152,
        -896776634, 700968686, -747915080, 1009259540,
        2041044702, -490971554, 487983883, 1991105499,
        1004265696, 1449407026, 1316239930, 504629770,
        -611169975, 168560134, 1816667172, -457679780,
        1570751170, 1857934291, -280777556, -1497079198,
        -1472622191, -1540254315, 936633572, -1947043463,
        852879335, 1133234376, 1500395319, -1210421907,
        -1946055283, 1689376213, -761508274, -532043351,
        -1260884884, -89369002, 133428468, 634383082,
        -1345690267, -1896580486, -381178194, 403703816,
        -714097990, -1997506440, 1867130149, 1918643758,
        607656988, -245913946, -948718412, 1368901318,
        600565992, 2090982877, -1662487436, 557719327,
        -577352885, -597574211, -2045932661, -2062579062,
        -1864339344, 1115438654, -999180875, -1429445018,
        -661632952, 84280067, 33027830, 303828494,
        -1547542175, 1600795957, -106014889, -798377543,
        -1860729210, 1486471617, 658119965, -1188585826,
        953803233, 334231800, -1288988520, 857870609,
        -1143838359, 1890179545, -1995993458, -1489791852,
        -1238525029, 574365214, -1844082809, 550103529,
        1233637070, -5614251, 2018519080, 2057691103,
        -1895592820, -128343647, -2146858615, 387583245,
        -630865985, 836232934, -964410814, -1194301336,
        -1014873791, -1339450983, 2002398509, 287182607,
        -881086288, -56077228, -697451589, 975967766
};

const int CRijndael::sm_T3[256] =
{
```

1671808611, 2089089148, 2006576759, 2072901243,
-233963534, 1807603307, 1873927791, -984313403,
810573872, 16974337, 1739181671, 729634347,
-31856642, -681396777, -1410970197, 1989864566,
-901410870, -2103631998, -918517303, 2106063485,
-99225606, 1508618841, 1204391495, -267650064,
-1377025619, -731401260, -1560453214, -1343601233,
-1665195108, -1527295068, 1922491506, -1067738176,
-1211992649, -48438787, -1817297517, 644500518,
911895606, 1061256767, -150800905, -867204148,
878471220, -1510714971, -449523227, -251069967,
1905517169, -663508008, 827548209, 356461077,
67897348, -950889017, 593839651, -1017209405,
405286936, -1767819370, 84871685, -1699401830,
118033927, 305538066, -2137318528, -499261470,
-349778453, 661212711, -1295155278, 1973414517,
152769033, -2086789757, 745822252, 439235610,
455947803, 1857215598, 1525593178, -1594139744,
1391895634, 994932283, -698239018, -1278313037,
695947817, -482419229, 795958831, -2070473852,
1408607827, -781665839, 0, -315833875,
543178784, -65018884, -1312261711, 1542305371,
1790891114, -884568629, -1093048386, 961245753,
1256100938, 1289001036, 1491644504, -817199665,
-798245936, -282409489, -1427812438, -82383365,
1137018435, 1305975373, 861234739, -2053893755,
1171229253, -116332039, 33948674, 2139225727,
1357946960, 1011120188, -1615190625, -1461498968,
1374921297, -1543610973, 1086357568, -1886780017,
-1834139758, -1648615011, 944271416, -184225291,
-1126210628, -1228834890, -629821478, 560153121,
271589392, -15014401, -217121293, -764559406,
-850624051, 202643468, 322250259, -332413972,
1608629855, -1750977129, 1154254916, 389623319,
-1000893500, -1477290585, 2122513534, 1028094525,
1689045092, 1575467613, 422261273, 1939203699,
1621147744, -2120738431, 1339137615, -595614756,
577127458, 712922154, -1867826288, -2004677752,
1187679302, -299251730, -1194103880, 339486740,
-562452514, 1591917662, 186455563, -612979237,
-532948000, 844522546, 978220090, 169743370,
1239126601, 101321734, 611076132, 1558493276,
-1034051646, -747717165, -1393605716, 1655096418,
-1851246191, -1784401515, -466103324, 2039214713,
-416098841, -935097400, 928607799, 1840765549,
-1920204403, -714821163, 1322425422, -1444918871,
1823791212, 1459268694, -200805388, -366620694,
1706019429, 2056189050, -1360443474, 135794696,
-1160417350, 2022240376, 628050469, 779246638,

```
	472135708, -1494132826, -1261997132, -967731258,
	-400307224, -579034659, 1956440180, 522272287,
	1272813131, -1109630531, -1954148981, -1970991222,
	1888542832, 1044544574, -1245417035, 1722469478,
	1222152264, 50660867, -167643146, 236067854,
	1638122081, 895445557, 1475980887, -1177523783,
	-2037311610, -1051158079, 489110045, -1632032866,
	-516367903, -132912136, -1733088360, 288563729,
	1773916777, -646927911, -1903622258, -1800981612,
	-1682559589, 505560094, -2020469369, -383727127,
	-834041906, 1442818645, 678973480, -545610273,
	-1936784500, -1577559647, -1988097655, 219617805,
	-1076206145, -432941082, 1120306242, 1756942440,
	1103331905, -1716508263, 762796589, 252780047,
	-1328841808, 1425844308, -1143575109, 372911126
};

const int CRijndael::sm_T4[256] =
{
	1667474886, 2088535288, 2004326894, 2071694838,
	-219017729, 1802223062, 1869591006, -976923503,
	808472672, 16843522, 1734846926, 724270422,
	-16901657, -673750347, -1414797747, 1987484396,
	-892713585, -2105369313, -909557623, 2105378810,
	-84273681, 1499065266, 1195886990, -252703749,
	-1381110719, -724277325, -1566376609, -1347425723,
	-1667449053, -1532692653, 1920112356, -1061135461,
	-1212693899, -33743647, -1819038147, 640051788,
	909531756, 1061110142, -134806795, -859025533,
	875846760, -1515850671, -437963567, -235861767,
	1903268834, -656903253, 825316194, 353713962,
	67374088, -943238507, 589522246, -1010606435,
	404236336, -1768513225, 84217610, -1701137105,
	117901582, 303183396, -2139055333, -488489505,
	-336910643, 656894286, -1296904833, 1970642922,
	151591698, -2088526307, 741110872, 437923380,
	454765878, 1852748508, 1515908788, -1600062629,
	1381168804, 993742198, -690593353, -1280061827,
	690584402, -471646499, 791638366, -2071685357,
	1398011302, -774805319, 0, -303223615,
	538992704, -50585629, -1313748871, 1532751286,
	1785380564, -875870579, -1094788761, 960056178,
	1246420628, 1280103576, 1482221744, -808498555,
	-791647301, -269538619, -1431640753, -67430675,
	1128514950, 1296947098, 859002214, -2054843375,
	1162203018, -101117719, 33687044, 2139062782,
	1347481760, 1010582648, -1616922075, -1465326773,
	1364325282, -1549533603, 1077985408, -1886418427,
	-1835881153, -1650607071, 943212656, -168491791,
```

```
	-1128472733, -1229536905, -623217233, 555836226,
	269496352, -58651, -202174723, -757961281,
	-842183551, 202118168, 320025894, -320065597,
	1600119230, -1751670219, 1145359496, 387397934,
	-993765485, -1482165675, 2122220284, 1027426170,
	1684319432, 1566435258, 421079858, 1936954854,
	1616945344, -2122213351, 1330631070, -589529181,
	572679748, 707427924, -1869567173, -2004319477,
	1179044492, -286381625, -1195846805, 336870440,
	-555845209, 1583276732, 185277718, -606374227,
	-522175525, 842159716, 976899700, 168435220,
	1229577106, 101059084, 606366792, 1549591736,
	-1027449441, -741118275, -1397952701, 1650632388,
	-1852725191, -1785355215, -454805549, 2038008818,
	-404278571, -926399605, 926374254, 1835907034,
	-1920103423, -707435343, 1313788572, -1448484791,
	1819063512, 1448540844, -185333773, -353753649,
	1701162954, 2054852340, -1364268729, 134748176,
	-1162160785, 2021165296, 623210314, 774795868,
	471606328, -1499008681, -1263220877, -960081513,
	-387439669, -572687199, 1953799400, 522133822,
	1263263126, -1111630751, -1953790451, -1970633457,
	1886425312, 1044267644, -1246378895, 1718004428,
	1212733584, 50529542, -151649801, 235803164,
	1633788866, 892690282, 1465383342, -1179004823,
	-2038001385, -1044293479, 488449850, -1633765081,
	-505333543, -117959701, -1734823125, 286339874,
	1768537042, -640061271, -1903261433, -1802197197,
	-1684294099, 505291324, -2021158379, -370597687,
	-825341561, 1431699370, 673740880, -539002203,
	-1936945405, -1583220647, -1987477495, 218961690,
	-1077945755, -421121577, 1111672452, 1751693520,
	1094828930, -1717981143, 757954394, 252645662,
	-1330590853, 1414855848, -1145317779, 370555436
};

const int CRijndael::sm_T5[256] =
{
	1374988112, 2118214995, 437757123, 975658646,
	1001089995, 530400753, -1392879445, 1273168787,
	540080725, -1384747530, -1999866223, -184398811,
	1340463100, -987051049, 641025152, -1251826801,
	-558802359, 632953703, 1172967064, 1576976609,
	-1020300030, -2125664238, -1924753501, 1809054150,
	59727847, 361929877, -1083344149, -1789765158,
	-725712083, 1484005843, 1239443753, -1899378620,
	1975683434, -191989384, -1722270101, 666464733,
	-1092530250, -259478249, -920605594, 2110667444,
	1675577880, -451268222, -1756286112, 1649639237,
```

-1318815776, -1150570876, -25059300, -116905068,
1883793496, -1891238631, -1797362553, 1383856311,
-1418472669, 1917518562, -484470953, 1716890410,
-1293211641, 800440835, -2033878118, -751368027,
807962610, 599762354, 33778362, -317291940,
-1966138325, -1485196142, -217582864, 1315562145,
1708848333, 101039829, -785096161, -995688822,
875451293, -1561111136, 92987698, -1527321739,
193195065, 1080094634, 1584504582, -1116860335,
1042385657, -1763899843, -583137874, 1306967366,
-1856729675, 1908694277, 67556463, 1615861247,
429456164, -692196969, -1992277044, 1742315127,
-1326955843, 126454664, -417768648, 2043211483,
-1585706425, 2084704233, -125559095, 0,
159417987, 841739592, 504459436, 1817866830,
-49348613, 260388950, 1034867998, 908933415,
168810852, 1750902305, -1688513327, 607530554,
202008497, -1822955761, -1259432238, 463180190,
-2134850225, 1641816226, 1517767529, 470948374,
-493635062, -1063245083, 1008918595, 303765277,
235474187, -225720403, 766945465, 337553864,
1475418501, -1351284916, -291906117, -1551933187,
-150919521, 1551037884, 1147550661, 1543208500,
-1958532746, -886847780, -1225917336, -1192955549,
-684598070, 1113818384, 328671808, -2067394272,
-2058738563, -759480840, -1359400431, -953573011,
496906059, -592301837, 226906860, 2009195472,
733156972, -1452230247, 294930682, 1206477858,
-1459843900, -1594867942, 1451044056, 573804783,
-2025238841, -650587711, -1932877058, -1730933962,
-1493859889, -1518674392, -625504730, 1068351396,
742039012, 1350078989, 1784663195, 1417561698,
-158526526, -1864845080, 775550814, -2101104651,
-1621262146, 1775276924, 1876241833, -819653965,
-928212677, 270040487, -392404114, -616842373,
-853116919, 1851332852, -325404927, -2091935064,
-426414491, -1426069890, 566021896, -283776794,
-1159226407, 1248802510, -358676012, 699432150,
832877231, 708780849, -962227152, 899835584,
1951317047, -58537306, -527380304, 866637845,
-251357110, 1106041591, 2144161806, 395441711,
1984812685, 1139781709, -861254316, -459930401,
-1630423581, 1282050075, -1054072904, 1181045119,
-1654724092, 25965917, -91786125, -83148498,
-1285087910, -1831087534, -384805325, 1842759443,
-1697160820, 933301370, 1509430414, -351060855,
-827774994, -1218328267, -518199827, 2051518780,
-1663901863, 1441952575, 404016761, 1942435775,
1408749034, 1610459739, -549621996, 2017778566,

```
        -894438527, -1184316354, 941896748, -1029488545,
        371049330, -1126030068, 675039627, -15887039,
        967311729, 135050206, -659233636, 1683407248,
        2076935265, -718096784, 1215061108, -793225406
};

const int CRijndael::sm_T6[256] =
{
        1347548327, 1400783205, -1021700188, -1774573730,
        -885281941, -249586363, -1414727080, -1823743229,
        1428173050, -156404115, -1853305738, 636813900,
        -61872681, -674944309, -2144979644, -1883938141,
        1239331162, 1730525723, -1740248562, -513933632,
        46346101, 310463728, -1551022441, -966011911,
        -419197089, -1793748324, -339776134, -627748263,
        768917123, -749177823, 692707433, 1150208456,
        1786102409, 2029293177, 1805211710, -584599183,
        -1229004465, 401639597, 1724457132, -1266823622,
        409198410, -2098914767, 1620529459, 1164071807,
        -525245321, -2068091986, 486441376, -1795618773,
        1483753576, 428819965, -2020286868, -1219331080,
        598438867, -495826174, 1474502543, 711349675,
        129166120, 53458370, -1702443653, -1512884472,
        -231724921, -1306280027, -1174273174, 1559041666,
        730517276, -1834518092, -252508174, -1588696606,
        -848962828, -721025602, 533804130, -1966823682,
        -1657524653, -1599933611, 839224033, 1973745387,
        957055980, -1438621457, 106852767, 1371368976,
        -113368694, 1033297158, -1361232379, 1179510461,
        -1248766835, 91341917, 1862534868, -10465259,
        605657339, -1747534359, -863420349, 2003294622,
        -1112479678, -2012771957, 954669403, -612775698,
        1201765386, -377732593, -906460130, 0,
        -2096529274, 1211247597, -1407315600, 1315723890,
        -67301633, 1443857720, 507358933, 657861945,
        1678381017, 560487590, -778347692, 975451694,
        -1324610969, 261314535, -759894378, -1642357871,
        1333838021, -1570644960, 1767536459, 370938394,
        182621114, -440360918, 1128014560, 487725847,
        185469197, -1376613433, -1188186456, -938205527,
        -2057834215, 1286567175, -1141990947, -39616672,
        -1611202266, -1134791947, -985373125, 878443390,
        1988838185, -590666810, 1756818940, 1673061617,
        -891866660, 272786309, 1075025698, 545572369,
        2105887268, -120407235, 296679730, 1841768865,
        1260232239, -203640272, -334657966, -797457949,
        1814803222, -1716948807, -99511224, 575138148,
        -995558260, 446754879, -665420500, -282971248,
        -947435186, -1042728751, -24327518, 915985419,
```

```
        -811141759, 681933534, 651868046, -1539330625,
        -466863459, 223377554, -1687527476, 1649704518,
        -1024029421, -393160520, 1580087799, -175979601,
        -1096852096, 2087309459, -1452288723, -1278270190,
        1003007129, -1492117379, 1860738147, 2077965243,
        164439672, -194094824, 32283319, -1467789414,
        1709610350, 2125135846, 136428751, -420538904,
        -642062437, -833982666, -722821367, -701910916,
        -1355701070, 824852259, 818324884, -1070226842,
        930369212, -1493400886, -1327460144, 355706840,
        1257309336, -146674470, 243256656, 790073846,
        -1921626666, 1296297904, 1422699085, -538667516,
        -476130891, 457992840, -1195299809, 2135319889,
        77422314, 1560382517, 1945798516, 788204353,
        1521706781, 1385356242, 870912086, 325965383,
        -1936009375, 2050466060, -1906706412, -1981082820,
        -288446169, 901210569, -304014107, 1014646705,
        1503449823, 1062597235, 2031621326, -1082931401,
        -363595827, 1533017514, 350174575, -2038938405,
        -2117423117, 1052338372, 741876788, 1606591296,
        1914052035, 213705253, -1960297399, 1107234197,
        1899603969, -569897805, -1663519516, -1872472383,
        1635502980, 1893020342, 1950903388, 1120974935
};

const int CRijndael::sm_T7[256] =
{
        -1487908364, 1699970625, -1530717673, 1586903591,
        1808481195, 1173430173, 1487645946, 59984867,
        -95084496, 1844882806, 1989249228, 1277555970,
        -671330331, -875051734, 1149249077, -1550863006,
        1514790577, 459744698, 244860394, -1058972162,
        1963115311, -267222708, -1750889146, -104436781,
        1608975247, -1667951214, 2062270317, 1507497298,
        -2094148418, 567498868, 1764313568, -935031095,
        -1989511742, 2037970062, 1047239000, 1910319033,
        1337376481, -1390940024, -1402549984, 984907214,
        1243112415, 830661914, 861968209, 2135253587,
        2011214180, -1367032981, -1608712575, 731183368,
        1750626376, -48656571, 1820824798, -122203525,
        -752637069, 48394827, -1890065633, -1423284651,
        671593195, -1039978571, 2073724613, 145085239,
        -2014171096, -1515052097, 1790575107, -2107839210,
        472615631, -1265457287, -219090169, -492745111,
        -187865638, -1093335547, 1646252340, -24460122,
        1402811438, 1436590835, -516815478, -344611594,
        -331805821, -274055072, -1626972559, 273792366,
        -1963377119, 104699613, 95345982, -1119466010,
        -1917480620, 1560637892, -730921978, 369057872,
```

```
        -81520232, -375925059, 1137477952, -1636341799,
        1119727848, -1954019447, 1530455833, -287606328,
        172466556, 266959938, 516552836, 0,
        -2038232704, -314035669, 1890328081, 1917742170,
        -262898, 945164165, -719438418, 958871085,
        -647755249, -1507760036, 1423022939, 775562294,
        1739656202, -418409641, -1764576018, -1851909221,
        -984645440, 547512796, 1265195639, 437656594,
        -1173691757, 719700128, -532464606, 387781147,
        218828297, -944901493, -1464259146, -1446505442,
        428169201, 122466165, -574886247, 1627235199,
        648017665, -172204942, 1002783846, 2117360635,
        695634755, -958608605, -60246291, -245122844,
        -590686415, -2062531997, 574624663, 287343814,
        612205898, 1039717051, 840019705, -1586641111,
        793451934, 821288114, 1391201670, -472877119,
        376187827, -1181111952, 1224348052, 1679968233,
        -1933268740, 1058709744, 752375421, -1863376333,
        1321699145, -775825096, -1560376118, 188127444,
        -2117097739, -567761542, -1910056265, -1079754835,
        -1645990854, -1844621192, -862229921, 1180849278,
        331544205, -1192718120, -144822727, -1342864701,
        -2134991011, -1820562992, 766078933, 313773861,
        -1724135252, 2108100632, 1668212892, -1149510853,
        2013908262, 418672217, -1224610662, -1700232369,
        1852171925, -427906305, -821550660, -387518699,
        -1680229657, 919489135, 164948639, 2094410160,
        -1297141340, 590424639, -1808742747, 1723872674,
        -1137216434, -895026046, -793714544, -669699161,
        -1739919100, -621329940, 1343127501, -164685935,
        -695372211, -1337113617, 1297403050, 81781910,
        -1243373871, -2011476886, 532201772, 1367295589,
        -368796322, 895287692, 1953757831, 1093597963,
        492483431, -766340389, 1446242576, 1192455638,
        1636604631, 209336225, 344873464, 1015671571,
        669961897, -919226527, -437395172, -1321436601,
        -547775278, 1933530610, -830924780, 935293895,
        -840281097, -1436852227, 1863638845, -611944380,
        -209597777, -1002522264, 875313188, 1080017571,
        -1015933411, 621591778, 1233856572, -1790836979,
        24197544, -1277294580, -459482956, -1047501738,
        -2073986101, -1234119374, 1551124588, 1463996600
};

const int CRijndael::sm_T8[256] =
{
        -190361519, 1097159550, 396673818, 660510266,
        -1418998981, -1656360673, -94852180, -486304949,
        821712160, 1986918061, -864644728, 38544885,
```

-438830001, 718002117, 893681702, 1654886325,
-1319482914, -1172609243, -368142267, -20913827,
796197571, 1290801793, 1184342925, -738605461,
-1889540349, -1835231979, 1836772287, 1381620373,
-1098699308, 1948373848, -529979063, -909622130,
-1031181707, -1904641804, 1480485785, -1183720153,
-514869570, -2001922064, 548169417, -835013507,
-548792221, 439452389, 1362321559, 1400849762,
1685577905, 1806599355, -2120213250, 137073913,
1214797936, 1174215055, -563312748, 2079897426,
1943217067, 1258480242, 529487843, 1437280870,
-349698126, -1245576401, -981755258, 923313619,
679998000, -1079659997, 57326082, 377642221,
-820237430, 2041877159, 133361907, 1776460110,
-621490843, 96392454, 878845905, -1493267772,
777231668, -212492126, -1964953083, -152341084,
-2081670901, 1626319424, 1906247262, 1846563261,
562755902, -586793578, 1040559837, -423803315,
1418573201, -1000536719, 114585348, 1343618912,
-1728371687, -1108764714, 1078185097, -643926169,
-398279248, -1987344377, 425408743, -923870343,
2081048481, 1108339068, -2078357000, 0,
-2138668279, 736970802, 292596766, 1517440620,
251657213, -2059905521, -1361764803, 758720310,
265905162, 1554391400, 1532285339, 908999204,
174567692, 1474760595, -292105548, -1684955621,
-1060810880, -601841055, 2001430874, 303699484,
-1816524062, -1607801408, 585122620, 454499602,
151849742, -1949848078, -1230456531, 514443284,
-249985705, 1963412655, -1713521682, 2137062819,
19308535, 1928707164, 1715193156, -75615141,
1126790795, 600235211, -302225226, -453942344,
836553431, 1669664834, -1759363053, -971956092,
1243905413, -1153566510, -114159186, 698445255,
-1641067747, -1305414692, -2041385971, -1042034569,
-1290376149, 1891211689, -1807156719, -379313593,
-57883480, -264299872, 2100090966, 865136418,
1229899655, 953270745, -895287668, -737462632,
-176042074, 2061379749, -1215420710, -1379949505,
983426092, 2022837584, 1607244650, 2118541908,
-1928084746, -658970480, 972512814, -1011878526,
1568718495, -795640727, -718427793, 621982671,
-1399243832, 410887952, -1671205144, 1002142683,
645401037, 1494807662, -1699282452, 1335535747,
-1787927066, -1671510, -1127282655, 367585007,
-409216582, 1865862730, -1626745622, -1333995991,
-1531793615, 1059270954, -1517014842, -1570324427,
1320957812, -2100648196, -1865371424, -1479011021,
77089521, -321194175, -850391425, -1846137065,

```
        1305906550, -273658557, -1437772596, -1778065436,
        -776608866, 1787304780, 740276417, 1699839814,
        1592394909, -1942659839, -2022411270, 188821243,
        1729977011, -606973294, 274084841, -699985043,
        -681472870, -1593017801, -132870567, 322734571,
        -1457000754, 1640576439, 484830689, 1202797690,
        -757114468, -227328171, 349075736, -952647821,
        -137500077, -39167137, 1030690015, 1155237496,
        -1342996022, 1757691577, 607398968, -1556062270,
        499347990, -500888388, 1011452712, 227885567,
        -1476300487, 213114376, -1260086056, 1455525988,
        -880516741, 850817237, 1817998408, -1202240816
};

const int CRijndael::sm_U1[256] =
{
        0, 235474187, 470948374, 303765277,
        941896748, 908933415, 607530554, 708780849,
        1883793496, 2118214995, 1817866830, 1649639237,
        1215061108, 1181045119, 1417561698, 1517767529,
        -527380304, -291906117, -58537306, -225720403,
        -659233636, -692196969, -995688822, -894438527,
        -1864845080, -1630423581, -1932877058, -2101104651,
        -1459843900, -1493859889, -1259432238, -1159226407,
        -616842373, -718096784, -953573011, -920605594,
        -484470953, -317291940, -15887039, -251357110,
        -1418472669, -1518674392, -1218328267, -1184316354,
        -1822955761, -1654724092, -1891238631, -2125664238,
        1001089995, 899835584, 666464733, 699432150,
        59727847, 226906860, 530400753, 294930682,
        1273168787, 1172967064, 1475418501, 1509430414,
        1942435775, 2110667444, 1876241833, 1641816226,
        -1384747530, -1551933187, -1318815776, -1083344149,
        -1789765158, -1688513327, -1992277044, -2025238841,
        -583137874, -751368027, -1054072904, -819653965,
        -451268222, -351060855, -116905068, -150919521,
        1306967366, 1139781709, 1374988112, 1610459739,
        1975683434, 2076935265, 1775276924, 1742315127,
        1034867998, 866637845, 566021896, 800440835,
        92987698, 193195065, 429456164, 395441711,
        1984812685, 2017778566, 1784663195, 1683407248,
        1315562145, 1080094634, 1383856311, 1551037884,
        101039829, 135050206, 437757123, 337553864,
        1042385657, 807962610, 573804783, 742039012,
        -1763899843, -1730933962, -1966138325, -2067394272,
        -1359400431, -1594867942, -1293211641, -1126030068,
        -426414491, -392404114, -91786125, -191989384,
        -558802359, -793225406, -1029488545, -861254316,
        1106041591, 1340463100, 1576976609, 1408749034,
```

```
	2043211483, 2009195472, 1708848333, 1809054150,
	832877231, 1068351396, 766945465, 599762354,
	159417987, 126454664, 361929877, 463180190,
	-1585706425, -1351284916, -1116860335, -1285087910,
	-1722270101, -1756286112, -2058738563, -1958532746,
	-785096161, -549621996, -853116919, -1020300030,
	-384805325, -417768648, -184398811, -83148498,
	-1697160820, -1797362553, -2033878118, -1999866223,
	-1561111136, -1392879445, -1092530250, -1326955843,
	-358676012, -459930401, -158526526, -125559095,
	-759480840, -592301837, -827774994, -1063245083,
	2051518780, 1951317047, 1716890410, 1750902305,
	1113818384, 1282050075, 1584504582, 1350078989,
	168810852, 67556463, 371049330, 404016761,
	841739592, 1008918595, 775550814, 540080725,
	-325404927, -493635062, -259478249, -25059300,
	-725712083, -625504730, -928212677, -962227152,
	-1663901863, -1831087534, -2134850225, -1899378620,
	-1527321739, -1426069890, -1192955549, -1225917336,
	202008497, 33778362, 270040487, 504459436,
	875451293, 975658646, 675039627, 641025152,
	2084704233, 1917518562, 1615861247, 1851332852,
	1147550661, 1248802510, 1484005843, 1451044056,
	933301370, 967311729, 733156972, 632953703,
	260388950, 25965917, 328671808, 496906059,
	1206477858, 1239443753, 1543208500, 1441952575,
	2144161806, 1908694277, 1675577880, 1842759443,
	-684598070, -650587711, -886847780, -987051049,
	-283776794, -518199827, -217582864, -49348613,
	-1485196142, -1452230247, -1150570876, -1251826801,
	-1621262146, -1856729675, -2091935064, -1924753501
};

const int CRijndael::sm_U2[256] =
{
	0, 185469197, 370938394, 487725847,
	741876788, 657861945, 975451694, 824852259,
	1483753576, 1400783205, 1315723890, 1164071807,
	1950903388, 2135319889, 1649704518, 1767536459,
	-1327460144, -1141990947, -1493400886, -1376613433,
	-1663519516, -1747534359, -1966823682, -2117423117,
	-393160520, -476130891, -24327518, -175979601,
	-995558260, -811141759, -759894378, -642062437,
	2077965243, 1893020342, 1841768865, 1724457132,
	1474502543, 1559041666, 1107234197, 1257309336,
	598438867, 681933534, 901210569, 1052338372,
	261314535, 77422314, 428819965, 310463728,
	-885281941, -1070226842, -584599183, -701910916,
	-419197089, -334657966, -249586363, -99511224,
```

-1823743229, -1740248562, -2057834215, -1906706412,
-1082931401, -1266823622, -1452288723, -1570644960,
-156404115, -39616672, -525245321, -339776134,
-627748263, -778347692, -863420349, -947435186,
-1361232379, -1512884472, -1195299809, -1278270190,
-2098914767, -1981082820, -1795618773, -1611202266,
1179510461, 1296297904, 1347548327, 1533017514,
1786102409, 1635502980, 2087309459, 2003294622,
507358933, 355706840, 136428751, 53458370,
839224033, 957055980, 605657339, 790073846,
-1921626666, -2038938405, -1687527476, -1872472383,
-1588696606, -1438621457, -1219331080, -1134791947,
-721025602, -569897805, -1021700188, -938205527,
-113368694, -231724921, -282971248, -466863459,
1033297158, 915985419, 730517276, 545572369,
296679730, 446754879, 129166120, 213705253,
1709610350, 1860738147, 1945798516, 2029293177,
1239331162, 1120974935, 1606591296, 1422699085,
-146674470, -61872681, -513933632, -363595827,
-612775698, -797457949, -848962828, -966011911,
-1355701070, -1539330625, -1188186456, -1306280027,
-2096529274, -2012771957, -1793748324, -1642357871,
1201765386, 1286567175, 1371368976, 1521706781,
1805211710, 1620529459, 2105887268, 1988838185,
533804130, 350174575, 164439672, 46346101,
870912086, 954669403, 636813900, 788204353,
-1936009375, -2020286868, -1702443653, -1853305738,
-1599933611, -1414727080, -1229004465, -1112479678,
-722821367, -538667516, -1024029421, -906460130,
-120407235, -203640272, -288446169, -440360918,
1014646705, 930369212, 711349675, 560487590,
272786309, 457992840, 106852767, 223377554,
1678381017, 1862534868, 1914052035, 2031621326,
1211247597, 1128014560, 1580087799, 1428173050,
32283319, 182621114, 401639597, 486441376,
768917123, 651868046, 1003007129, 818324884,
1503449823, 1385356242, 1333838021, 1150208456,
1973745387, 2125135846, 1673061617, 1756818940,
-1324610969, -1174273174, -1492117379, -1407315600,
-1657524653, -1774573730, -1960297399, -2144979644,
-377732593, -495826174, -10465259, -194094824,
-985373125, -833982666, -749177823, -665420500,
2050466060, 1899603969, 1814803222, 1730525723,
1443857720, 1560382517, 1075025698, 1260232239,
575138148, 692707433, 878443390, 1062597235,
243256656, 91341917, 409198410, 325965383,
-891866660, -1042728751, -590666810, -674944309,
-420538904, -304014107, -252508174, -67301633,
-1834518092, -1716948807, -2068091986, -1883938141,

```
	-1096852096, -1248766835, -1467789414, -1551022441,
};

const int CRijndael::sm_U3[256] =
{
	0, 218828297, 437656594, 387781147,
	875313188, 958871085, 775562294, 590424639,
	1750626376, 1699970625, 1917742170, 2135253587,
	1551124588, 1367295589, 1180849278, 1265195639,
	-793714544, -574886247, -895026046, -944901493,
	-459482956, -375925059, -24460122, -209597777,
	-1192718120, -1243373871, -1560376118, -1342864701,
	-1933268740, -2117097739, -1764576018, -1680229657,
	-1149510853, -1234119374, -1586641111, -1402549984,
	-1890065633, -2107839210, -1790836979, -1739919100,
	-752637069, -567761542, -919226527, -1002522264,
	-418409641, -368796322, -48656571, -267222708,
	1808481195, 1723872674, 1910319033, 2094410160,
	1608975247, 1391201670, 1173430173, 1224348052,
	59984867, 244860394, 428169201, 344873464,
	935293895, 984907214, 766078933, 547512796,
	1844882806, 1627235199, 2011214180, 2062270317,
	1507497298, 1423022939, 1137477952, 1321699145,
	95345982, 145085239, 532201772, 313773861,
	830661914, 1015671571, 731183368, 648017665,
	-1119466010, -1337113617, -1487908364, -1436852227,
	-1989511742, -2073986101, -1820562992, -1636341799,
	-719438418, -669699161, -821550660, -1039978571,
	-516815478, -331805821, -81520232, -164685935,
	-695372211, -611944380, -862229921, -1047501738,
	-492745111, -274055072, -122203525, -172204942,
	-1093335547, -1277294580, -1530717673, -1446505442,
	-1963377119, -2014171096, -1863376333, -1645990854,
	104699613, 188127444, 472615631, 287343814,
	840019705, 1058709744, 671593195, 621591778,
	1852171925, 1668212892, 1953757831, 2037970062,
	1514790577, 1463996600, 1080017571, 1297403050,
	-621329940, -671330331, -1058972162, -840281097,
	-287606328, -472877119, -187865638, -104436781,
	-1297141340, -1079754835, -1464259146, -1515052097,
	-2038232704, -1954019447, -1667951214, -1851909221,
	172466556, 122466165, 273792366, 492483431,
	1047239000, 861968209, 612205898, 695634755,
	1646252340, 1863638845, 2013908262, 1963115311,
	1446242576, 1530455833, 1277555970, 1093597963,
	1636604631, 1820824798, 2073724613, 1989249228,
	1436590835, 1487645946, 1337376481, 1119727848,
	164948639, 81781910, 331544205, 516552836,
	1039717051, 821288114, 669961897, 719700128,
```

```
	-1321436601, -1137216434, -1423284651, -1507760036,
	-2062531997, -2011476886, -1626972559, -1844621192,
	-647755249, -730921978, -1015933411, -830924780,
	-314035669, -532464606, -144822727, -95084496,
	-1224610662, -1173691757, -1390940024, -1608712575,
	-2094148418, -1910056265, -1724135252, -1808742747,
	-547775278, -766340389, -984645440, -935031095,
	-344611594, -427906305, -245122844, -60246291,
	1739656202, 1790575107, 2108100632, 1890328081,
	1402811438, 1586903591, 1233856572, 1149249077,
	266959938, 48394827, 369057872, 418672217,
	1002783846, 919489135, 567498868, 752375421,
	209336225, 24197544, 376187827, 459744698,
	945164165, 895287692, 574624663, 793451934,
	1679968233, 1764313568, 2117360635, 1933530610,
	1343127501, 1560637892, 1243112415, 1192455638,
	-590686415, -775825096, -958608605, -875051734,
	-387518699, -437395172, -219090169, -262898,
	-1265457287, -1181111952, -1367032981, -1550863006,
	-2134991011, -1917480620, -1700232369, -1750889146
};

const int CRijndael::sm_U4[256] =
{
	0, 151849742, 303699484, 454499602,
	607398968, 758720310, 908999204, 1059270954,
	1214797936, 1097159550, 1517440620, 1400849762,
	1817998408, 1699839814, 2118541908, 2001430874,
	-1865371424, -1713521682, -2100648196, -1949848078,
	-1260086056, -1108764714, -1493267772, -1342996022,
	-658970480, -776608866, -895287668, -1011878526,
	-57883480, -176042074, -292105548, -409216582,
	1002142683, 850817237, 698445255, 548169417,
	529487843, 377642221, 227885567, 77089521,
	1943217067, 2061379749, 1640576439, 1757691577,
	1474760595, 1592394909, 1174215055, 1290801793,
	-1418998981, -1570324427, -1183720153, -1333995991,
	-1889540349, -2041385971, -1656360673, -1807156719,
	-486304949, -368142267, -249985705, -132870567,
	-952647821, -835013507, -718427793, -601841055,
	1986918061, 2137062819, 1685577905, 1836772287,
	1381620373, 1532285339, 1078185097, 1229899655,
	1040559837, 923313619, 740276417, 621982671,
	439452389, 322734571, 137073913, 19308535,
	-423803315, -273658557, -190361519, -39167137,
	-1031181707, -880516741, -795640727, -643926169,
	-1361764803, -1479011021, -1127282655, -1245576401,
	-1964953083, -2081670901, -1728371687, -1846137065,
	1305906550, 1155237496, 1607244650, 1455525988,
```

```
        1776460110, 1626319424, 2079897426, 1928707164,
        96392454, 213114376, 396673818, 514443284,
        562755902, 679998000, 865136418, 983426092,
        -586793578, -737462632, -820237430, -971956092,
        -114159186, -264299872, -349698126, -500888388,
        -1787927066, -1671205144, -2022411270, -1904641804,
        -1319482914, -1202240816, -1556062270, -1437772596,
        -321194175, -438830001, -20913827, -137500077,
        -923870343, -1042034569, -621490843, -738605461,
        -1531793615, -1379949505, -1230456531, -1079659997,
        -2138668279, -1987344377, -1835231979, -1684955621,
        2081048481, 1963412655, 1846563261, 1729977011,
        1480485785, 1362321559, 1243905413, 1126790795,
        878845905, 1030690015, 645401037, 796197571,
        274084841, 425408743, 38544885, 188821243,
        -681472870, -563312748, -981755258, -864644728,
        -212492126, -94852180, -514869570, -398279248,
        -1626745622, -1778065436, -1928084746, -2078357000,
        -1153566510, -1305414692, -1457000754, -1607801408,
        1202797690, 1320957812, 1437280870, 1554391400,
        1669664834, 1787304780, 1906247262, 2022837584,
        265905162, 114585348, 499347990, 349075736,
        736970802, 585122620, 972512814, 821712160,
        -1699282452, -1816524062, -2001922064, -2120213250,
        -1098699308, -1215420710, -1399243832, -1517014842,
        -757114468, -606973294, -1060810880, -909622130,
        -152341084, -1671510, -453942344, -302225226,
        174567692, 57326082, 410887952, 292596766,
        777231668, 660510266, 1011452712, 893681702,
        1108339068, 1258480242, 1343618912, 1494807662,
        1715193156, 1865862730, 1948373848, 2100090966,
        -1593017801, -1476300487, -1290376149, -1172609243,
        -2059905521, -1942659839, -1759363053, -1641067747,
        -379313593, -529979063, -75615141, -227328171,
        -850391425, -1000536719, -548792221, -699985043,
        836553431, 953270745, 600235211, 718002117,
        367585007, 484830689, 133361907, 251657213,
        2041877159, 1891211689, 1806599355, 1654886325,
        1568718495, 1418573201, 1335535747, 1184342925
};

const char CRijndael::sm_rcon[30] =
{
        1, 2, 4, 8, 16, 32,
        64, -128, 27, 54, 108, -40,
        -85, 77, -102, 47, 94, -68,
        99, -58, -105, 53, 106, -44,
        -77, 125, -6, -17, -59, -111
};
```

```
const int CRijndael::sm_shifts[3][4][2] =
{
	{ {0, 0}, {1, 3}, {2, 2}, {3, 1} },
	{ {0, 0}, {1, 5}, {2, 4}, {3, 3} },
	{ {0, 0}, {1, 7}, {3, 5}, {4, 4} }
};

//Error Messages
char const* CRijndael::sm_szErrorMsg1 = "Object not Initialized";
char const* CRijndael::sm_szErrorMsg2 = "Data not multiple of Block Size";

//Null chain
char const* CRijndael::sm_chain0 =
"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0";

//CONSTRUCTOR
CRijndael::CRijndael() : m_bKeyInit(false)
{
}

//DESTRUCTOR
CRijndael::~CRijndael()
{
}

//Expand a user-supplied key material into a session key.
// key        - The 128/192/256-bit user-key to use.
// chain      - initial chain block for CBC and CFB modes.
// keylength  - 16, 24 or 32 bytes
// blockSize  - The block size in bytes of this Rijndael (16, 24 or 32 bytes).
void CRijndael::MakeKey(char const* key, char const* chain, int keylength, int
blockSize)
{
	if(NULL == key)
		throw exception("Empty key");
	if(!(16==keylength || 24==keylength || 32==keylength))
		throw exception("Incorrect key length");
	if(!(16==blockSize || 24==blockSize || 32==blockSize))
		throw exception("Incorrect block length");
	m_keylength = keylength;
	m_blockSize = blockSize;
	//Initialize the chain
	memcpy(m_chain0, chain, m_blockSize);
	memcpy(m_chain, chain, m_blockSize);
	//Calculate Number of Rounds
	switch(m_keylength)
	{
		case 16:
```

```
                m_iROUNDS = (m_blockSize == 16) ? 10 : (m_blockSize ==
24 ? 12 : 14);
                break;

        case 24:
                m_iROUNDS = (m_blockSize != 32) ? 12 : 14;
                break;

        default: // 32 bytes = 256 bits
                m_iROUNDS = 14;
    }
    int BC = m_blockSize / 4;
    int i, j;
    for(i=0; i<=m_iROUNDS; i++)
    {
        for(j=0; j<BC; j++)
            m_Ke[i][j] = 0;
    }
    for(i=0; i<=m_iROUNDS; i++)
    {
        for(j=0; j<BC; j++)
            m_Kd[i][j] = 0;
    }
    int ROUND_KEY_COUNT = (m_iROUNDS + 1) * BC;
    int KC = m_keylength/4;
    //Copy user material bytes into temporary ints
    int* pi = tk;
    char const* pc = key;
    for(i=0; i<KC; i++)
    {
        *pi = (unsigned char)*(pc++) << 24;
        *pi |= (unsigned char)*(pc++) << 16;
        *pi |= (unsigned char)*(pc++) << 8;
        *(pi++) |= (unsigned char)*(pc++);
    }
    //Copy values into round key arrays
    int t = 0;
    for(j=0; (j<KC)&&(t<ROUND_KEY_COUNT); j++,t++)
    {
        m_Ke[t/BC][t%BC] = tk[j];
        m_Kd[m_iROUNDS - (t/BC)][t%BC] = tk[j];
    }
    int tt, rconpointer = 0;
    while(t < ROUND_KEY_COUNT)
    {
        //Extrapolate using phi (the round key evolution function)
        tt = tk[KC-1];
        tk[0] ^= (sm_S[(tt >> 16) & 0xFF] & 0xFF) << 24 ^
            (sm_S[(tt >>  8) & 0xFF] & 0xFF) << 16 ^
```

```
                        (sm_S[ tt & 0xFF] & 0xFF) <<  8 ^
                        (sm_S[(tt >> 24) & 0xFF] & 0xFF) ^
                        (sm_rcon[rconpointer++]  & 0xFF) << 24;
                if(KC != 8)
                        for(i=1, j=0; i<KC;)
                                tk[i++] ^= tk[j++];
                else
                {
                        for(i=1, j=0; i<KC/2; )
                                tk[i++] ^= tk[j++];
                        tt = tk[KC/2-1];
                        tk[KC/2] ^= (sm_S[ tt & 0xFF] & 0xFF) ^
                                (sm_S[(tt >>  8) & 0xFF] & 0xFF) <<  8 ^
                                (sm_S[(tt >> 16) & 0xFF] & 0xFF) << 16 ^
                                (sm_S[(tt >> 24) & 0xFF] & 0xFF) << 24;
                        for(j = KC/2, i=j+1; i<KC; )
                                tk[i++] ^= tk[j++];
                }
                //Copy values into round key arrays
                for(j=0; (j<KC) && (t<ROUND_KEY_COUNT); j++, t++)
                {
                        m_Ke[t/BC][t%BC] = tk[j];
                        m_Kd[m_iROUNDS - (t/BC)][t%BC] = tk[j];
                }
        }
        //Inverse MixColumn where needed
        for(int r=1; r<m_iROUNDS; r++)
                for(j=0; j<BC; j++)
                {
                        tt = m_Kd[r][j];
                        m_Kd[r][j] = sm_U1[(tt >> 24) & 0xFF] ^
                                sm_U2[(tt >> 16) & 0xFF] ^
                                sm_U3[(tt >>  8) & 0xFF] ^
                                sm_U4[tt & 0xFF];
                }
        m_bKeyInit = true;
}

//Convenience method to encrypt exactly one block of plaintext, assuming
//Rijndael's default block size (128-bit).
// in         - The plaintext
// result     - The ciphertext generated from a plaintext using the key
void CRijndael::DefEncryptBlock(char const* in, char* result)
{
        if(false==m_bKeyInit)
                throw exception(sm_szErrorMsg1);
        int* Ker = m_Ke[0];
        int t0 = ((unsigned char)*(in++) << 24);
        t0 |= ((unsigned char)*(in++) << 16);
```

```
t0 |= ((unsigned char)*(in++) << 8);
(t0 |= (unsigned char)*(in++)) ^= Ker[0];
int t1 = ((unsigned char)*(in++) << 24);
t1 |= ((unsigned char)*(in++) << 16);
t1 |= ((unsigned char)*(in++) << 8);
(t1 |= (unsigned char)*(in++)) ^= Ker[1];
int t2 = ((unsigned char)*(in++) << 24);
t2 |= ((unsigned char)*(in++) << 16);
t2 |= ((unsigned char)*(in++) << 8);
(t2 |= (unsigned char)*(in++)) ^= Ker[2];
int t3 = ((unsigned char)*(in++) << 24);
t3 |= ((unsigned char)*(in++) << 16);
t3 |= ((unsigned char)*(in++) << 8);
(t3 |= (unsigned char)*(in++)) ^= Ker[3];
int a0, a1, a2, a3;
//Apply Round Transforms
for (int r = 1; r < m_iROUNDS; r++)
{
        Ker = m_Ke[r];
        a0 = (sm_T1[(t0 >> 24) & 0xFF] ^
                sm_T2[(t1 >> 16) & 0xFF] ^
                sm_T3[(t2 >>  8) & 0xFF] ^
                sm_T4[t3 & 0xFF]) ^ Ker[0];
        a1 = (sm_T1[(t1 >> 24) & 0xFF] ^
                sm_T2[(t2 >> 16) & 0xFF] ^
                sm_T3[(t3 >>  8) & 0xFF] ^
                sm_T4[t0 & 0xFF]) ^ Ker[1];
        a2 = (sm_T1[(t2 >> 24) & 0xFF] ^
                sm_T2[(t3 >> 16) & 0xFF] ^
                sm_T3[(t0 >>  8) & 0xFF] ^
                sm_T4[t1 & 0xFF]) ^ Ker[2];
        a3 = (sm_T1[(t3 >> 24) & 0xFF] ^
                sm_T2[(t0 >> 16) & 0xFF] ^
                sm_T3[(t1 >>  8) & 0xFF] ^
                sm_T4[t2 & 0xFF]) ^ Ker[3];
        t0 = a0;
        t1 = a1;
        t2 = a2;
        t3 = a3;
}
//Last Round is special
Ker = m_Ke[m_iROUNDS];
int tt = Ker[0];
result[0] = sm_S[(t0 >> 24) & 0xFF] ^ (tt >> 24);
result[1] = sm_S[(t1 >> 16) & 0xFF] ^ (tt >> 16);
result[2] = sm_S[(t2 >>  8) & 0xFF] ^ (tt >>  8);
result[3] = sm_S[t3 & 0xFF] ^ tt;
tt = Ker[1];
result[4] = sm_S[(t1 >> 24) & 0xFF] ^ (tt >> 24);
```

```
	result[5] = sm_S[(t2 >> 16) & 0xFF] ^ (tt >> 16);
	result[6] = sm_S[(t3 >>  8) & 0xFF] ^ (tt >>  8);
	result[7] = sm_S[t0 & 0xFF] ^ tt;
	tt = Ker[2];
	result[8] = sm_S[(t2 >> 24) & 0xFF] ^ (tt >> 24);
	result[9] = sm_S[(t3 >> 16) & 0xFF] ^ (tt >> 16);
	result[10] = sm_S[(t0 >>  8) & 0xFF] ^ (tt >>  8);
	result[11] = sm_S[t1 & 0xFF] ^ tt;
	tt = Ker[3];
	result[12] = sm_S[(t3 >> 24) & 0xFF] ^ (tt >> 24);
	result[13] = sm_S[(t0 >> 16) & 0xFF] ^ (tt >> 16);
	result[14] = sm_S[(t1 >>  8) & 0xFF] ^ (tt >>  8);
	result[15] = sm_S[t2 & 0xFF] ^ tt;
}

//Convenience method to decrypt exactly one block of plaintext, assuming
//Rijndael's default block size (128-bit).
// in         - The ciphertext.
// result     - The plaintext generated from a ciphertext using the session key.
void CRijndael::DefDecryptBlock(char const* in, char* result)
{
	if(false==m_bKeyInit)
		throw exception(sm_szErrorMsg1);
	int* Kdr = m_Kd[0];
	int t0 = ((unsigned char)*(in++) << 24);
	t0 = t0 | ((unsigned char)*(in++) << 16);
	t0 |= ((unsigned char)*(in++) << 8);
	(t0 |= (unsigned char)*(in++)) ^= Kdr[0];
	int t1 = ((unsigned char)*(in++) << 24);
	t1 |= ((unsigned char)*(in++) << 16);
	t1 |= ((unsigned char)*(in++) << 8);
	(t1 |= (unsigned char)*(in++)) ^= Kdr[1];
	int t2 = ((unsigned char)*(in++) << 24);
	t2 |= ((unsigned char)*(in++) << 16);
	t2 |= ((unsigned char)*(in++) << 8);
	(t2 |= (unsigned char)*(in++)) ^= Kdr[2];
	int t3 = ((unsigned char)*(in++) << 24);
	t3 |= ((unsigned char)*(in++) << 16);
	t3 |= ((unsigned char)*(in++) << 8);
	(t3 |= (unsigned char)*(in++)) ^= Kdr[3];
	int a0, a1, a2, a3;
	for(int r = 1; r < m_iROUNDS; r++) // apply round transforms
	{
		Kdr = m_Kd[r];
		a0 = (sm_T5[(t0 >> 24) & 0xFF] ^
			sm_T6[(t3 >> 16) & 0xFF] ^
			sm_T7[(t2 >>  8) & 0xFF] ^
			sm_T8[ t1        & 0xFF] ) ^ Kdr[0];
		a1 = (sm_T5[(t1 >> 24) & 0xFF] ^
```

```
                        sm_T6[(t0 >> 16) & 0xFF] ^
                        sm_T7[(t3 >>  8) & 0xFF] ^
                        sm_T8[ t2        & 0xFF] ) ^ Kdr[1];
                a2 = (sm_T5[(t2 >> 24) & 0xFF] ^
                        sm_T6[(t1 >> 16) & 0xFF] ^
                        sm_T7[(t0 >>  8) & 0xFF] ^
                        sm_T8[ t3        & 0xFF] ) ^ Kdr[2];
                a3 = (sm_T5[(t3 >> 24) & 0xFF] ^
                        sm_T6[(t2 >> 16) & 0xFF] ^
                        sm_T7[(t1 >>  8) & 0xFF] ^
                        sm_T8[ t0        & 0xFF] ) ^ Kdr[3];
                t0 = a0;
                t1 = a1;
                t2 = a2;
                t3 = a3;
        }
        //Last Round is special
        Kdr = m_Kd[m_iROUNDS];
        int tt = Kdr[0];
        result[ 0] = sm_Si[(t0 >> 24) & 0xFF] ^ (tt >> 24);
        result[ 1] = sm_Si[(t3 >> 16) & 0xFF] ^ (tt >> 16);
        result[ 2] = sm_Si[(t2 >>  8) & 0xFF] ^ (tt >>  8);
        result[ 3] = sm_Si[ t1 & 0xFF] ^ tt;
        tt = Kdr[1];
        result[ 4] = sm_Si[(t1 >> 24) & 0xFF] ^ (tt >> 24);
        result[ 5] = sm_Si[(t0 >> 16) & 0xFF] ^ (tt >> 16);
        result[ 6] = sm_Si[(t3 >>  8) & 0xFF] ^ (tt >>  8);
        result[ 7] = sm_Si[ t2 & 0xFF] ^ tt;
        tt = Kdr[2];
        result[ 8] = sm_Si[(t2 >> 24) & 0xFF] ^ (tt >> 24);
        result[ 9] = sm_Si[(t1 >> 16) & 0xFF] ^ (tt >> 16);
        result[10] = sm_Si[(t0 >>  8) & 0xFF] ^ (tt >>  8);
        result[11] = sm_Si[ t3 & 0xFF] ^ tt;
        tt = Kdr[3];
        result[12] = sm_Si[(t3 >> 24) & 0xFF] ^ (tt >> 24);
        result[13] = sm_Si[(t2 >> 16) & 0xFF] ^ (tt >> 16);
        result[14] = sm_Si[(t1 >>  8) & 0xFF] ^ (tt >>  8);
        result[15] = sm_Si[ t0 & 0xFF] ^ tt;
}

//Encrypt exactly one block of plaintext.
// in           - The plaintext.
// result       - The ciphertext generated from a plaintext using the key.
void CRijndael::EncryptBlock(char const* in, char* result)
{
        if(false==m_bKeyInit)
                throw exception(sm_szErrorMsg1);
   if(DEFAULT_BLOCK_SIZE == m_blockSize)
        {
```

```
		DefEncryptBlock(in, result);
		return;
	}
	int BC = m_blockSize / 4;
	int SC = (BC == 4) ? 0 : (BC == 6 ? 1 : 2);
	int s1 = sm_shifts[SC][1][0];
	int s2 = sm_shifts[SC][2][0];
	int s3 = sm_shifts[SC][3][0];
	//Temporary Work Arrays
	int i;
	int tt;
	int* pi = t;
	for(i=0; i<BC; i++)
	{
		*pi = ((unsigned char)*(in++) << 24);
		*pi |= ((unsigned char)*(in++) << 16);
		*pi |= ((unsigned char)*(in++) << 8);
		(*(pi++) |= (unsigned char)*(in++)) ^= m_Ke[0][i];
	}
	//Apply Round Transforms
	for(int r=1; r<m_iROUNDS; r++)
	{
		for(i=0; i<BC; i++)
			a[i] = (sm_T1[(t[i] >> 24) & 0xFF] ^
				sm_T2[(t[(i + s1) % BC] >> 16) & 0xFF] ^
				sm_T3[(t[(i + s2) % BC] >>  8) & 0xFF] ^
				sm_T4[ t[(i + s3) % BC] & 0xFF] ) ^ m_Ke[r][i];
		memcpy(t, a, 4*BC);
	}
	int j;
	//Last Round is Special
	for(i=0,j=0; i<BC; i++)
	{
		tt = m_Ke[m_iROUNDS][i];
		result[j++] = sm_S[(t[i] >> 24) & 0xFF] ^ (tt >> 24);
		result[j++] = sm_S[(t[(i + s1) % BC] >> 16) & 0xFF] ^ (tt >> 16);
		result[j++] = sm_S[(t[(i + s2) % BC] >>  8) & 0xFF] ^ (tt >>  8);
		result[j++] = sm_S[ t[(i + s3) % BC] & 0xFF] ^ tt;
	}
}

//Decrypt exactly one block of ciphertext.
// in		- The ciphertext.
// result	- The plaintext generated from a ciphertext using the session key.
void CRijndael::DecryptBlock(char const* in, char* result)
{
	if(false==m_bKeyInit)
		throw exception(sm_szErrorMsg1);
	if(DEFAULT_BLOCK_SIZE == m_blockSize)
```

```
	{
		DefDecryptBlock(in, result);
		return;
	}
	int BC = m_blockSize / 4;
	int SC = BC == 4 ? 0 : (BC == 6 ? 1 : 2);
	int s1 = sm_shifts[SC][1][1];
	int s2 = sm_shifts[SC][2][1];
	int s3 = sm_shifts[SC][3][1];
	//Temporary Work Arrays
	int i;
	int tt;
	int* pi = t;
	for(i=0; i<BC; i++)
	{
		*pi = ((unsigned char)*(in++) << 24);
		*pi |= ((unsigned char)*(in++) << 16);
		*pi |= ((unsigned char)*(in++) << 8);
		(*(pi++) |= (unsigned char)*(in++)) ^= m_Kd[0][i];
	}
	//Apply Round Transforms
	for(int r=1; r<m_iROUNDS; r++)
	{
		for(i=0; i<BC; i++)
			a[i] = (sm_T5[(t[i] >> 24) & 0xFF] ^
				sm_T6[(t[(i + s1) % BC] >> 16) & 0xFF] ^
				sm_T7[(t[(i + s2) % BC] >>  8) & 0xFF] ^
				sm_T8[ t[(i + s3) % BC] & 0xFF]) ^ m_Kd[r][i];
		memcpy(t, a, 4*BC);
	}
	int j;
	//Last Round is Special
	for(i=0,j=0; i<BC; i++)
	{
		tt = m_Kd[m_iROUNDS][i];
		result[j++] = sm_Si[(t[i] >> 24) & 0xFF] ^ (tt >> 24);
		result[j++] = sm_Si[(t[(i + s1) % BC] >> 16) & 0xFF] ^ (tt >> 16);
		result[j++] = sm_Si[(t[(i + s2) % BC] >>  8) & 0xFF] ^ (tt >>  8);
		result[j++] = sm_Si[ t[(i + s3) % BC] & 0xFF] ^ tt;
	}
}

void CRijndael::Encrypt(char const* in, char* result, size_t n, int iMode)
{
	if(false==m_bKeyInit)
		throw exception(sm_szErrorMsg1);
	//n should be > 0 and multiple of m_blockSize
	if(0==n || n%m_blockSize!=0)
		throw exception(sm_szErrorMsg2);
```

```
	int i;
	char const* pin;
	char* presult;
	if(CBC == iMode) //CBC mode, using the Chain
	{
		for(i=0,pin=in,presult=result; i<n/m_blockSize; i++)
		{
			Xor(m_chain, pin);
			EncryptBlock(m_chain, presult);
			memcpy(m_chain, presult, m_blockSize);
			pin += m_blockSize;
			presult += m_blockSize;
		}
	}
	else if(CFB == iMode) //CFB mode, using the Chain
	{
		for(i=0,pin=in,presult=result; i<n/m_blockSize; i++)
		{
			EncryptBlock(m_chain, presult);
			Xor(presult, pin);
			memcpy(m_chain, presult, m_blockSize);
			pin += m_blockSize;
			presult += m_blockSize;
		}
	}
	else //ECB mode, not using the Chain
	{
		for(i=0,pin=in,presult=result; i<n/m_blockSize; i++)
		{
			EncryptBlock(pin, presult);
			pin += m_blockSize;
			presult += m_blockSize;
		}
	}
}

void CRijndael::Decrypt(char const* in, char* result, size_t n, int iMode)
{
	if(false==m_bKeyInit)
		throw exception(sm_szErrorMsg1);
	//n should be > 0 and multiple of m_blockSize
	if(0==n || n%m_blockSize!=0)
		throw exception(sm_szErrorMsg2);
	int i;
	char const* pin;
	char* presult;
	if(CBC == iMode) //CBC mode, using the Chain
	{
		for(i=0,pin=in,presult=result; i<n/m_blockSize; i++)
```

```
		{
			DecryptBlock(pin, presult);
			Xor(presult, m_chain);
			memcpy(m_chain, pin, m_blockSize);

			pin += m_blockSize;
			presult += m_blockSize;
		}
	}
	else if(CFB == iMode) //CFB mode, using the Chain, not using Decrypt()
	{
		for(i=0,pin=in,presult=result; i<n/m_blockSize; i++)
		{
			EncryptBlock(m_chain, presult);
			//memcpy(presult, pin, m_blockSize);
			Xor(presult, pin);
			memcpy(m_chain, pin, m_blockSize);
			pin += m_blockSize;
			presult += m_blockSize;
		}
	}
	else //ECB mode, not using the Chain
	{
		for(i=0,pin=in,presult=result; i<n/m_blockSize; i++)
		{
			DecryptBlock(pin, presult);
			pin += m_blockSize;
			presult += m_blockSize;
		}
	}
}
```

```
//Test.cpp

#include "Rijndael.h"
#include <iostream>

using namespace std;

//Function to convert unsigned char to string of length 2
void Char2Hex(unsigned char ch, char* szHex)
{
	unsigned char byte[2];
	byte[0] = ch/16;
	byte[1] = ch%16;
	for(int i=0; i<2; i++)
	{
		if(byte[i] >= 0 && byte[i] <= 9)
```

```
                szHex[i] = '0' + byte[i];
            else
                szHex[i] = 'A' + byte[i] - 10;
        }
        szHex[2] = 0;
}

//Function to convert string of length 2 to unsigned char
void Hex2Char(char const* szHex, unsigned char& rch)
{
        rch = 0;
        for(int i=0; i<2; i++)
        {
            if(*(szHex + i) >='0' && *(szHex + i) <= '9')
                rch = (rch << 4) + (*(szHex + i) - '0');
            else if(*(szHex + i) >='A' && *(szHex + i) <= 'F')
                rch = (rch << 4) + (*(szHex + i) - 'A' + 10);
            else
                break;
        }
}

//Function to convert string of unsigned chars to string of chars
void CharStr2HexStr(unsigned char const* pucCharStr, char* pszHexStr, int iSize)
{
        int i;
        char szHex[3];
        pszHexStr[0] = 0;
        for(i=0; i<iSize; i++)
        {
            Char2Hex(pucCharStr[i], szHex);
            strcat(pszHexStr, szHex);
        }
}

//Function to convert string of chars to string of unsigned chars
void HexStr2CharStr(char const* pszHexStr, unsigned char* pucCharStr, int iSize)
{
        int i;
        unsigned char ch;
        for(i=0; i<iSize; i++)
        {
            Hex2Char(pszHexStr+2*i, ch);
            pucCharStr[i] = ch;
        }
}

void main()
{
```

```
	try
	{
		char szHex[33];
		//One block testing
		CRijndael oRijndael;
		oRijndael.MakeKey("abcdefghabcdefgh",
"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 16, 16);
		char szDataIn[] = "aaaaaaaabbbbbbbb";
		char szDataOut[17] = "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0";
		oRijndael.EncryptBlock(szDataIn, szDataOut);
		CharStr2HexStr((unsigned char*)szDataIn, szHex, 16);
		cout << szHex << endl;
		CharStr2HexStr((unsigned char*)szDataOut, szHex, 16);
		cout << szHex << endl;
		memset(szDataIn, 0, 16);
		oRijndael.DecryptBlock(szDataOut, szDataIn);
		CharStr2HexStr((unsigned char*)szDataIn, szHex, 16);
		cout << szHex << endl;
	}
	catch(exception& roException)
	{
		cout << roException.what() << endl;
	}
}
```

ÖZGEÇMİŞ

Zekeriya ALTUN, 10.05.1971 de Batman'da doğdu. İlk, orta ve lise eğitimini Antalya'da tamamladı. 1988 yılında Antalya Endüstri Meslek Lisesi, Elektronik Bölümünden mezun oldu. 1989 yılında başladığı İTÜ SMF Elektrik Elektronik Mühendisliği bölümünü 1996 yılında bitirdi. 1996 - 1999 yılları arasında Sakarya'daki Tekofen Elektrik Tic. Ltd. Şti.nde mühendis olarak çalıştı. Bu süre içerisinde şirketin değişik şantiyelerinde şantiye şefliği yaptı. 2001 yılında 4 arkadaşıyla birlikte Sakarya ilindeki Penta Sistem Mühendislik Tic. Ltd. Şti. firmasını kurdular. Halen aynı firmanın ortağı olarak serbest piyasada faaliyet göstermektedir.