GÜVENİLİR, HATA DÜZELTMELİ VE DİJİTAL İMZALI PROTOKOL GELİŞTİRME

A. Çağrı BÖLÜCEK

YÜKSEK LİSANS TEZİ
ELEKTRİK – ELEKTRONİK MÜHENDİSLİĞİ

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KASIM 2005
ANKARA

GÜVENİLİR, HATA DÜZELTMELİ VE DİJİTAL İMZALI PROTOKOL GELİŞTİRME

(Yüksek Lisans Tezi)

A.Çağrı BÖLÜCEK

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
Kasım 2005

ÖZET

Teknolojik gelişmeler her zaman birtakım sorun ve ihtiyaçları da beraberinde getirmiştir. Bugün, evlerde kullanılan elektronik cihazlardan endüstriyel veya askeri alanda kullanılan cihazlara kadar birçok anlamda kullanımı kolaylaştırıcı çözümler sunulmaktadır.

Son zamanlarda oldukça popüler olan akıllı ev sistemleri ve diğer benzeri otomasyon projeleri bunlara örnek gösterilebilir. Bu tarzda çözümler geliştirilirken iletişimin, kötü niyetli şahıslardan saklanarak, gizlilik içinde güvenle sürdürülebilmesi için karşılaşılabilecek problemlerin hesaba katılarak geliştirildiği bir protokole ihtiyaç vardır. Bu çalışmada böylesi bir ihtiyaca model olabilecek bir yapı sunulmaya çalışılmıştır.

Uzun yıllardır üzerinde çalışılarak olgunlaştırılan ve birçok farklı yöntemin sunulduğu şifreleme algoritmaları, geliştirilmesi düşünülen ve özellikle, kontrol sistemlerini hedef alan uygulamalarda, kullanılması tasarlanan protokol için temel teşkil etmektedir.

Bu bakımdan, özellikle son yıllarda bu alanda hakimiyet kurmuş açık anahtarlı şifreleme yöntemleri incelenmiş ve en güvenilir yöntemlerden olan RSA algoritması protokolde kullanılmaya uygun görülmüştür. Bu algoritmanın

ihtiyaç duyduğu modüler aritmetik bilgisi fazla ayrıntıya girilmeden sunulmuştur.

Uygulama, platform olarak seçilen Microsoft Visual C++ 6.0 ile herhangi bir hazır kütüphane kullanmaksızın yazılarak bir proje altında toplanmıştır.

Bilim Kodu : 926
Anahtar Kelimeler : Protokol, şifreleme, kimlik doğrulama, hata tespiti, c++
Sayfa Adedi : 115
Tez Yöneticisi : Öğr.Gör.Dr. Nursel AKÇAM

DEVELOPING A SECURE, ERROR CORRECTIONAL AND DIGITALLY SIGNATURED PROTOCOL

(M.Sc. Thesis)

A.Çağrı BÖLÜCEK

GAZI UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
December 2005

ABSTRACT

Technological developments have always brought some troubles and needings with it. Today, the electronic devices which from using in the houses to industrial or military purposes have too many solutions that makes the usage, easier in much manner.

There are some needings that should get in count during such progresses because of to prevent the information in communication from third persons. In this working, the main aim is to present a structure that can be a model of such a needing.

The cryptographic algorithms that have been developing for years are a basic for the protocol which is especially planned for control based system targeting applications.

For this reason the asymmetric public key cryptograpy algorithms which are particularly progressed last years have been investigated and one of the most reliable public key algorithms, RSA has been decided to use in the protocol. The modular arithmetic knowledge which used by RSA has also been presented superficially.

It has been decided to use Microsoft Visual C++ 6.0 as a development platform. All needed fuctions have been fully implemented without using any predefined class or library and compiled in a MFC project.

Science Code : 926
Key Words : Protocol, cryptography, authentication, error detection, c++
Page Number : 115
Adviser : Dr. Nursel AKÇAM

TEŞEKKÜR

Çalışmalarımın her adımında yardım ve katkılarını esirgemeyen kıymetli hocam Dr. Nursel AKÇAM'a, işyerimdeki desteklerinden ötürü müdürüm Elek.Elektronik Yüksek Mühendisi Sayın Numan Kemal Saydam'a, mesai arkadaşım Hasan Gürler'e, akademik çalışmalarım için bana hep ilham vermiş sevgili annemin anısına ve bana her konuda destek olan kardeşlerim Burcu ve Bahadır'a teşekkür ederim.

İÇİNDEKİLER

Sayfa

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

SİMGELER VE KISALTMALAR LİSTESİ

Bu çalışmada kullanılmış bazı simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklama**
a ≡ b	a sayısı b sayısına denktir.
a | b	a sayısı b nin tam bölenidir.
Byte	8 bitlik değer
Checksum	Kontrol-Toplamı
Φ(n)	Euler phi fonksiyonu.
G	Polinomal değerini simgeler.
Host	İletişimi yöneten taraf.
KU_b	Genel Anahtar
KR_b	Özel Anahtar
Nonce	Oturumu belirleyecek özel bir bilgiyi barındıran mesaj
ord(a)	$a^t \equiv 1 \pmod n$ şartını sağlayacak t sayısı.
Polinomal	Kontrol-toplamının bulunması için kullanılan bölünen değeri.
$\mathbf{Q_n}$	Kuadratik artıkların oluşturduğu küme.
$\overline{\mathbf{Q}}_\mathbf{n}$	Kuadratik artık olmayan tamsayıların oluşturduğu küme.
Slave	İletişimde Host tarafından yönetilen taraf.
X	Şifrelenmemiş Düz Metin
Y	Şifrelenmiş Mesaj
$\mathbf{Z}_\mathbf{n}^*$	Z_n kümesinin çarpan grubu

Kısaltmalar	**Açıklama**
crt	Chinese Remainder Theorem (Çin Kalan Teoremi)
crc	Cyclic Redundancy Check (Döngüsel Artıklık Denetimi)
des	Data Encryption Standard (Veri Şifreleme Standardı)

Kısaltmalar	Açıklama
dss	Data Security Standard (Veri Güvenliği Standardı)
gcd	Greatest Common Divisor (Ortak Bölenlerin En Büyüğü)
gnfs	Generalized Number Field Sieve (Genel Sayı Cismi)
IBM	International Business Machines (Uluslararası İş Makineleri)
kdc	Key Distribution Center (Anahtar Dağıtım Merkezi)
lan	Local Area Network (Yerel Ağ)
mips	Million Instruction per Second (Saniyede işlenen komut sayısı)
MIT	Massachusetts Institute of Technology
mod	Modül
obeb	Ortak Bölenlerin En Büyüğü
okek	Ortak Katların En Küçüğü
pgp	Pretty Good Privacy
rf	Radyo Frekansı (Radio Frequency)
RSA	Rivest, Shamir ve Adleman'ın şifreleme algoritması
snfs	Special Number Field Sieve (Özel Sayı Cismi)

1. GİRİŞ

Bu çalışma, teknolojik gelişimlerin beraberinde getirdiği veri iletişiminde güvenlik sorunlarına getirilen güncel çözümleri derleyerek bir protokol altında toplamayı amaçlamaktadır. Protokolün yapı itibari ile RF haberleşme ve kontrol sistemlerinde kullanılmaya uygun olması düşünülmüştür.

Böyle bir protokolün geliştirilmesinde en önemli kısmı şüphesiz şifreleme algoritmaları oluşturacağından, araştırmanın büyük bir bölümü bu konuya ayrılmıştır. Şifreleme algoritmaları değerlendirilirken protokolün kontrol niteliğine uygunluğu gözetilerek gerçek zamanlı işlemleri engellemeyecek derecede hızlı olmasına dikkat edilmiştir. Bilindiği gibi şifrelemenin temelleri çok eskilere dayanmaktadır ancak esas çarpıcı gelişimini 1976 yılında Diffie ve Hellman' ın yayınladıkları "New directions in Cryptography" isimli makale ile yapmıştır (1). Bu makalede şifreleme için devrim niteliğinde olan "açık anahtarlamalı şifreleme" tanıtılmış, kesikli logaritmik hesaplamaların zorluğuna dayanan anahtar değişimi metodu sunulmuştur. Daha sonraki çalışmalar açık anahtarlı şifreleme algoritmalarını geliştirmiştir.

Benzer bir protokol oluşturma çabası 2004 yılında Nguyen, H. Aziz ve S.M. Ryan, tarafından sunulmuştur. Aziz ve Ryan araştırmalarında uzaktan kumanda edilebilen sistemler için yeni bir protokol modelini önermişlerdir (2) . Çalışmalarında odaklandıkları nokta iletişimin çoğunlukla komutlardan oluşması dolayısı ile mümkün olduğunca düşük bant genişliğine ihtiyaç duyması idi. Bu bakımdan güvenlik ihtiyacına fazlaca önem verilmemiş sadece algoritmanın gizli tutulması önerilmiştir. Güvenlik için harcanacak bant genişliği veri kayıplarının azaltılması için hata kontrolüne ayrılmıştır.

Güvenlik kaygısı gütmeyen çok daha genel bir çalışma Wolisz, Schieferdecker ve Walch tarafından sunulmuştur (3). Bu araştırmada iletişim protokolleri bir bütün

olarak düşünülmüş, herhangi bir uygulamaya dahil edilebilmeleri için ihtiyaç duyulacak gereksinimler belirlenerek birtakım test yöntemleri önerilmiştir.

Bir diğer önemli nokta veri iletişiminde bulunan terminallerin kimliklerinin doğruluğunun sorgulanmasıdır ki bu kısım aslen birçok şifreleme algoritmasının temelinde zaten yer almaktadır, bu bakımdan aynı başlık altında incelenmişler ancak çalışmadaki önemi dolayısı ile ayriyeten vurgulanmıştır.

İkinci bölüm şifreleme ve kimlik kontrolü konularına genel bir giriş niteliğindedir. İlerleyen bölümlerde ihtiyaç duyulacak matematiksel önbilgi bu bölümde sunularak, sayılar teoreminin temelleri kısaca belirtilmiştir.

Üçüncü bölümde protokolde kullanılması daha uygun görülen açık anahtarlı şifreleme sistemleri bir bütün olarak incelendikten sonra, mevcut protokoller arasında niteliklerinin uygunluğu ve popülaritesiyle gelişime açık olduğu düşünülerek seçilen RSA algoritması ayrıntılarıyla incelenmiştir.

Açık anahtarlı şifreleme mantığını kullanan tüm yöntemlerin ortak problemi haline gelmiş anahtarların duyurulması işlemi dördüncü bölümde ele alınmıştır. Mevcut çözümler ve araştırmalar bu bölümde sunulmuş, uygun herhangi bir yöntemin seçilmesi, bu protokolü kullanmaya niyetlenen geliştiricilerin, seçimine bırakılmıştır. Ayrıca bu bölümde, açık anahtarlı şifreleme sistemine entegre edilebilecek bir kimlik doğrulama mekanizması sunulmuştur.

Daha önce de belirtildiği gibi protokolün RF sistemlerde kullanılması amaçlandığından muhtemel veri kayıplarının oldukça fazla yaşanacağı ortadadır. Bu zayıflığın giderilmesi kullanılacak donanımsal araçların dikkatli seçilmesi ve bir standardizasyona bağlanması ile mümkün olacaktır ki bu husus, çalışmada yer alamayacak kadar derindir ve bir başka araştırma konusunu oluşturabilir. Bununla birlikte verideki kayıpların veya bozulmaların neden olacağı hatalı kodları engellemek amacı ile protokole “Error Correction” (Hata düzeltme) algoritması da dahil edilmeli ve diğer durumlarda da gözetilen gerçek zamanlı işlemleri

engellemeyecek hızda çalışabilmesi gözetilmelidir. Beşinci bölüm bu konuya yoğunlaşmaktadır.

Son bölümde ise bir model olarak sunulan protokolün bazı temel özelliklerini simüle eden ve Microsoft Visual Studio .NET ile geliştirilmiş programın açıklamasına yer verilmiştir. Program ve kaynak kodları, tez ile birlikte ek olarak sunulmuştur.

2. MATEMATİKSEL ÖNBİLGİ

Bu bölüm sayılar teorisi ve tekrarlı kare alma konularını içeren birtakım temel bilgileri hatırlatmak, ileriki bölümlerde referans olarak kullanılmak maksadı ile hazırlanmıştır.

2.1. Sayılar Teorisi

2.1.1. Bölünebilirlik

Tamsayılar kümesi; Z={..., -3,-2,-1,0,1,2,3,...} kümesi ile tanımlıdır.

Tanım 1: *a* ve *b* tamsayılar olsun. Eğer b=a.c eşitliğini sağlayan bir *c* tamsayısı tanımlanabiliyorsa *a* için *b* ' nin bölenidir denir. "*a*|*b*" ile sembolize edilir.

Örnek: (-3) | 18 , çünkü 18=(-3).(-6) veya (173 | 0) çünkü 0= (173).(0)

Aşağıda tamsayılarda bölünebilirliğin temel özellikleri verilmiştir:

Özellikler:

1. $a|a$
2. Eğer $a|b$ ve $b|c$ ise $a|c$ olur
3. Eğer $a|b$ ve $a|c$ ise her x,y doğal sayısı için $a|(bx+cy)$
4. Eğer $a|b$ ve $b|a$ ise $a=\pm b$.

Tanım 2: (tamsayılar için bölme algoritması) $b\geq 1$ olacak şekilde *a* ve *b* tamsayıları seçilsin. *a bölüm b* işlemi q:bölüm ve r: kalan' ı göstermek üzere:

a= qb + r olarak belirtilir. ($0 \leq r \leq b$)

Bölme işleminde kalan sayıya *a mod b ile* erişilebilir.

Tanım 3: Eğer $c|a$ ve $c|b$ olacak şekilde bir c tamsayısı mevcutsa, c tamsayısına *a ve b* ' nin ortak bölenidir denir.

Tanım 4: Eğer negatif olmayan bir d tamsayısı:

1) *a ve b* nin ortak böleni
2) $c|a$ ve $c|b$ iken $c|d$

şartlarını sağlıyorsa d tamsayısına ortak bölenlerin en büyüğü denir ve OBEB(*a,b*) veya gcd(a,b) olarak gösterilir.

Örnek: 12 ve 18 tamsayılarının ortak bölenleri {±1,±2,±3,±6}, ve ortak bölenlerin en büyüğü gcd(12,18)=6.

Tanım 5: Eğer negatif olmayan bir l tamsayısı

1) $a|l$ *ve* $b|l$ iken ve
2) $a|c$ ve $b|c$ iken $l|c$

şartlarını sağlıyorsa l tamsayısına *a ve b* tamsayılarının ortak katlarının en küçüğü denir ve OKEK(a,b)=l veya lcm(a,b)=l olarak gösterilir.

Teorem 1: *a* ve *b* pozitif tamsayılar ise OKEK(a,b)=(a.b)/OBEB(a,b) dir.
*Örnek***:** a=12 ve b=18 pozitif tamsayıları için, OBEB(12,18)=16 iken OKEK(12,18)= 12.18/6=36 eşitliği sağlanır.

Teorem 2: p asal sayısı p|ab ise aynı zamanda p|a veya p|b dır. (veya her iki durumu da sağlar)

Teorem 3: Sonsuz miktarda asal sayı mevcuttur.

Teorem 4: Bir sayının asal çarpanlarına ayrılmış hali yalnız tek bir şekilde ifade edilebilir.

Teorem 5: $e_i \geq 0$ ve $f_i \geq 0$ olmak üzere, $a=p_1^{e_1}p_2^{e_2}...p_k^{e_k}$, $b=p_1^{f_1}p_2^{f_2}...p_k^{f_k}$ ise;

$$OBEB(a,b)= p_1^{min(e1,f1)}\ p_2^{min(e2,f2)}...p_k^{min(ek,fk)} \quad [2.1]$$

ve

$$OKEK(a,b)= p_1^{max(e1,f1)}\ p_2^{max(e2,f2)}...p_k^{max(ek,fk)} \quad [2.2]$$

Örnek: a=4864 = 2^8.19 , b=3458 =2.7.13.19 seçilsin. OBEB(4864,3458)=2.19 ve OKEK(4864,3458)=2^8.7.13.19 ile teoremi sağlamaktadır.

Tanım 6: $n \geq 1$ için $\varphi(n) \rightarrow$ [1,n] aralığındaki n ile aralarında asal olan tamsayıları belirtsin. φ : "Euler phi" fonksiyonu olarak adlandırılır.

Teorem 6: (Euler phi fonksiyonunun özellikleri)

1. p asal bir sayı ise φ(p)=p-1.
2. Eğer OBEB(m,n) =1, ise φ(m.n)= φ(m). φ(n) olur.
3. n tamsayısının çarpanlarına ayrılmış hali; $p_1^{e_1}\,p_2^{e2}.....p_k^{e_k}$ ise:

$$\varphi(n) = p_1^{e1-1}(p_1-1)p_2^{e2-1}(p_2-1)...p_k^{ek-1}\,(p_k-1)$$

$$=n\left(1-\frac{1}{p1}\right)\left(1-\frac{1}{p2}\right).....\left(1-\frac{1}{pk}\right)$$

2.1.2. Öklit algoritmaları

a ve *b ;* n den küçük veya eşit değerler alabilen ve negatif olmayan tamsayılar olsun. n tamsayısının ikilik sistemde (binary) ifadesi için *ln(n)+1* bite ihtiyaç olacaktır. Aşağıdaki Çizelge 2.1.'de dört temel matematik işlemi için kullanılan klasik algoritmaların bit gereksinimleri gösterilmiştir

Çizelge 2.1. Tamsayılar kümesinde işlemlerin bit ihtiyaçları

İŞLEM	Bit İhtiyacı
Toplama a+b	$O(\ln a + \ln b) = O(\ln n)$
Çıkarma a-b	$O(\ln a - \ln b) = O(\ln n)$
Çarpma a.b	$O((\ln a)(\ln b)) = O((\ln n)^2)$
Bölme a=qb+r	$O((\ln q)(\ln b)) = O((\ln n)^2)$

İki tamsayının ortak bölenlerinin en büyüğü bulunurken uygulanacak en efektif algoritma aşağıda belirtilen teoremdeki temelle açıklanan Öklit (Euclidean) Algoritmasıdır.

Teorem 7: *a* ve *b* pozitif tamsayılar ve a > b olsun. Bu tamsayılar arasında OBEB(a,b) = OBEB(b, (a *mod* b)) olacak şekilde bir eşitlik vardır.
Öklit algoritması bu teoremi kullanarak aşağıdaki sırada verilen bölme işlemlerini gerçekleştirerek sonuca ulaşır.

$$
\begin{aligned}
a &= q_1 b + r_1, & 0 < r_1 < b \\
b &= q_2 r_1 + r_2, & 0 < r_2 < r_1 \\
r_1 &= q_3 r_2 + r_3, & 0 < r_3 < r_2 \\
&\vdots \\
r_{n-2} &= q_n r_{n-1} + r_n, & 0 < r_n < r_{n-1} \\
r_{n-1} &= q_{n+1} r_n + 0, & (a > b > r_1 > r_2 > \ldots > r_n > 0)
\end{aligned}
$$

en büyük ortak bölen:

$OBEB(a,b) = OBEB(b,r_1) = OBEB(r_2,r_1) = \ldots = OBEB(r_n,0) = r_n.$

Netice itibariyle $OBEB(a,b) = r_n$ olacaktır.

Çizelge 2.2. Öklit algoritması

Algoritma**:** İki tamsayının OBEB' ini hesaplayan öklit algoritması:

GİRDİ: iki negatif olmayan a ve b tamsayıları (a≥b).

ÇIKTI: a ve b tamsayılarının ortak bölenlerinin en büyüğü

1.***while***(b≠0)
do
(a mod b) → r,
b→a
r→b
2.***return*** a

Teorem 8: Yukarıdaki algoritmanın çalışma zamanını belirleyen şey $O((\ln n)^2)$ ile ifade edilen bit operasyonudur.

Öklit algoritması a ve b gibi iki tamsayının d ile ifade edilen OBEB' ini hesaplamak yerine ax+by=d eşitliğini sağlayacak x ve y sayılarını hesaplayacak şekilde genişletilebilir.

Çizelge 2.3. Genişletilmiş Öklit algoritması

Algoritma**:** Genişletilmiş öklit algoritması:

GİRDİ: iki negatif olmayan a ve b tamsayıları (a≥b).

ÇIKTI: d=OBEB(a,b) ve ax+by=d eşitliğini sağlayan x ve y tamsayıları (negatif olabilir!)

1.***If*** b=0 ***then*** a→d, x≠1 ***then*** 0→y, ***return*** (d,x,y).

2.$x_2 \rightarrow 1$, $x_1 \rightarrow 0$, $y_2 \rightarrow 0$, $y_1 \rightarrow 1$ (Initialize)

3. ***while*****(**b>0**)**

a/b→q, (a-qb)→r, $(x_2 - qx_1) \rightarrow x$, $(y_2 - qy_1) \rightarrow y$.

b→a, r→b, $x_1 \rightarrow x_2$, $x \rightarrow x_1$, $y_1 \rightarrow y_2$ ve $y \rightarrow y_1$.

b→a, r→b, $x_1 \rightarrow x_2$, $x \rightarrow x_1$, $y_1 \rightarrow y_2$ ve $y \rightarrow y_1$.

4.a→d, x2→x, y2→y ve ***return*****(** d,x,y**).**

Teorem 9: Genişletilmiş öklit algoritmasının çalışma zamanını belirleyen $O((\ln n)^2)$ ile ifade edilen bit operasyonudur.

2.1.3. Denklik

n pozitif bir tamsayı olsun;

Tanım 7: a ve b tamsayıları için (*b mod n*) işleminin sonucu a' yı veriyor ise bu iki tamsayı mod n' e göre birbirlerine denktir denir.

Teorem 10: (Denkliğin özellikleri) Tüm a,a_1,b,b_1,c tamsayıları için aşağıdaki önermeler doğrudur:

1. $a \equiv b \pmod{n}$ için gerek ve yeter şart a ve b tamsayılarının n ile bölündüklerinde aynı kalanı vermesidir.
2. $a \equiv a \pmod{n}$ (yansıma özelliği)
3. $a \equiv b \pmod{n}$ ise $b \equiv a \pmod{n}$ dır. (simetri özelliği)
4. $a \equiv b \pmod{n}$ ve $b \equiv c \pmod{n}$ ise $a \equiv c \pmod{n}$' dir (geçişme özelliği)
5. $a \equiv a_1 \pmod{n}$ ve $b \equiv b_1 \pmod{n}$ ise $(a+b) \equiv (a_1 + b_1) \pmod{n}$ ve $ab \equiv a_1 b_1 \pmod{n}$.

Tanım 8: Modül n in tamsayı kümesi Z_n ile gösterilir ve {0,1,2,...,(n-1)}' den oluşur. Z_n kümesinde gerçekleştirilen toplama, çıkarma ve çarpma işlemleri modül n' e göre gerçekleştirilir.

Tanım 9 (CRT (Çin Kalan Teoremi)): aralarında asal n1,n2,n3,...,nk tamsayıları için denklik sisteminin:

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

yalnız tek bir çözümü vardır ve $n = n_1 n_2 ... n_k$.

Örnek: $x \equiv 3 \pmod{7}$, $x \equiv 7 \pmod{13}$ denklik çiftinin mod (7.13) = mod 91' e göre tekbir çözüm kümesi vardır ve $x \equiv 59 \pmod{91}$.

Teorem 11: OBEB(n_1,n_2) = 1, ise x ≡ a (mod n_1), x ≡ a (mod n_2) denklik çiftinin tek çözümü x ≡ a (mod n_1n_2)' dir.

Tanım 10: Z_n kümesinin çarpan grubu Z_n^* ile gösterilir ve

Z_n^* = { a Є Z_n | OBEB(a,n) =1 }. n' in asal olması durumunda

Z_n^* = { a | 1 ≤ a ≤ (n-1) }.

Tanım 11: Z_n^* kümesinin eleman sayısı Z_n^*' in derecesini belirtir ve

$|Z_n^*|$ ile gösterilir.

Teorem 12: n ≥ 2 şartını sağlayan bir tamsayı iken:

1. (Euler Teoremi) Eğer a Є Z_n^* ise $a^{\varphi(n)} \equiv 1 \pmod n$.

2. Eğer n iki farklı asal sayının çarpımı ise ve r ≡ s (mod φ(n)) ise; tüm a tamsayıları için $a^r \equiv a^s \pmod n$ olur. Başka bir deyişle mesela n gibi bir üssel sayının modülünü alma işlemi φ(n)' in modülünü hesaplamaya indirgenebilir.

Tanım 12: a,b Є Z_n olsun. modül n' in çarpmaya göre tersi x Є Z_n dir ve a.x ≡ 1 (mod n) şartını sağlayacak bir tek a tamsayısı mevcuttur.

Teorem 13: a Є Z_n iken a tamsayısının, sadece OBEB(a,n) =1 şartını sağlaması durumunda tersi alınabilir.

Teorem 14: p bir asal sayı olsun.

1.(Fermat Teoremi) OBEB(a,p)=1 ise $a^{p-1} \equiv 1 \pmod p$.

2.Tüm a tamsayıları için $a^p \equiv a \pmod p$.

Tanım 13: a Є Z_n^* olsun. a tamsayısının derecesi, $a^t \equiv 1 \pmod n$ şartını sağlayan en küçük t tamsayısıdır ve ord(a) ile gösterilir.

Teorem 15: a Є Z_n^* tamsayısının derecesi t ise ve $a^s \equiv 1 \pmod n$ ise t tamsayısı s yi böler.

Tanım 14: g Є Z_n^* olsun. g tamsayısının derecesi φ(n) ise g için Z_n^* in temel elemanı veya üreteni denir. Z_n^*' in bir üreteni varsa Z_n^* için devirlidir (cyclic) denir.

Teorem 16: Z_n^* in temel elemanının özellikleri:

1. Z_n^*' in temel elemanının bulunması için gerek ve yeter şart p asal bir sayı ve k ≥ 1 olmak üzere n = 2, 4, p^k veya $2p^k$ ile ifade edilebiliyor olmasıdır.

2. Eğer g Z_n^*' in temel elemanı ise Z_n^* = { a^i (mod n) | 0 ≤ i ≤ (φ(n)-1) }.

3. g; Z_n^*' in temel elemanı ise b = g^i (mod n) sayısının da Z_n^*' in temel elemanı olması için gerek ve yeter şart OBEB(i, φ(n)) = 1. Z_n^*'in devirlilik özelliği kullanılırsa Z_n^*' in temel eleman sayısı φ(φ(n)) ile bulunur.

4. g Є Z_n^* tamsayısının temel eleman olabilmesi için gerek ve yeter şart, her asal p için $g^{\varphi(n)/p} \neq 1 \pmod n$ sağlamasıdır.

Tanım 15: a Є Z_n^* olsun. Eğer $x^2 \equiv a \pmod n$ denkliğini sağlayacak bir x Є Z_n^* tamsayısı mevcut ise a için n moduna göre "kuadratik artık (residue)" denir.n moduna göre kuadratik artıkların oluşturduğu küme Q_n ve kuadratik artık olmayanların oluşturduğu küme $\overline{Q_n}$ ile sembolize edilir.

Örnek: g=6 tamsayısı Z_{13}^*' ün temel elemanıdır. g' nin üsleri aşağıdaki çizelgede Q_{13} ={1,3,4,9,12} ve $\overline{Q_{13}}$ ={2,5,6,7,8,11} olarak gösterilmiştir.

Çizelge 2.4. g=6 için oluşturulmuş örnek hesaplama

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
g^i mod13	1	6	10	8	9	2	12	7	3	5	4	11

2.3. Tekrarlı Kare Alma Metodu

b^n (mod m) işlemi söz konusu iken m ve n sayılarının çok büyük olması durumunda tekrarlı kare alma yöntemi modüler aritmetiğin temel hesaplama yöntemlerinden biridir. Bu işlemi b nin sürekli üssünü almaktan daha mantıklı ve hızlı bir yöntemi vardır. İlgili algoritma şöyle çalışmaktadır:

$n_0,n_1,...,n_{k-1}$ 'n sayısının ikili basamaklarını belirtmek üzere:

$n = n_0 + 2n_1 + 4n_2 + ... + 2^{k-1}n_{k-1}$ (nj = 0 or 1).

Buradan aşağıdaki hesaplama yapılabilir:

$$b^2 , (b^2)^2 = b^4 , (b^4)^2 = b^8 , ..., (b^{2^{k-2}})^2 = b^{2^{k-1}} \qquad [2.3]$$

$$b^n = b^{n0 + 2n1 + 4n2 + ... + 2^{k-1} nk-1}$$

$$= b^{n0}.(b^2)^{n1}.(b^4)^{n2}...(b^{2^{k-1}})^{n k-1}$$

Görüldüğü gibi b^2 ; b^4 ,b^8 ,...,$(b^{2^{k-2}})$ = $b^{2^{k-1}}$ hesaplanarak tüm bu değerlere temel oluşturduğundan b^n (mod m) kolaylıkla hesaplanabilir.

Örnek: 432^{678} (mod 987) işlemini hesaplamak istediğimizi varsayalım: Bu işlemi yapmanın en kısa yolu kare alma işleminin belli bir sayı seçilerek bu sayı için tekrarlanmasıyla olacaktır ki:

$$432^2 = 186624 \equiv 81 , 432^4 \equiv 81^2 \equiv 639 , 432^8 \equiv 639^2 \equiv 690 ...432^{512} \equiv 858$$

678 = 512 + 128 +32 +4 +2 olacağından dolayı,

$$432^{678} \equiv (81)(639)...(858) \equiv 204$$

Eğer sayıların birkaç – yüz basamaktan oluştuğu düşünülürse çarpım işlemlerinin gerçekleştirilebilmesi için özel birtakım rutinlerin geliştirilmesi gerekmektedir.

Hızlı üs alma işleminin ardında yatan fikir, üssün 2' nin bir kuvveti olması durumunda kare alma işleminin yinelenmesidir.

$$x^8 = ((x^2)^2)^2$$

Eğer söz konusu üs 2' nin kuvveti değilse 2' nin kuvvetlerinin toplamı şeklinde yazılır.

$$x^{291} = x^{256} . x^{32} . x^{2} . x^{1}$$

Bu şekilde düşünüldüğünde x^{n}' in hesabının gerçekleştirebilmesi için (log n) adet işlem yapılması gerekir.

3. AÇIK ANAHTARLI ŞİFRELEME

Açık anahtarlı şifrelemenin gelişmesi, kriptografi tarihindeki en büyük devrimdir. Başlangıcından günümüze kadar, bütün kriptografik sistemler, yerine koyma (süpstütüsyon) ve permütasyon işlemlerinin temel alınmasıyla oluşturuldular. Sadece elle hesaplanabilen algoritmalarla çalışabilme döneminden sonra, şifreleme/deşifreleme yapan rotor makinelerinin ortaya çıkması sonucunda, geleneksel kriptografide büyük bir gelişme kaydedildi. Elektro-mekanik rotor, çok fazla inceliklere sahip ve karmaşık kriptografik sistemlerin geliştirilebilmesini sağladı. Mevcut bilgisayarlarla daha karmaşık sistemler tasarlandı ve bu sistemlerden en bilineni olan (IBM´in geliştirdiği) Lucifer projesi gelişerek DES´i (Data Encryption Standard) oluşturdu ve DES`i dünyadaki kriptografi teknikleri arasında en yüksek seviyeye getirdi. Rotor makineleri ve DES, önemli avantajlar sunmalarına rağmen, halen, yerine koyma ve permütasyon işlemlerine bağımlıdırlar.

Açık anahtarlı kriptografi, kendisinden önceki gelişmelerden radikal bir kopuştur. Açık anahtarlı kriptografik sistemlerin en önemli özellikleri, süpstütüsyon ve permütasyondan çok matematiksel işlevler üzerine temellenmiş olmalarıdır. Daha da önemlisi, açık anahtarlı kriptografi, tek anahtar kullanan simetrik geleneksel şifreleme algoritmalarının tersine, iki ayrı anahtarın asimetrik kullanımını öngörür. Anahtar dağıtımı ve kimlik denetimi gibi gizlilik ve güven gerektiren durumlarda, iki anahtar kullanımı etkili sonuçlar ortaya koymuştur.

Başlangıç olarak, açık anahtarlı şifreleme ile ilgili bazı yaygın, yanlış fikirlerden bahsedelim. Bu hususta en yanlış düşüncelerden birisi, açık anahtarlı şifrelemenin, kriptoanalize karşı geleneksel şifreleme yöntemlerinden daha güvenli olduğudur. Her iki sistemi birbirine karşı bu derece genel bir sınava tabii tutmak hiç de doğru olmayacaktır. Örneğin böyle bir iddia, Gardner´ın meşhur Scientific America dergisinde 1977 yılında yayınladığı makalesinde yapılmıştır (4). Aslen, şifrelemenin güvenliği, anahtarın uzunluğuna ve kırılan şifreli metnin içerdiği hesapsal işlemlerin karmaşıklığına dayanır. İster geleneksel ister açık anahtarlı şifreleme olsun,

kriptoanalitik bakış açısına göre birini direğinden (yöntemler açısından) üstün tutmak yanlış olur.

Bir diğer yanlış düşünce de, genel amaçlı kullanım için geliştirilmiş bir teknik olan açık anahtarlı şifrelemenin, geleneksel şifrelemeyi modası geçmiş kıldığıdır. Tam tersine, geleneksel şifrelemeden vazgeçileceği sanısı, açık anahtarlı şifreleme yöntemlerinin, ağır matematiksel fonksiyonlarından dolayı, ihtimal dışı gözükmektedir. Gizli anahtarların, açık anahtarlı şifreleme sistemleri ile dağıtılmaları bunun en belirgin örneğidir.

Son olarak, açık anahtarlı şifreleme kullanılırken, geleneksel şifrelemenin daha hantal anahtar dağıtım merkezleri ile karşılaştırıldığında, açık anahtarlı sistemlerin anahtar dağıtımının üzerinde kafa yorulması gerekmeyen, sıradan ve basit bir iş olduğuna dair yanlış bir anlayış vardır. Aslında, protokolün bazı biçimleri gereklidir fakat, geleneksel şifreleme yöntemlerinin ihtiyaç duyduğu ve bölüm 4.1'de açıklanan, merkezi temsilciler ve prosedürler, açık anahtarlı şifrelemenin ihtiyaç duyduklarından daha basit daha karmaşık veya daha etkili değildir.

Bu bölüm, açık anahtarlı şifrelemeye genel bir giriş mahiyetindedir. İlk önce, bu yöntem kavramsal anlamda tanıtılmaya çalışılacaktır. Açık anahtarlı kriptografi ile ilgili en ilginç nokta; açık anahtarlı kriptografinin, pratik olarak uyarlanışı gösterilmeden, tekniğin mimarisi geliştirilmesi ve tamamen doğru kabul edilerek yayınlanmasıdır.

Daha sonra, açık anahtarlı şifreleme yöntemi için, uygulanabilir olarak gösterilen en önemli şifreleme/deşifreleme algoritması olan ve geliştirilmesi planlanan protokol için hem standardizasyon hem de akademik olarak gelişime en açık olduğu düşünülen RSA algoritması incelenecektir. Bu bölümün sonunda, açık anahtarlı sistemler için, anahtar dağıtımı ve anahtar dağıtım yönetimleri incelenerek geliştirilmesi planlanan protokole en uygun olanı belirlenecektir.

Açık anahtarlı kripto sistemlerinin çoğunluğu, sayılar teorisini temel alır. Bu bölümde verilen sonuçları algılamak için, sayılar teorisine tamamen hakim olmak gerekmese dahi bir önceki bölümde anlatılan gerekli temel matematiksel bilgilerin hatırlanması bu bölümün daha kolay ve hızlı anlaşılmasına ciddi ölçüde yardımcı olacaktır.

3.1. Açık Anahtarlı Şifreleme Sistemlerinin Prensipleri

Açık anahtarlı şifrelemenin genel amacı, geleneksel şifrelemenin en büyük iki problemine çözüm sağlamaktı. Bu problemlerden ilki gizli anahtarların dağıtımıdır. Gizli anahtar, geleneksel şifreleme uygulamalarının (DES, IDEA, Blowfish, CAST128, RC5, ...) kullandığı anahtarlardır.

Geleneksel şifrelemeden yararlanarak birbirlerine şifrelenmiş metinler gönderecek olan taraflar, şifreleme ve deşifreleme işlemleri için, ya bir şekilde kendilerine ulaştırılmış olan anahtarı kullanırlar veya bir anahtar dağıtım merkezinden faydalanırlar. Açık anahtarlı kriptografinin mucitlerinden olan Whitfield Diffie kriptografinin özü olan, iletişimde %100 güvenlik esasını hiçe sayan bir anahtar dağıtım merkezi kullanma gerekliliğini ortadan kaldırılmıştır (1). Bu durum tarafların kullanacakları gizli anahtarları bir anahtar dağıtım yetkilisinden almaları istendiği taktirde üçüncü bir kişi için iletişimin anlaşılabilir olma tehlikesini barındırır.

Diffie, üzerinde düşündüğü ikinci problem olan "dijital imza" konusunun, önceki ile ilgisi olmayan başka bir konu olduğunu fark etmiştir. Eğer kriptografinin kullanımı, sadece askeri konularda değil, özel ve kâr amaçlı uygulamalarda da kullanılacak kadar yaygın olsaydı, bu durumlar için kullanılacak elektronik belge ve dokümanlarda da, kağıt dokümanlarda kullanılan kişisel imzalara gerek duyulurdu ve dijital imzalar sayesinde, bir mesajı kimin gönderdiği kesinlikle bilinmiş olur ve bu da herkesi memnun eden bir metot olurdu.

Diffie ve Hellman, 1976`da, her iki probleme de, daha önceki bütün kriptografik gelişimlerden farklı, radikal bir çözüm getirmeyi başarmışlardır.

3.2. Açık Anahtarlı Şifreleme Sistemlerinin Karakteristikleri

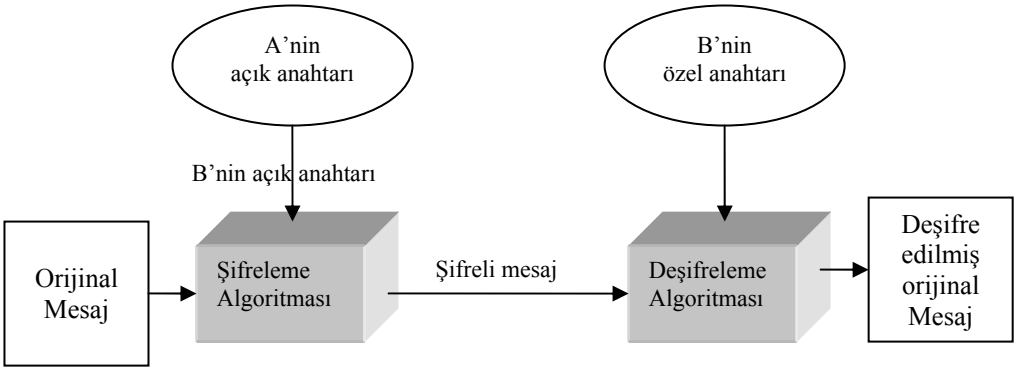
Açık anahtarlı şifreleme/deşifreleme algoritmaları, şifreleme için bir anahtara, deşifreleme içinse bu anahtarla ilişkisi olan başka ikinci bir anahtara ihtiyaç duyarlar. Bu durumda bir güvenlik sağlamış olur. Bu algoritmalar şu önemli karakteristiğe sahiptirler:

*Sadece kriptografik algoritma ve deşifreleme anahtarı verilmişken, bir takım basit hesaplamalar ile şifreleme anahtarını bulmak mümkün değildir.

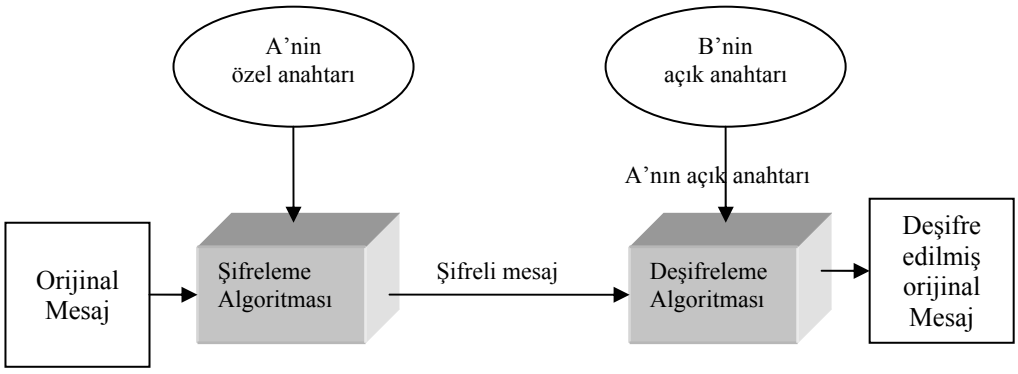
Bununla beraber RSA gibi bazı algoritmalar şu karakteristikleri de gösterirler:
*Her iki benzer anahtar da şifreleme ve deşifreleme için kullanılabilir. Bununla beraber, bir anahtar şifreleme için kullanılmışsa, deşifreleme için diğer anahtar kullanılmalıdır.

Şekil 3.1.*a.*, açık anahtarlı şifreleme yöntemini göstermektedir. Başlıca adımlar şunlardır:

1. Ağdaki her sistem, mesaj alındığında şifreleme ve deşifreleme için kullanacak olduğu anahtar parçasını yaratır.
2. Her sistem, şifreleme anahtarını herkesçe erişilebilecek bir dosya yada yazmaç içerisine kaydederek paylaştırır. Bu anahtarın, açık olan kısmıdır (public key). Özel anahtar saklı tutulur.
3. Eğer, A, B' ye bir mesaj yollamak isterse, mesajı B' nin açık anahtarını kullanarak şifreler.
4. B, mesajı aldığında, bu mesajı kendi özel anahtarını kullanarak deşifre eder. Diğer hiçbir alıcı mesajı deşifreleyemez, çünkü mesajı deşifre edecek olan özel anahtarı sadece B bilir.



(a) Sifreleme



(b) Kimlik Kontrolü

Şekil 3.1. *a.* Açık anahtarlı şifreleme yapısı *b.* Kimlik kontrolü yapısı

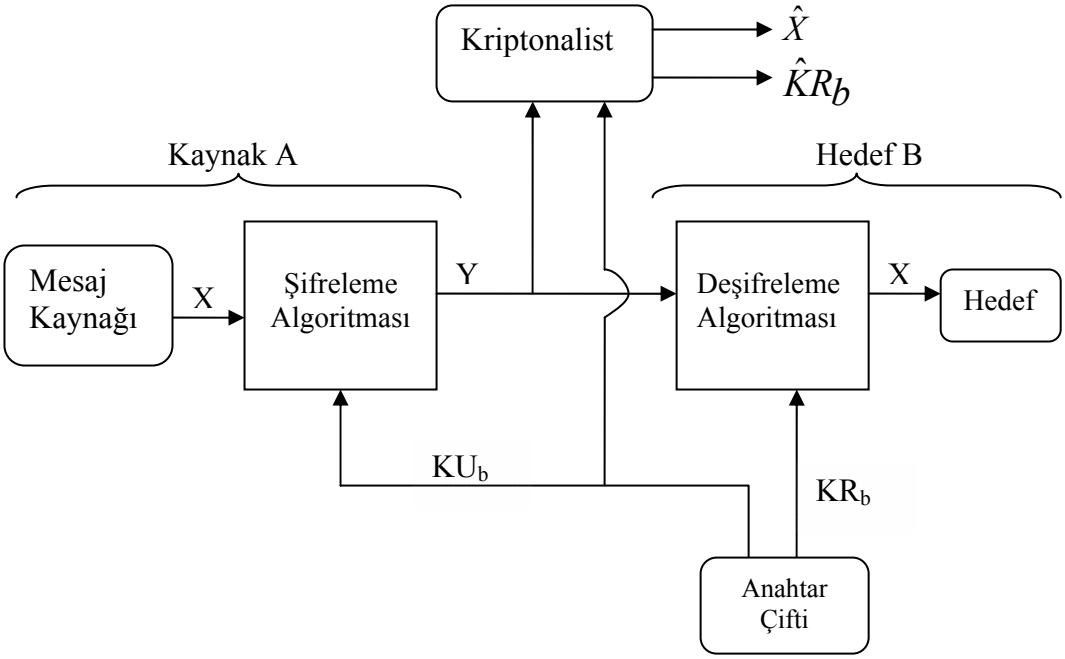
Şekil.3.1.'den anlaşılacağı üzere her katılımcı, diğerlerinin açık anahtarlarına erişim hakkına sahiptir ve katılımcılar özel anahtarlarlarını lokal olarak yaratırlar. Bu yüzden, özel anahtarların paylaşılmasına gerek yoktur. Herhangi bir sebepten ötürü özel anahtarlar sahipleri tarafından değiştirilmek istenebilirler, bu durumda değişmiş olan yeni açık anahtar ilgili yerlere yeniden gönderilerek eskisi ile yer değiştirilir.

Çizelge 3.1., geleneksel ve açık anahtarlı şifrelemenin farklarını açıkça göstermektedir. Geleneksel şifrelemede kullanılan anahtarı, açık anahtarlı şifrelemede kullanılan anahtarlardan ayırmak için "gizli anahtar" olarak, açık

anahtarlı şifrelemede kullanılan iki anahtarı da, “genel anahtar” ve “özel anahtar” olarak isimlendirilmiştir. Özel anahtar, her zaman gizli tutulacak olan anahtardır, fakat, geleneksel şifrelemedeki gizli anahtarla karışmaması için gizli anahtar yerine özel anahtar diyerek isimlendirilmiştir.

Çizelge 3.1. Geleneksel ve açık anahtarlı şifreleme

Geleneksel şifrelemede:	**Açık anahtarlı şifrelemede**:
Çalışması bakımından:	
1. Şifreleme ve deşifreleme için aynı algoritma aynı anahtarla birlikte kullanılır.	1. Şifreleme ve deşifreleme için bir algoritma ve anahtarlardan birisi kullanılır. Şifreleme için kullanılan anahtar, deşifreleme için kullanılamayacağı gibi tersi de mümkün değildir.
2. Gönderen ve alan, algoritmayı ve anahtarı paylaşmalıdır.	2. Gönderen ve alan, ilişkili anahtarlardan birine sahip olmalıdırlar (aynı olanı değil).
Güvenlik bakımından:	
1. Anahtar gizli tutulmalıdır.	1. Anahtarlardan biri gizli tutulmalıdır.
2. Diğer bilgiler saklandığında, mesajı deşifre etmek imkansız olmalıdır.	2. Diğer bilgiler saklandığında, mesajı deşifre etmek imkansız olmalıdır.
3. Algoritma ve şifreli metin örnekleri bilmek, anahtarı çözmek için yetersiz olmalıdır.	3. Algoritma, şifreli metin örnekleri bilmek ve anahtarlardan birine sahip olmak, diğer anahtarı bulmak için yetersiz olmalıdır.



Şekil 3.2. Açık anahtarlı şifreleme yapısı (ayrıntılı)

Şekil 3.2. yardımıyla, açık anahtarlı şifreleme yapısı açıklanabilir: A, mesaj gönderecek bir kaynağı ve göndermeyi düşündüğü $X = \{X_1, X_2, ..., X_M\}$, genel bir alfabe kullanarak oluşturduğu, M sonlu sayısında kelimeden oluşan bir mesajı simgeler. Bu X mesajı, B alıcısı için tasarlanır. B, birbiri ile ilişkili olan bir anahtar çifti yaratır: bir genel anahtar; KU_b ve bir özel anahtar KR_b. KR_b, yalnız *B* tarafından bilinir. KU_b ise, *A* tarafından erişilebilecek olan *B*`nin açık anahtarı olur.

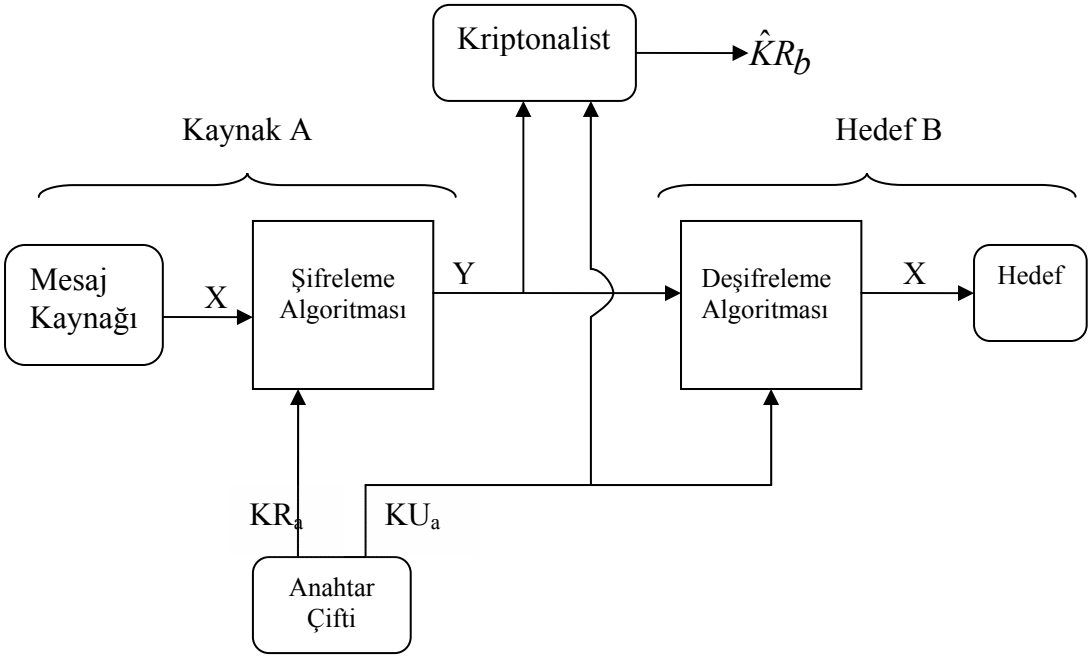
X düz metnini ve KU_b anahtarını girdi olarak alan *A*, mesajı; E_{KU_b} şifreleme fonksiyonu ile şifreleyerek, $Y = \{Y_1, Y_2, ..., Y_N\}$ metnine dönüştürür.

$$Y = E_{KU_b}(X) \qquad [3.1]$$

Alıcı olan *B*, özel anahtarın sahibi olarak, şifreli metni düz yazıya; D_{KR_b} çözümleme fonksiyonu ile çözümler.

$$X = D_{KR_b}(Y) \qquad [3.2]$$

Bir üçüncü kişi, iletişimi izleyerek şifreli metin Y`yi ele geçirirse, ve $KU_b$`ye sahipse, aynı zamanda, KR_b ve X için bir erişim hakkına sahip değilse, X yada KR_b`yi elde etme girişiminde bulunacaktır. Üçüncü kişinin, şifreleme ve deşifreleme algoritmalarını bildiği varsayılır. Eğer, rakip sadece bu mesaj ile ilgileniyorsa, $\hat{X}$ şifreli metni üzerinde yapacağı hesaplamalarla, bir X düz metni oluşturmaya çabalayacaktır. Bununla beraber, rakibin teşebbüsü genellikle gelecek mesajları da okuyabilmek üzere, tahmini $\hat{K}R_b$ üstünde hesaplamalar yaparak KR_b'yi elde etmek olur.



Şekil 3.3. Açık anahtarlı şifrelemede kimlik doğrulama

Şekil 3.2.'de, güvenliğin sağlanması gösterilirken, Şekil 3.3.'te, açık anahtarlı kriptografinin kimlik doğrulamayı nasıl sağladığı gösterilmiştir:

$$Y = E_{KR_a}(X) \qquad [3.3]$$

$$X = D_{KU_a}(Y) \qquad [3.4]$$

Bu durumda, A, B`ye göndermek üzere bir mesaj hazırlar ve göndermeden önce mesajı kendi özel anahtarıyla şifreler. B, bu mesajı, (sadece) A`nın genel anahtarını

kullanarak deşifre edebilir. Çünkü, A mesajı kendi özel anahtarı ile şifrelemiştir dolayısıyla sadece, A bu mesajı hazırlayabilir. Bu yüzden özel anahtar ile şifrelenmiş tüm mesajlar dijital imza olarak düşünülebilir. Bununla birlikte, A`nın kendi özel anahtarı ile şifrelediği mesajın, A`nın özel anahtarına sahip olmayan bir kişi tarafından değiştirilmesi imkansızdır, dolayısıyla, mesajın içeriğinin de kontrol edilmesi ile, hem bütünlük hem de kaynak doğrulama ihtiyaçları karşılanmış olur.

Önceki yapıda, tüm mesaj şifrelenmiş gönderici ve mesajın içeriğinin güvenliği sağlanmıştı. Fakat bu, mesajın saklanması ve pratik kullanımı esnasında sorunlara neden olacaktır. Her doküman, pratik kullanım için düz metin halinde saklanmalıdır. Fakat orijinalliğinin ispat edilmesi gereken durumlarda kullanılmak üzere, mesajın şifrelenmiş hali de ayrıca saklanır. Bu işi başarmanın daha verimli bir yolu, mesajın en önemli olan bitlerinin şifrelenmesi olarak düşünülebilir.

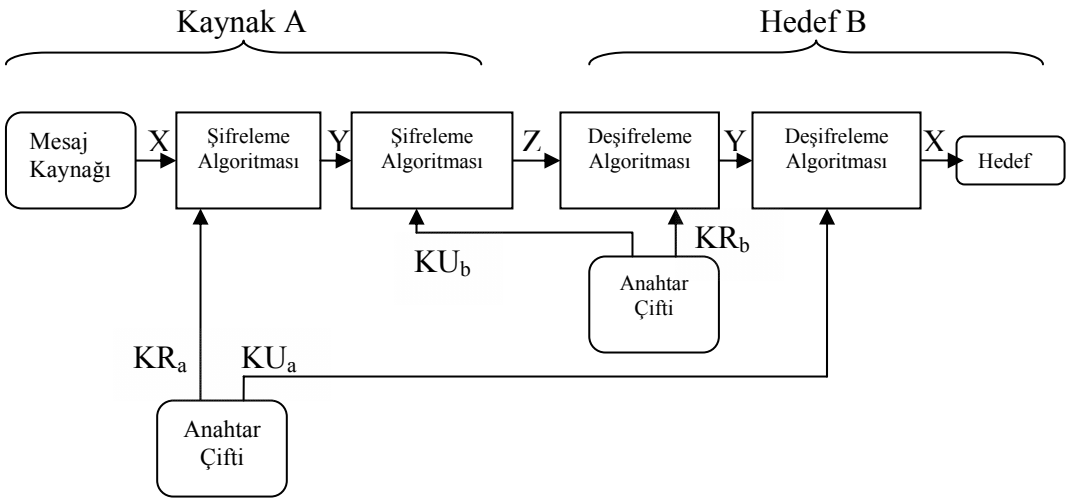
Örneğin öyle bir bit grubu olsun ki, bu kısım mesajın tanımlayıcısı olsun ve bu belirleyici bilinmeden/değiştirilmeden dokümanda bir değişiklik yapılması mümkün olmasın. Eğer bu belirleyici, göndericinin özel anahtarı ile şifrelenmişse, bu kısım, mesajın orijinalliğini, ardışıklığını ve içeriğini güvenlik altında tutan bir imza gibi düşünülebilir.

Bu çalışmada, geliştirilmesi planlanan protokolün kontrol amaçlı olacağı düşünüldüğünden bu türden uygulamalar konu dışında kalmaktadır dolayısı ile dijital imzalamanın bu yönü burada belirtilmişse de ayrıntılara girilmeyecektir

Bununla birlikte, hem kimlik doğrulama, hem de, gizlilik iki açık anahtar kullanılması ile sağlanır (Şekil 3.4.):

$$Z = E_{KU_b}\left\lfloor E_{KR_a}(X)\right\rfloor \qquad [3.5]$$

$$X = D_{KU_a}\left\lfloor D_{KR_b}(Z)\right\rfloor \qquad [3.6]$$



Şekil 3.4. İki açık anahtar kullanımı

Bu durumda, bir mesaj, şifrelenmeye başlanmadan önce, özel anahtar ile şifrelenir. Bu adım kimlik doğrulamayı sağlar. Daha sonra, bu yeni şifreli mesaj, alıcının genel anahtarı ile yeniden şifrelenir. Bu da gizliliği sağlar. Bu metodun dezavantajı iki kez şifrelenmiş olan metinin iki kez deşifrelenerek açılması esnalarında kaybedilecek fazladan zaman olarak düşünülür. Fakat mesaja sağladığı gizlilik ve kimlik doğrulama vazgeçilemez bir özelliktir.

3.3. Açık Anahtarlı Şifreleme Sisteminin Uygulamaları

Daha öncede belirtildiği gibi açık anahtarlı sistemler karakteristik olarak, birisi gizli tutulan, diğeri ise genel kullanım için açılmış olan iki anahtarla çalışan kriptografik algoritmalar kullanırlar. Uygulamaya bağımlı olarak, gönderici ya kendisinin özel anahtarını, ya alıcının genel anahtarını ya da ikisini birden, kimi kriptografik fonksiyonları gerçekleştirmek için kullanır. Geniş bir bakış açısı ile, açık anahtarlı kripto sistemlerin kullanımını üç kategoride incelenebilir:

* Şifreleme/Deşifreleme: Gönderici, bir mesajı alıcının genel anahtarı ile şifreler.
* Dijital İmzalama: Gönderen, mesajı kendi özel anahtarı ile imzalar. Bu imzalama, tamamen özel bir mesaj veya gönderilmesi planlanan mesajın tamamı yada küçük bir kısmı kullanılarak yapılır.

*Anahtar Değişimi: İki taraf ortaklaşa bir oturumda anahtarlarını değiş tokuş ederler. Bu işlem için bir çok farklı yöntem izlemek mümkündür.

Kimi algoritmalar, bu özelliklerden sadece bir yada iki tanesini gerçekleştirebilirken, bazıları bunların tümünü gerçekleştirebilir. Çizelge 3.2. kimi açık anahtarlı algoritmaların bu özelliklerden hangilerini desteklediğini göstermektedir.

Çizelge 3.2. Açık anahtarlı kripto sistemler için uygulamalar

Algoritma	**Şifreleme / Deşifreleme**	**Dijital İmza**	**Anahtar Değişimi**
RSA	Evet	Evet	Evet
Diffie-Hellman	Hayır	Hayır	Evet
DSS	Hayır	Evet	Hayır

3.4. Açık Anahtarlı Şifreleme için Gereklilikler

Çizelge 3.2.'de, iki benzer anahtarı temel alan kriptografik algoritmalar ifade edilmiştir. Diffie ve Hellman, bu algoritmaların varlığını göstermeksizin bu sistemi varsaymışlardır. Bununla birlikte, bu algoritmaların yerine getirmeleri gereken durumlar şöyle ifade edilir:

1. Bir B için, anahtar parçalarını (genel anahtar ve özel anahtar) yaratmak, hesapsal olarak kolay olmalıdır.
2. Gönderenin (A), mesajı göndereceği kişinin (B) genel anahtarını bildiği ve şifrelenecek olan mesajı (M) bildiği durumda, uygun şifreli metni yaratmak hesapsal olarak kolay olmalıdır.

$$C = E_{KU_b}(M) \qquad [3.7]$$

3. Alıcı B`nin, özel anahtarını kullanarak, şifrelenmiş mesajı orijinal haline getirmesi hesapsal olarak kolay olmalıdır.

$$X = D_{KR_b}(C) = D_{KR_b}\left[E_{KU_b}(M)\right] \qquad [3.8]$$

4. Herhangi bir üçüncü kişi için, genel anahtarı bilerek, özel anahtarı bulması hesapsal olarak imkansız olmalıdır.

5. Herhangi bir rakip için, genel anahtarı, şifreli metini (C) bilerek orijinal mesajı (M) elde etmesi hesapsal olarak imkansız olmalıdır.

Bunlara ek olarak, yararlı olmasına rağmen gerekli olmayan altıncı bir madde eklenebilir ki:

6. Şifreleme ve deşifreleme fonksiyonları her iki sıra ile de uygulanabilir olmalıdır.

$$M = E_{KU_b}\left[D_{KR_b}(M)\right] \qquad [3.9]$$

Bunlar gerçekleştirilmesi zor gerekliliklerdir bu yüzden, açık anahtarlı şifreleme fikrinin ileri sürüldüğünden bu yana geçen yıllar süresince sadece tek bir algoritma geniş bir kitle tarafından kabul edilmiştir.

Bu kadar zor gerekliliklerin istenmesinin sebeplerinin açıklanmasından önce, tek yönlü fonksiyonun açıklanması yerinde olacaktır. Söz konusu olan tek yönlü fonksiyonun bire-bir olduğu bir aralıkta, tersini hesaplamak imkansız iken, fonksiyonun kendisinin hesaplanması kolaydır.

$Y = f(X)$ çok kolay,
$Y = f^{-1}(X)$ imkansız...

Genellikle, "kolay" dan kasıt, fonksiyonun girdi uzunluğuna bağlı olarak polinomal bir zaman süresi içerisinde çözülebilir olmasıdır. Şöyle ki, eğer girdi uzunluğu *n* bit kadarsa, fonksiyonun hesaplanması için gereken süre *a* bir sabit sayı iken, n^a gibi bir fonksiyonla orantılı olmalıdır. "imkansız" tanımı itibari ile açık olmasa da, bir problemin çözümünün olanaksız olmasından, giriş büyüklüğüne bağlı olarak çözüm için harcanan çabanın, polinomal zamandan daha hızlı arttığı durum anlaşılır. Örneğin, girdi *n* bit ile gösterilirken, fonksiyonun çözülme zamanı 2^n gibi bir

fonksiyona bağlı olarak artıyorsa, bu fonksiyonun çözümünün imkansız olduğunu düşünülebilir.

3.5. RSA Algoritması

Diffie ve Hellman tarafından hazırlanmış öncü bir makale 1976 yılında, gizli anahtarlı sistemlerin gerekliliklerini yerine getiren bir kriptografik algoritmada uzmanlaşan kriptolojistlere karşı yeni bir yöntem tanıtmıştır. Bu yeni yönteme karşı verilen yanıtlardan ilki; 1977'de MIT'de Ron Rivest, Adi Shamir, ve Len Adleman (RSA) tarafından ortaya atılmıştır ve ilk olarak 1978'de basılmıştır (5). Rivest-Shamir-Adleman (RSA) yapısı; üstünlüğü kabul gördüğünden itibaren, geniş çapta tek olarak kabul edilmiş ve genel anahtarlı şifreleme yönteminin genel amacını yerine getirmiştir.

RSA yapısı itibari ile, birtakım n`ler için 0 ve (n-1) arasındaki tamsayılardan oluşan şifreli ve düz metnin kullanıldığı bir blok şifrelemedir. Bu bölüme RSA algoritmasının açıklanmasıyla başlanacak daha sonra RSA`nın hesapsal ve kripto analitik anlamları incelenmeye çalışılacaktır.

3.5.1. RSA algoritmasının tanımı

Rivest, Shamir, ve Adleman tarafından ortaya konulan yapı; düz metnin, blokların içinde şifrelenmesini esas alır. Her blok, bir ‘n’ sayısından daha az bir ikili değere sahiptir. Bloğun büyüklüğü, $\log_2(n)$’e eşit veya ondan daha küçük olmalıdır; pratikte, blok büyüklüğü 2^k bittir, n ise $2^k < n \leq 2^{k+1}$ aralığındadır. Şifreleme ve deşifreleme; e ve d anahtarları oluşturmak için seçilen asal sayılar olmak üzere, düz metin bloğu *M* ve şifreli metin bloğu *C* için şu şekildedir:

$$C = M^e \bmod n \qquad [3.10]$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n \qquad [3.11]$$

Hem gönderen hem de alıcı n’in değerini bilmelidir. Gönderen e’nin değerini bilir, ve sadece alıcı d’nin değerini bilir. Böylece; $KU = \{e, n\}$ bir genel anahtar, ve

$KR = \{d, n\}$ bir özel anahtar olur ve bu bir genel anahtarlı şifreleme algoritmasıdır. Genel anahtarlı şifreleme için tatmin edici olması bakımından, bu algoritma aşağıdaki gereklilikleri yerine getirmelidir:

1. $M < n$ olduğu koşulda, $M^{ed} = M \bmod n$ iken; e,d,n değerlerini bulmak mümkün olmalıdır.

2. $M < n$ koşulunu sağlayan tüm M değerleri için, M^e ve C^d hesaplanması nispeten kolay olmalıdır.

3. Yalnız e ve n verildiğinde, d'nin hesaplanması imkansız olmalıdır.

İlk gerekliliğin öngördüğü üzere aşağıdaki yapı için bir ilişki bulmamız gerekiyor:

$$M^{ed} = M \bmod n \qquad [3.12]$$

Euler`in teoremine göre, verilen iki asal sayı p ve q, ve iki tamsayı n ve m olmak üzere, $n = pq$ ve $0 < m < n$ olduğu durumda, keyfi seçilmiş bir k tamsayısı seçilmiş sayılar ile şöyle bir ilişki oluşturur:

$$m^{k\Phi(n)+1} = m^{k(p-1)(q-1)+1} \equiv m \bmod n \qquad [3.13]$$

Buradaki $\Phi(n)$ fonksiyonunun döndürdüğü değer, n'den küçük olan ve n ile aralarında asal olan tam sayıların, miktarıdır. p ve q asal sayı olmak üzere $\Phi(pq) = (p-1)(q-1)$ olur. Böylece aşağıdaki eşitlik sağlanıyorsa birinci gereklilikte istenilen ilişkiye ulaşılabilir:

$$ed = k\Phi(n) + 1 \qquad [3.14]$$

Bu durumda aşağıdaki denkliklerden söz edilebilir:

$$ed \equiv 1 \bmod \Phi(n) \qquad [3.15]$$

$$d \equiv e^{-1} \bmod \Phi(n) \qquad [3.16]$$

e ve d, $\bmod \Phi(n)$' in çarpmaya göre tersidir. Dikkat edilecek olursa, modüler aritmetiğin kurallarına göre, bu denkliğin doğru olması yalnız d'nin (ve sonucunda e'nin) $\Phi(n)$ ile aralarında asal olması durumunda mümkündür. Bu durumda, gcd($\Phi(n)$, d) = 1 (Ortak bölenlerin en büyüğü 1' e eşit) demektir.

Gerekliliklerin açıklanmasının ardından, RSA yapısının çıkarılmasına başlanabilir. Yapının bileşenleri sırasıyla şöyledir :

p,q; iki asal sayı (gizli, seçilmiş)
$n = pq$; (genel, hesaplanmış)
e; gcd($\Phi(n)$, e) = 1; $1 < e < \Phi(n)$ olacak şekilde (genel, seçilmiş)
$d \equiv e^{-1} \bmod \Phi(n)$; (gizli, hesaplanmış)

{d,n} çifti özel anahtarı, {e,n} çifti ise genel anahtarı oluşturur. Bir *B* kullanıcısının, *A* kullanıcısına, *A* kullanıcısının genel anahtarını kullanarak bir *M* mesajını göndermek istediğini varsayalım. Bu durumda *B*, $C = M^e \pmod n$ formülü yardımı ile *C* şifreli mesajını elde edecek ve bunu *A'*ya gönderecektir. *A* kullanıcısı bu mesajı aldığında, $M = C^d \pmod n$ formülü ile mesajı deşifre edecektir. Bu algoritmanın açıklaması işe şu şekilde yapılabilir:

Yukarıda belirtildiği gibi, e ve d aşağıdaki denkliği sağlayacak şekilde seçilmişti:

$$d \equiv e^{-1} \bmod \Phi(n) \qquad [3.17]$$

Bunun sonucunda,

$$ed \equiv 1 \bmod \Phi(n) \qquad [3.18]$$

Bu yüzden ed, $k\Phi(n)+1$`in bir formudur. Bu denkliğin Euler teoreminin bir sonucu olduğu, iki asal sayı olan *p* ve *q*, birer tamsayı olan *n* (*n* = *pq*) ve *M* (0 < M < n) alınarak kolaylıkla ispatlanabilir.

$$M^{k\Phi(n)+1} = M^{k(p-1)(q-1)+1} \equiv M \pmod n \qquad [3.19]$$

Dolayısıyla, $M^{ed} = \bmod(n)$;

$$\begin{aligned} &C = M^e \bmod n \\ &M = C^d \bmod(n) \equiv (M^e)^d \bmod(n) \equiv M^{ed} \bmod(n) \equiv M \bmod n \end{aligned} \qquad [3.20]$$

olduğu görülür.

Anahtar Üretimi

* p ve q sayıları seçilir (p,q asal sayılar)
* n= px q sayısını hesapla
* Φn = (p-1)(q-1) sayısını hesapla
* e tamsayısını seç (OBEB(Φn, e)=1); 1< e < Φn
*d tamsayısını hesapla $d = e^{-1} \bmod(\Phi n)$
* Açık Anahtar : KU = [e , n]
* Özel Anahtar : KR = [d, n]

Şifreleme

*Düz-Metin: M < n
*Şifreli – Metin $C = M^e \pmod n$

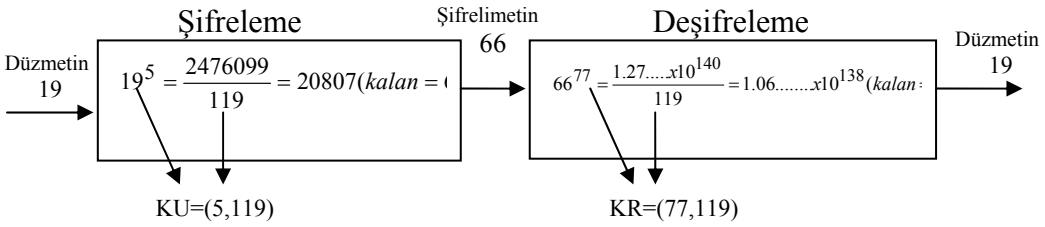
Deşifreleme

* Şifreli – Metin C
* Düz – Metin $M = C^d \pmod n$

Şekil 3.5. RSA algoritmasının yapısı

Şekil 3.5.'te, RSA algoritması özetlenmiştir ve Şekil 3.6.'da bir örnek verilmiştir. Bu örnekteki anahtarlar şu şekilde yaratılmıştır:

1. İki adet asal sayı oluşturulur, p=7 ve q=17
2. n değeri hesaplanır, $n=pq$=7x17=119.
3. $\Phi(n)$ değeri hesaplanır, $\Phi(n) = (p-1)(q-1) = 96$
4. $\Phi(n)$'den küçük olacak ve $\Phi(n)$ ile aralarında asal olacak bir e sayısı seçilir, e=5.
5. Son olarak, $de = 1 \bmod 96$ ve $d < 96$ şartını sağlayan bir d tamsayısı belirlenir. Belirlenen diğer sayılar göz önüne alındığında doğru değer, d = 77 olmalıdır. Çünkü, 77x5 = 385 =4x96+1'dir.



Şekil 3.6. RSA Şifrelemesi için bir örnek

Bu işlemler sonucunda elde edilen değerlere göre genel anahtar $KU = \{5,119\}$, özel anahtar da $KR = \{77,119\}$ olarak bulunur. Bu örnek, anahtarların M=19 düz metni için kullanımını gösterir. Şifreleme için, 19'un beşinci kuvveti alınarak 2476099 sayısı elde edilir. Daha sonra bu sayının 119 ile bölümünden kalan bulunur ve şifreli metin "66" sayısı elde edilir. Benzer şekilde deşifreleme, $66^{77} \equiv 19 \bmod 119$ işlemi yapılarak "19" yeniden elde edilir.

3.6. Hesaplama Yöntemleri

Bu bölümde, RSA algoritmasının kullanımı için gerekli hesaplamanın karmaşıklığıyla ilgili önemli hususlar incelenecektir. Aslında düşünülmesi gereken iki önemli nokta vardır: anahtar üretimi ve şifreleme/deşifreleme. Öncelikle şifreleme/deşifreleme işlemleri incelenecek ve daha sonra anahtar üretimi konusuna dönülecektir.

3.6.1. Şifreleme ve deşifreleme

RSA`da hem şifreleme hem de deşifreleme, tamsayıların tamsayı kuvvetlerini almayı ve mod alma işlemlerini gerektirir. Eğer ilk önce tamsayıların üsleri alınıp, daha sonra *mod n* ile indirgersek, ara değerler devasa büyüklükte sayılar olurlar. Bu sorunu bir nebze azaltmak için modüler aritmetiğin önceki bölümde de belirtilen şu özelliğinden yararlanılabilir:

$$\left[(a \bmod n)x(b \bmod n)\right] \bmod n = (axb) \bmod n$$

Bu sayede, ara değerler modül *n*'e göre indirgenebilir. Bu da hesaplamayı pratik hale getirir. Bir diğer husus, üssün verimidir. Çünkü, RSA' da potansiyel olarak büyük üsler ile işlem yapılır. Verimin ne kadar artabileceğini görmek üzere, x^{16}'nın hesaplanmak istendiğini varsayılsın. Normal bir yöntem 15 çarpım gerektirir:

$$x^{16} = x*x*x*x*x*x*x*x*x*x*x*x*x*x*x*x$$

Bununla birlikte, aynı sonuca, her bir kısmi sonucun karesini alınarak; x^2, x^4, x^8, x^{16} olacak şekilde dört adımda da ulaşılabilir.

Daha genel olarak a^m değerinin (*a* ve *m* birer pozitif tam sayı) hesaplanmak istendiğini varsayalım. Eğer *m* tamsayısı $b_k, b_{k-1}, ..., b_0$ gibi ikilik sistemde ifade edilecek olursa:

$$m = \sum_{bi \neq 0} 2^i \qquad [3.21]$$

olur. Böylece,

$$a^m = a^{\sum_{bi\neq 0}\gamma} = \prod_{bi\neq 0} a^{(2^i)} \qquad [3.22]$$

$$a^m \bmod n = \left[\prod_{bi\neq 0} a^{(2^i)}\right] \bmod n = \prod_{bi\neq 0}\left[a^{(2^i)} \bmod n\right]$$

eşitlikleri yazılabilir.

Bu sonuç sayesinde, a^b mod *n* işlemini hesaplamak üzere aşağıdaki algoritma geliştirilebilir. Algoritmanın altındaki verilen çizelgede algoritmanın çalışmasını örneklemektedir. *c* değeri aslında gerekli olmasa da son değeri üssün değerine eşit olacağından dolayı açıklayıcı bir niteliktedir.

```
c ← 0; d ← 1
for i ← k downto 0
    do c ← 2 x c
       d ← (d x d)mod n
       if bi=1
          then c ← c+1
               d ←  (d x a)mod n
return d
```

Çizelge 3.3. Mod alma algoritmasının çalışması

i	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
d	7	49	157	526	160	241	298	166	67	1

3.6.2. Anahtar üretimi

Açık anahtarlı şifreleme sisteminin uygulamasından önce, her iki katılımcı anahtar parçalarını üretmelidir. Bu üretim işlemi aşağıdaki vazifeleri ihtiva eder:

* İki asal sayı hesaplanması, *p* ve *q*
* *e* ya da *d*'nin seçilip değerinin hesaplanması.

Öncelikle, *p* ve *q* nun seçimi düşünülür çünkü, herhangi bir potansiyel saldırgan *n=pq*'nun değerini biliyor olacaktır, birtakım özel metotlarla *p* ve *q'* nun bulunmasını engellemek için, *p* ve *q* sayıları yeterince büyük bir seriden seçilmiş olmalıdırlar (*p* ve *q* büyük sayılar olmalıdır). Diğer yandan, büyük asal sayıları bulmak için kullanılan yöntem yeterince verimli olmalıdır.

Günümüzde, verimli ve büyük asal sayılar üreten kullanışlı standart bir teknik yoktur. Genel olarak kullanılan yöntem ise şöyle çalışmaktadır: istenen büyüklük aralığında rasgele bir tek sayı seçilir ve bunun asal olup olmadığı kontrol edilir. Eğer asal değilse, bu işlem asal bir sayı buluncaya kadar sürdürülür.

Asallığı kontrol eden bir çok test geliştirilmiştir. Bu testlerin neredeyse tümü yaklaşıklıktan bahseder yani test, söz konusu (yeterince büyük) tam sayının muhtemelen asal olup olmadığını belirler (6 – 14). Bu eksikliğine karşın, testler,

sayının asal olma olasılığının 1.00' a çok yakın olduğu durumlarda sorunsuz çalışabilirler. Örnek olarak, en verimli ve popüler test algoritmalarından birisi Miller-Rabin algoritmasıdır. Bu algoritmada ve diğer bir çok algoritmada, *n* sayısının asallığını kontrol etmek için, *n* ile bir takım işlemlere sokulmak üzere *n* den küçük rasgele bir *a* sayısı seçilir. Eğer *n* testi geçemezse bunun anlamı *n* sayısının asal olmadığıdır. Eğer *n* testi geçerse bu durumda sayı asal olabilir veya olmayabilir. Eğer *n* sayısı bu gibi çok fazla sayıda testten başarılı olursa, *n* için muhtemelen asaldır denir.

Özet olarak, asal sayı kontrol prosedürü aşağıdaki adımları izler:

1. Rasgele bir tek tam sayı "*n*" seçilir.
2. $a < n$ koşulunu sağlayan rasgele bir *a* sayısı seçilir.
3. *n* için asallığı yaklaşık olarak test eden Miller-Rabin gibi bir test gerçekleştirilir, eğer *n* testi geçemezse birinci adıma geri dönülür.
4. Eğer *n* bir çok sayıda testi başarı ile geçmişse *n* asal olarak kabul edilir yada ikinci adıma geri dönülür.

Bu, oldukça uzun süren ve tekdüze bir işlemdir fakat bu işlem sadece yeni bir asal sayıya ihtiyaç duyulduğunda yani yeni bir genel veya özel anahtar (KU, KR) gerektiğinde uygulanır.

Asal bir sayı bulununcaya kadar geçen sürede asallığı reddedilen sayıların miktarı önemli bir ayrıntıdır. Asal sayılar teorisine göre bir *N* asal sayısına yakın olan ilk asal sayı, bu sayıya ortalama *ln N* adet tam sayı uzaklıktadır. Bu durumda, bir asal sayı bulunduğunda, bir sonraki asal sayı yaklaşık olarak *ln N* tamsayı ötede olacaktır. Aynı zamanda bu aralıkta bulunan çift sayıların asal olmadığı kesin olduğundan, bir asal sayıya ulaşılması için en fazla *ln N/2* deneme yapılması gerekir. Örneğin 2200 sayısını merkez alarak asal sayı aramaya başlayan bir kişinin bir asal sayıya ulaşmak için sayı ekseninde bir yöne doğru yaklaşık *ln(2200)/2* = 70 sayı denemesi gerekir.

p ve *q* değerlerine sahip olunmasından sonra bir *e* değeri seçilmesi ve *d* değerinin hesaplanması ya da alternatif olarak *d* değeri seçilerek *e* değerinin hesaplanmasının ardından anahtar üretme işlemi tamamlanmış olur. *e* sayısı seçilirken, bu sayının

OBEB($\Phi(n), e$) = 1 eşitliğini sağlaması yani $\Phi(n)$ ile aralarında asal olması gerektiği dikkate alınmalıdır. Daha sonra da, $d = e^{-1} \bmod \Phi(n)$ eşitliği kullanılarak *d* değeri elde edilir. Neyse ki, aynı anda iki tam sayının en büyük ortak bölenini bulan, ve bu bölen 1 ise, bu sayılardan birisinin tersini, diğerinin modülüne göre hesaplayan tek bir algoritma vardır. Bu algoritma genişletilmiş Öklit algoritması olarak anılır. Prosedür, yarattığı seri halindeki rasgele sayıların her birisini, $\Phi(n)$ ile arasında asal olanı bulununcaya kadar teker teker dener. Bu noktada $\Phi(n)$ ile aralarında asal olacak şekilde kullanışlı bir sayı bulmak için kaç rasgele sayının test edilmesi gerektiği merak edilebilir. Rasgele seçilmiş iki sayının aralarında asal olma olasılığı 0.6`dır. Bu da demek oluyor ki, aranılan tam sayıya ulaşmak için bir kaç test yapmak yeterli olur.

3.7. RSA Algoritması'nın Güvenliği

RSA algoritmasına saldırı için kullanılabilecek üç temel metot aşağıdaki gibidir:

* Eksiksiz Arama (Brute-force) Atakları: Özel anahtarın tüm alternatifler için tek tek denenmesi ile gerçekleştirilir.
* Matematik Atakları: Birkaç farklı yöntem olmakla beraber, hepsinin temel amacı n çarpımını oluşturan iki asal sayıyı bulmaktır.
* Zaman Atakları: Deşifreleme algoritmasının çalışması esnasında geçen zamana bağlıdır.

Brute-force yöntemine karşı RSA'nın savunması diğer kripto sistemlerin çözümünden farklı değildir. Bu çözüm geniş anahtar uzayı kullanmaktır. *e* ve *d*`nin bit sayıları ne kadar büyük olursa bu ataktan korunma o kadar iyidir. Bununla beraber, hem anahtar üretimi hem de şifreleme/deşifreleme işlemleri için algoritmanın içerdiği hesaplamalar karmaşıklaşacak, büyük anahtarlarla çalışan sistemin de çalışma zamanından kaybı olacaktır.

3.7.1. Çarpan problemi

RSA`ya karşı gerçekleştirilebilecek matematik atak yöntemleri üç başlıkta incelenebilir:

* n'in iki asal çarpanının bulunması. Bu sayıların bulunmasıyla $\Phi(n) = (p-1)(q-1)$ hesaplanabilir ve dolayısıyla $d = e^{-1} (\mathrm{mod}\, \Phi(n))$ hesaplanabilir.

* $\Phi(n)$'in p ve q bulunmadan, doğrudan hesaplanması. Aynı şekilde buradan $d = e^{-1} (\mathrm{mod}\, \Phi(n))$ hesaplanabilir.

* d'nin, $\Phi(n)$ hesaplanmadan hesaplanması.

RSA`ya yönelik en çok tartışılan kripto analiz yöntemi, n sayısını kendisini oluşturan iki asal çarpanına ayırmaktır. Saptanmış bir n sayısı için $\Phi(n)$ hesaplanması, n' in çarpanlarına ayrılması demektir. Günümüzde, verilen n ve e sayıları için d değerini hesaplayan algoritmalar üs alma problemiyle birlikte hesaplamaya harcanan zaman ile ilgili bir problem yaratmaktadırlar. Bu bakımdan çarpanlara ayırma algoritmasının performansı, RSA`nın güvenliği için bir tehdit olarak düşünülebilir.

Çok büyük asal sayılardan oluşturulmuş çok büyük bir n sayısını çarpanlarına ayırmak oldukça zor bir işlem olacaktır; fakat bu n'in kullanımından daha büyük bir zorluk değildir. 1977 yılında, RSA`nın üç yaratıcısı, Scientific American okuyucularına, derginin Martin Gardner tarafından hazırlanan "Matematik Oyunları" kısmına, deşifre edilmesi için RSA ile şifrelenmiş bir metin koydular. Bu metni deşifre ederek getirecek olan kişiye de $100 ödül vereceklerini, fakat bu işlemin 40 milyar yıldan uzun süreceğini söylediler. 1994 Nisanında, internet üzerinden çalışan bir grup, sadece 8 aylık bir çalışma ile ödülü almaya hak kazandı. Bu meydan okumada kullanılan genel anahtar 129 rakam yani yaklaşık 428 bitti. RSA Laboratuarları, 100, 110, 120 ve bu şekilde artan anahtar boyutları ile şifrelenmiş metinler için meydan okumaları sürdürdü. En son sonuçlanan iddia da 130 rakamdan oluşan anahtar ile şifrelenmiş metnin çözülmesi oldu. Çizelge 3.4. tarihlerine göre alınmış sonuçları listelemektedir. Sarf edilen efor kolonu MIPS birimi ile ifade edilmektedir ve 1 MIPS-yılı, saniyede 1 milyon işlem yapan bir işlemcinin bir yıl

süre ile çalışması anlamına gelir. Bu durumda, yaklaşık $3x10^{13}$ adet işlem gerçekleştirilmiş olmaktadır. 200 Mhz.' lik bir Pentium işlemci yaklaşık 50-MIPS`lik kapasiteye sahiptir.

Çizelge 3.4. n sayısının çarpanlarına ayrılması için bazı algoritmaların analizi

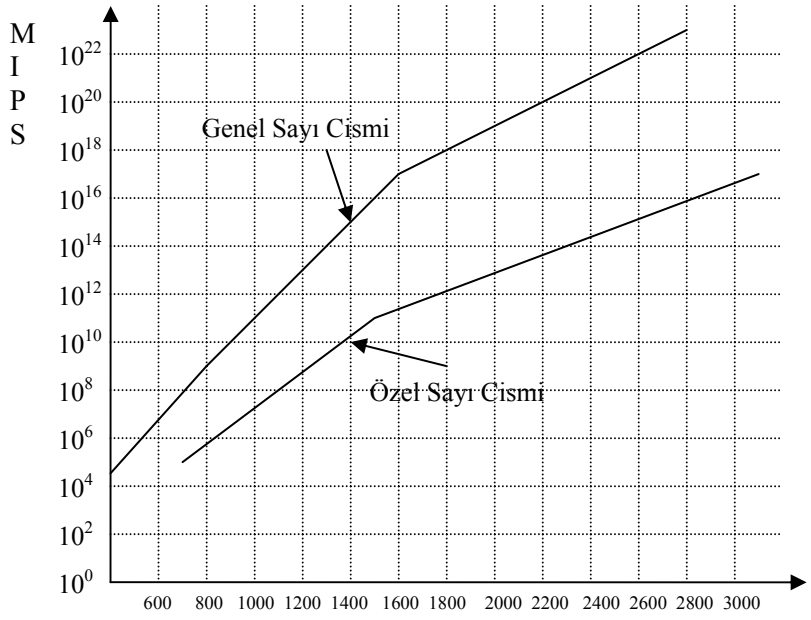
Anahtarın Rakam Sayısı	Yaklaşık Bit Sayısı	Verinin Elde Edilmesi	MIPS-yılı	Algoritma
100	332	Nisan 1991	7	Kuadratik eleme
110	365	Nisan 1992	75	Kuadratik eleme
120	398	Haziran 1993	830	Kuadratik eleme
129	428	Nisan 1994	5000	Kuadratik eleme
130	431	Nisan 1996	500	Genelleştirilmiş sayı alanı elemesi

Son zamanlara kadar, çarpanlara ayırma atakları quadratik eleme adı verilen bir yöntem kullanılarak yapılmakta idi. RSA-130 için yapılan atakta ise daha yeni bir algoritma; "Genelleştirilmiş sayı cismi elemesi (GNFS)" kullanılmıştır. Bu sayede, RSA-129' dan daha büyük bir sayının çarpanlarına ayrılması 10%'luk bir çaba ile gerçekleştirilebildi.

Bilgisayar teknolojisinin hızla gelişmesi ve çarpanlara ayırma algoritmalarının zamanla optimize edilerek daha hızlanmaları neticesinde, büyük anahtar boyutları kullanmanın verdiği gözdağı yavaş yavaş kriptoanalistler için önemsizleşmektedir.

Çizelge 3.4.'den de, bir algoritma değişikliğinin ne kadar etkili bir hız artışı sağladığı görülebilir. Üstelik, GNFS üzerinde yapılacak bir optimizasyon sonucunda çok daha

etkin bir algoritma oluşturulması mümkündür. Aslında "Özel Sayı Cismi Elemesi (SNFS)" adında bir algoritma, sayıların çarpanlarını GNFS' ye göre çok daha hızlı bulmaktadır. Şekil 3.7.'de bu iki algoritmanın performans karşılaştırması görülebilir. SNFS kadar yada ondan daha hızlı çalışabilecek bir genel çarpanlara ayırma algoritması her an geliştirilebilir. Bu yüzden RSA`da kullanılacak anahtar boyutu seçilirken özen gösterilmelidir.



Şekil 3.7. Özel sayı cismi elemesi ve genel sayı cismi elemesi yöntemlerinin performans karşılaştırılması

Üretilen *n* sayılarının asal çarpanlarına kolayca ayrılmaması için, *p* ve *q* sayıları seçilirken aşağıda belirtilen bazı kısıtlamaların göz önünde bulundurulması gerekir.

1. *p* ve *q* sayılarının uzunlukları birbirinden sadece birkaç rakam farklı olmalıdır.
2. Hem (*p*-1) hem de (q-1) büyük bir asal çarpan içermelidir.
3. OBEB(*p*-1,*q*-1) küçük olmalıdır.

Ayrıca, ispatlandığı üzere, eğer $e<n$ ve $d<n^{1/4}$ ise, *d* kolaylıkla hesaplanabilir.

3.7.2. Zaman atakları

Bir kriptografi uzmanı olan Paul Kocher, bir bilgisayarın şifreli bir metni çözerken harcadığı zaman dilimlerinden yararlanılarak o metnin oluşturulması esnasında kullanılan özel anahtarın hesaplanabileceğini kanıtlamıştır (15). Zaman atakları sadece RSA için değil, diğer açık anahtarlı kriptografik sistemler içinde uygulanabilen bir saldırı metodudur.

Zaman atağı, bir hırsızın, soymak istediği kasanın parola kombinasyonunu tahmin etmek için, o kasayı açan bir kişinin çevirdiği her bir numaranın ne kadar sürdüğünü izlemesine oldukça benzemektedir. Bu atak yöntemi, modüler üs alma algoritması kullanılarak açıklanabilir fakat, atak yöntemi belirlenmiş zamanlarda çalışmayan herhangi bir uygulamaya da adapte edilebilir. Bu algoritmada, her adımda, modüler üs alma işlemi bit-bit ve bir modüler çarpım işlemi gerçekleştirilmektedir. Ayrıca, ekstra bir modüler çarpım işlemi de değeri 1 olan her bit için yapılır.

Kocher`in de makalesinde belirttiği gibi, bu atağı anlamak son derece kolaydır (15). Hedef sistemin çoğu koşulda çok hızlı çalıştığı fakat bazı girdiler için normalden biraz daha yavaş çalışan bir modüler çarpım algoritması kullandığını farz edelim. Atak, en soldaki bitten, b_k'dan başlayarak bit bit ilerlemeye devam eder. İlk *j* bitin bilindiği farz edilirse verilen bir şifreli metin için atağı gerçekleştiren kişi, ilk *j* adımı *for* döngüsü ile bitirebilir. İşlemin sonradan gelen adımları bilinmeyen üs bitine bağlıdır. Eğer bit set edilmişse, d← (d x a) mod n işlemi çalışır. *a* ve *d*`nin bazı değerleri için modüler çarpım son derece yavaş olur ve atağı gerçekleştiren kişi bu bitlerin değerinin ne olduğunu anlar. Bu yüzden, eğer deşifreleme algoritmasının çalışma zamanı dikkatle gözlenirse, bu algoritma parçası her 1 bit için çalışma zamanını yavaşlatır ve o an üzerinde çalıştığı bitin değerinin 1 olduğu tahmin edilebilecektir. Aynı şekilde algoritmanın anlık çalışma zamanı hızlıysa o anda üstünde çalışılan bitin değerinin 0 olduğu anlaşılır.

Pratikte, modüler üs alma tanımlamaları tüm algoritmanın çalışma zamanı üzerinde bu kadar büyük değişiklikler yaratmamaktadır, yinede algoritmalar içerisinde, bu atağı pratik hale getirecek yeterince materyal bulunmaktadır.

Zaman atakları önemli bir tehlike arz etmektedir. Atağı önlemek için çok basit önlemler alınabilir:

Sabit üs alma zamanı: Tüm üs alma fonksiyonlarının çalışma zamanı eşit hale getirilebilir. Fonksiyonun çalışması erken bitse dahi, geriye bir değer döndürmesi bir miktar bekletilerek hepsinin çalışma zamanı belli bir değerde sabitlenebilir. Bu çok basit bir önlemdir fakat, performansı kötü yönde çok fazla etkileyecektir.

Rasgele Gecikme: Daha iyi bir performans, üs alma algoritmasına rasgele bekleme süresi eklenerek zaman atağını yanıltılmasıyla elde edilebilir. Kocher, bu rasgele değerlerin dikkatsizce seçilmesi durumunda, zaman atağı gerçekleştiren kişinin biraz daha fazla gayret göstererek yine başarıya ulaşmasının mümkün olacağına dikkat çekmektedir (15).

Körleştirmek: Üs alma işleminin gerçekleştirilmesinden önce, şifreli metin parçasını rasgele bir sayı ile çoğaltarak zaman atağının gerçekleştiren kişinin sağlıklı bir veri elde etmesi engellenebilir. RSA veri güvenliği (data security) kuruluşunun geliştirdiği bir koruma yöntemi mevcuttur. RSA veri güvenliği, bu atak yönteminin toplam performansa 2% ila 10%'luk bir kayıp getirdiğinin de altını çizmektedirler.

4. ANAHTAR YÖNETİMİ

Açık anahtarlı kriptografinin en önemli özelliklerinden birisi, anahtar dağıtımı problemine getirdiği yeniliktir. Açık anahtarlı şifreleme kullanmak için iki çok önemli neden vardır:

* Açık anahtarların dağıtımı.
* Gizli anahtarların dağıtımı için açık anahtarlı şifreleme kullanımı.

4.1. Açık Anahtarların Dağıtımı

Açık anahtarların dağıtılması için kullanılan birkaç farklı teknik vardır. Neredeyse tüm önerilmiş yöntemler aşağıdaki gibi gruplandırılabilir:

* Genel duyuru,
* Herkes tarafından erişilebilir adres rehberi,
* Açık anahtar yetkilisi,
* Açık anahtar sertifikası.

4.1.1. Açık anahtarların duyurulması

RSA gibi bir açık anahtarlı bir şifreleme algoritmasının kullanımı tamamıyla kabul edilmişse açık anahtar bir başkasına gönderilebilir veya bütün iletişim ağına duyurabilir. Örneğin, RSA algoritmasını kullanan PGP`nin (Pretty Good Privacy) popülaritesinin artması sonucu bir çok PGP kullanıcısı, herkese açık forumlara, USENET haber gruplarına veya diğer mail listelerine attıkları mesajların sonuna açık anahtarlarını eklemeyi benimsemişlerdir.

Bu yöntem çok kolay ve bir o kadar kullanışlı olmasına karşın, büyük bir yetersizliği vardır. Örneğin herhangi birisi siz olduğunu iddia ederek açık anahtarını kişilere duyurabilir. Yani bir kullanıcı kendisini, doğru olmadığı halde, *A* kullanıcısı gibi tanıtabilir ve kendi açık anahtarını *A* kullanıcısının açık anahtarıymış gibi kişilere ilan edebilir. Gerçek *A* kullanıcısı sahtekarlığın farkına varana veya bir başka

kullanıcı onu uyarana dek, sahte *A* kullanıcısı gerçek *A* kullanıcısına gönderilmek üzere şifrelenmiş bütün mesajları okuyabilir.

4.1.2. Herkes tarafından erişilebilir adres rehberi

Herkesçe erişilebilir dinamik bir açık anahtar adres rehberinin, iyi bir şekilde korunması ve organize edilmesiyle, daha yüksek derecede güvenlik sağlanabilir. Adres rehberlerindeki anahtarların dağıtımı ve korunması güvenilir kimi kişi veya kuruluşların sorumluluğunda olmalıdır. Böyle bir hizmet tasarlanırken asgariyetle aşağıdaki koşullar sağlanmış olunmalıdır:

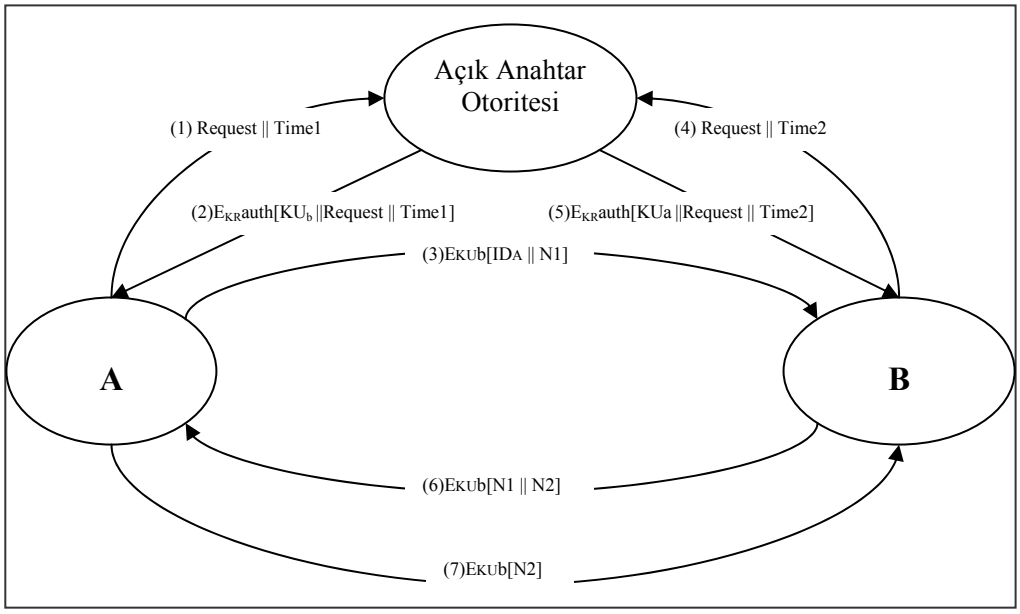
1. Adres rehberinin her elemanı için bulunacak {ad, açık anahtar} bölümleri iyi korunmalıdır.
2. Her kullanıcı rehber yetkilileri ile bir açık anahtar kaydetmeli ve bu işlemi yüz yüze ya da kimlik kontrolü ile güvenli hale getirilmiş bir iletişim metodu ile yapmalıdır.
3. Herhangi bir üye adres rehberindeki genel anahtarını, bu anahtarla ilişkili olan özel anahtarın herhangi bir sebepten ötürü güvenilirliğini yitirdiğini düşünmesinden ötürü istediği zaman bir yenisi ile değiştirebilmelidir.
4. Bu sistemin yönetimi, periyodik olarak adres rehberini güncellemeli, bir telefon rehberinde olduğu gibi isimleri ve genel anahtarların kopyasını saklamalı ve güncellemeleri geniş bir kitleye ulaşabilen yayın organı (gazete veya internet sitesi gibi) yoluyla diğer kullanıcılara haber vermelidir.
5. Üyeler aynı zamanda adres rehberine elektronik yolla da ulaşabilmelidirler. Bunun için de adres rehberinin yönetimi, gizlilik ve kimlik denetimi gerektiren güvenli bir iletişim metodu kullanılması zorunluluğunu yerine getirmelidir.

Bu yapının bireysel genel duyuru metodundan daha güvenli olduğu oldukça açıktır fakat halen kimi savunmasız noktalar bulunmaktadır. Eğer bir saldırgan, adres rehberi yöneticisinin özel anahtarını bir şekilde ele geçirir ya da onu hesapsal metotlarla bulmayı başarırsa, kolay bir şekilde güvenlik engellerini aşarak kullanıcıların açık anahtarlarını istediği açık anahtarlar ile değiştirip, onlara

gönderilen şifreli mesajları okuyabilir ve istediği kişiyi taklit ederek diğer üyelere onun adına mesaj atabilir.

4.1.3. Açık anahtar yetkilisi

Genel anahtar yönetimi için daha güçlü bir güvenlik, herkes tarafından erişilebilir adres defteri üzerinde daha sıkı bir kontrol uygulanması ile elde edilebilir. G. L. Popek ve C. S. Kline'ın 1979 yılında makaleleri baz alınarak oluşturulmuş senaryo Şekil 4.1.' de gösterilmiştir (16). Bu yapıda da, önceki yapıda olduğu gibi bütün üyelerin açık anahtarlarının saklandığı ve dağıtımının yapıldığı dinamik bir merkezi rehberin varlığı söz konusudur. Buna ek olarak, her üye, yöneticiye ait olduğunu bildikleri bir açık anahtara sahiptir. Şekil 4.1.'de numaralanmış olan adımlar ile anahtar dağıtımı gerçekleşir.



Şekil 4.1. Açık anahtar yetkilisi yardımıyla anahtar dağıtımı

1. *A* kullanıcısı, *B* kullanıcısının şu anki açık anahtarını öğrenmek istediğini ifade eden ve zaman bilgisi ile kodlanmış (timestamped) bir mesajı genel anahtar yöneticisine gönderir.

2. Yönetici, kendi özel anahtarı olan KR_{auth} ile şifrelenmiş bir mesajı *A* kullanıcısına cevap olarak gönderir. Bu sayede, *A* kullanıcısı bu mesajı yöneticinin açık genel anahtarı ile deşifre eder ve bu mesajın yöneticiden geldiğine emin olur. Mesaj aşağıdaki bilgileri ihtiva eder:

**A* kullanıcısının *B* kullanıcısı için mesaj yollayabilmesi için *B* kullanıcısının genel anahtarı,

* Orijinal istek; *A* kullanıcısının, yöneticiye gönderdiğini düşündüğü istek ile, yöneticinin aldığını bildirdiği isteğin aynı olduğundan emin olmak için kullanılmak üzere,

* Orijinal zaman imzası; bu sayede *A* kullanıcısının gelen mesajın eski olmadığı ve dolayısıyla, gönderilmiş genel anahtarın *B* kullanıcısı yerine bir başkasının genel anahtarı olmadığından emin olması için.

3. *A* kullanıcısı *B*' nin genel anahtarını kullanarak, içerisinde *A*` nın tanımlayıcısı (ID_A) ve bu iletişimin tanımlayıcısı olan bir kelime (N_1) içeren mesajı şifreler ve *B* kullanıcısına gönderir.

4. 4. ve 5. adımlarda, *B* kullanıcısı *A* kullanıcısının genel anahtarını öğrenmek üzere yöneticiye 1. adımda *A*'nın yaptığı gibi bir mesaj gönderir ve 1. ve 2. adımlardaki iletişimin aynısı, yönetici ile *B* arasında gerçekleşir.

Bu noktada, genel anahtarlar *A* ve *B*`ye güvenli şekilde iletilmiştir ve artık bu kullanıcılar kendi aralarında güvenli iletişime başlayabilirler. Bununla birlikte fazladan iki adım eklemek mümkündür:

5. *B* kullanıcısı *A*'nın genel anahtarını kullanarak, *A*'nın tanımlayıcı kelimesini (N_1) ve kendi yarattığı yeni bir tanımlayıcı kelimeyi (N_2) şifreler ve gönderir. Çünkü 3. adımda verilen mesajını sadece *B* deşifre edebilir, ve *B*'nin N_1`i 6. adımda verilen mesaj ile döndürmesi, *A*'nın konuştuğu kişinin umduğu *B* olduğundan emin olmasını sağlayacaktır.

6. *A* kullanıcısı da *B*'nin genel anahtarı ile içerisinde N_2'nin bulunduğu mesajı şifreleyip *B*'ye gönderir ve böylece, *B* konuştuğu kişinin umduğu *A* olduğundan emin olur.

Görüldüğü gibi, toplam 7 mesaj gerekmektedir. Bununla beraber, ilk dört adım her mesajlaşma istendiğinde gerçekleştirilmeyebilir, çünkü kullanıcılar birbirlerinin genel anahtarlarını daha sonraki adımlarda kullanmak üzere saklayabilirler. Bir kullanıcı periyodik olarak, sürekli konuştuğu kişilerin genel anahtarlarının halen geçerli olduğundan emin olmak için genel anahtar isteğinde bulunmalıdır.

4.1.4. Açık anahtar sertifikası

Şekil 4.1.'de verilen yöntemin hala bazı açık noktaları vardır. Genel anahtar yöneticisi, sistemin önemli bir dar boğazını teşkil etmektedir. Bir kullanıcı temasa geçmek istediği her kullanıcı için genel anahtarlarını istemek üzere yöneticiye başvurmalıdır. Yine öncekinde olduğu gibi, isim rehberi ve anahtarları, çalınmaya yada müdahalelere karşı yönetici tarafından korunmaya çalışılmaktadır.

Alternatif bir yöntem, ilk kez L. M. Kohnfelder tarafından, 1978 yılında yayımlanan makalede sunulmuştur (17). Bu yönteme göre kullanıcılar sertifika kullanarak, bir genel anahtar yöneticisine başvurmadan güvenli bir yolla anahtarlarını değiştirebilmekte ve güvenli iletişime başlayabilmektedirler. Bir sertifika yöneticisi/yetkilisi tarafından oluşturulmuş sertifika, bir genel anahtarı ile birlikte ek bilgiyi içerir ve bu sertifika üzerindeki genel anahtara uygun özel anahtar ile beraber kullanıcıya verilir. Bir kullanıcı diğer bir kullanıcıya açık anahtarını, sertifikasını göndererek bildirir. Diğer kullanıcı da, bu sertifikanın bir otorite tarafından yaratılmış olduğunu doğrulayabilir. Bu yöntemin gereklilikleri şu şekilde sıralanabilir:

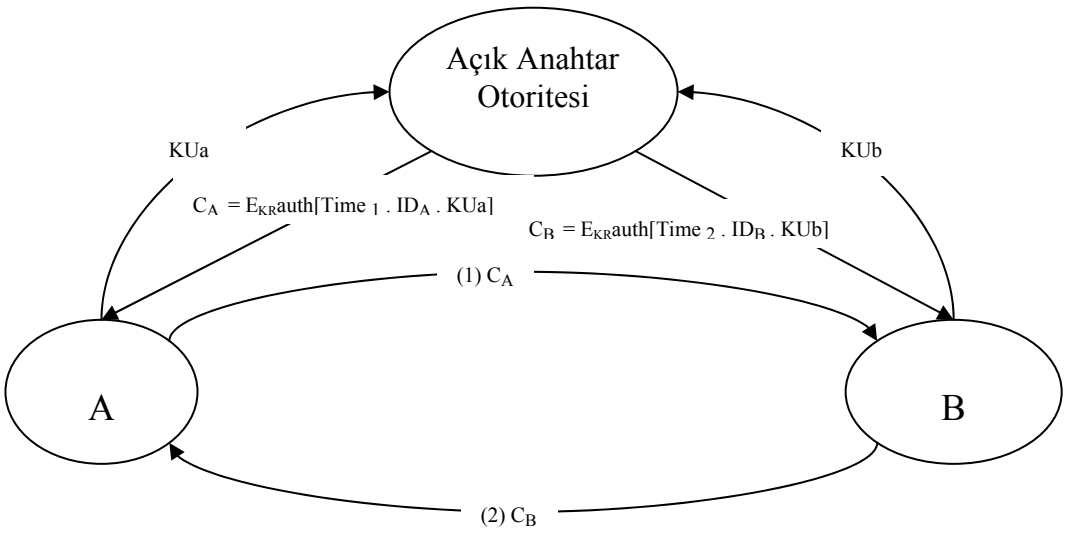
1. Herhangi bir kullanıcı bir sertifikanın kullanıcısının adını ve açık anahtarını öğrenmek üzere okuyabilmelidir.

2. Herhangi bir kullanıcı bu sertifikanın, sertifika yöneticileri tarafından oluşturuluş orijinal bir sertifika olduğunu kanıtlayabilmelidir.
3. Sertifikaları, sadece sertifika yöneticisi değiştirebilmeli ve oluşturabilmelidir.

Bu gereklilikler, Kohnfelder'in makalesinde aynen yer almaktadır (17) ve bu gerekliliklere, D. E. Denning tarafından şu gereklilik de eklenmiştir (18) :

4. Herhangi bir kullanıcı bir sertifikanın geçerliliğini kanıtlayabilmelidir.

Bir sertifika şemasının yapısı Şekil 4.2.'de gösterilmektedir. Her kullanıcı bir genel anahtara ve bir sertifikaya sahip olmak üzere sertifika yöneticisine başvurmalıdır. Başvurular yüz yüze, yada kimlik denetimi ve gizlilik sağlanması ile güvenli hale getirilmiş bir iletişim yöntemi ile gerçekleştirilmelidir.



Şekil 4.2. Sertifika otoritesi yardımıyla anahtar dağıtımı

Sertifika yöneticisi/otoritesi, bir A kullanıcısı için sertifikayı şu şekilde oluşturur:

$$C_A = E_{KR_{AUTH}}[T, ID_A, KU_A] \qquad [4.1]$$

burada KR_{auth}, sertifika otoritesi tarafından kullanılmış olan özel anahtardır. Daha sonra *A* kullanıcısının sertifikasını gönderdiği bir kullanıcı, sertifikayı şu şekilde okur ve doğrular:

$$D_{KU_{AUTH}}[C_A] = D_{KU_{AUTH}}[E_{KR_{AUTH}}[T, ID_A, KU_A]] = (T, ID_A, KU_A) \qquad [4.2]$$

kullanıcı, sertifikayı deşifre etmek için, otoritenin açık anahtarını kullanır. Çünkü sertifika sadece otoritenin açık anahtarı kullanıldığı taktirde okunabilir hale gelebilir; bu da sertifikanın geldiği yerin gerçekten sertifika otoritesi olduğunu kanıtlar. ID_A ve KU_A, elemanları, kullanıcıya, *A* kullanıcısının tanımlayıcı bilgilerini ve genel anahtarını sunar. Son olarak *T* zaman pulu da, sertifikanın son geçerli olabileceği tarihi gösterir.

Bu koşullar altında, bir özel anahtarın tehlikeye düşmesi ile bir kredi kartının kaybedilmesi birbirine benzetilebilir. Kredi kartı kaybedildiğinde, sağlayıcısından işlemlerinin durdurulması istenir, fakat bunda geç kalındığı takdirde sorunlar yaşanabilir.

4.2. Gizli Anahtarların Açık Anahtarlı Dağıtımı

Açık anahtarlar dağıtıldığında yada erişilebilir hale geldiğinde, iki kullanıcı arasındaki iletişim güvenlik altına alınmış olacaktır. Buna rağmen kimi sebeplerden ötürü, gizli veri transferi için geleneksel şifreleme yöntemleri kullanılmak isteniyor olabilir. Bu durumda, geleneksel şifrelemenin gizli anahtarlarının dağıtımı için açık anahtarlı şifreleme yöntemleri çok iyi bir araç olacaktır.

4.2.1. Basit gizli anahtar dağıtımı

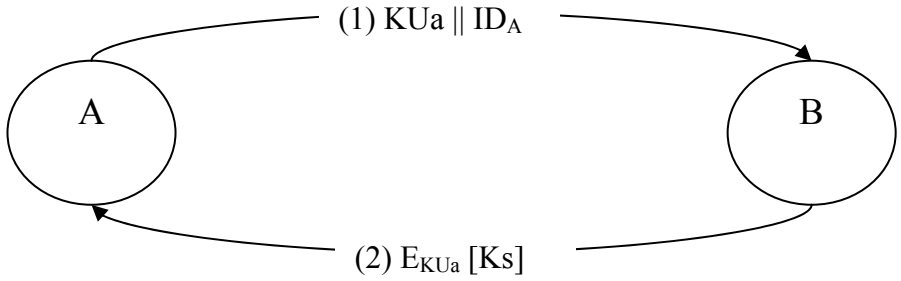
Bunun için Şekil 4.3.'de gösterilmiş son derece basit bir örnek, 1979 yılında R. C. Merkle tarafından ifade edilmiştir (19). Eğer *A*, *B* ile bir iletişim içerisine girmek istiyorsa, aşağıdaki prosedür kullanılabilir:

1. *A* bir açık/özel anahtar çifti olan $\{KU_a, KR_a\}$'yı yaratır ve *B* kullanıcısına KU_a ve *A*'nın tanımlayıcısı olan ID_A`yı içeren bir mesaj gönderir.

2. *B* bir gizli anahtar üretir (K_s), ve bunu, *A* kullanıcısına $E_{KU_a}[K_S]$ şeklinde şifreledikten sonra gönderir.

3. *A*, $D_{KR_a}[E_{KU_a}[K_S]]$'yi hesaplar ve gizli anahtarı elde eder. Yalnız *A* mesajı deşifre edebilecek ve K_S gizli anahtarı sadece *A* ve *B* tarafından bilinecektir.

4. *A* kullanıcısı; KU_a ve KR_a`yı, *B* kullanıcısı ise; $KU_a$`yı görevleri bittiğinden dolayı yok eder.



Şekil 4.3. Gizli anahtarın açık anahtarlı şifreleme ile dağıtılması

Artık *A* ve *B* geleneksel şifreleme ve oturum anahtarı K_S ile güvenli bir şekilde iletişim kurabilirler. İletişimin sona ermesi ile beraber iki kullanıcıda oturum anahtarını yok ederler. Bu yöntem, basitliğine karşın gayet etkilidir. İletişimin başlamasından önce hiçbir anahtarın olmadığı gibi, iletişimin sonunda da geriye hiçbir anahtar kalmaz. Bu sayede, anahtarların güvenliğini tehlikeye düşürecek riskler minimuma indirgenmiş olur. Aynı zamanda iletişim, pasif ataklara karşı da güvenlik altında gerçekleşmiş olur.

Fakat, bu yöntem, aktif ataklara karşı savunmasız kalmaktadır. Eğer bir rakip *(E)*, iletişim kanalının akışına müdahale etme şansına sahipse, bu kişi, tespit edilmesi için herhangi bir iz bırakmadan aşağıdaki şekilde iletişimin güvenliğine gölge düşürebilir:

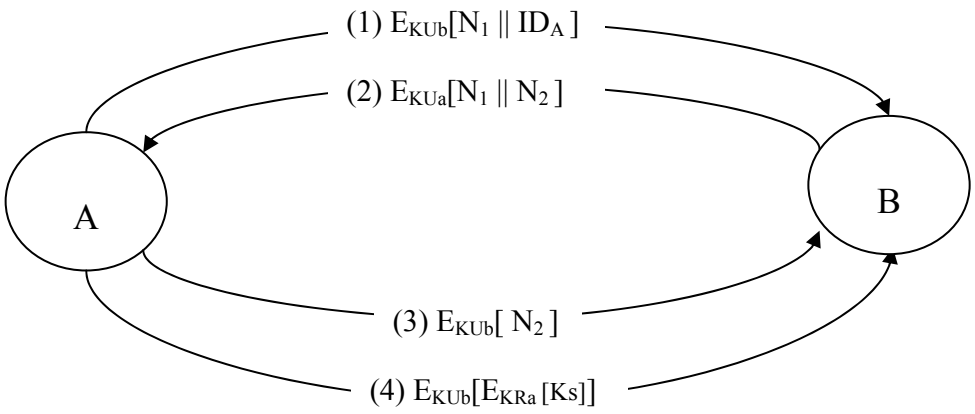
1. *A* bir açık / özel anahtar çifti olan $\{KU_a, KR_a\}$'yı yaratır ve *B* kullanıcısına KU_a ve *A*'nın tanımlayıcısı olan ID_A'yı içeren bir mesaj gönderir.

2. *E*, iletişimi durdurup, kendi özel / açık anahtar cifti olan $\{KU_e, KR_e\}$'yi oluşturur ve $KU_e \| ID_a$ ikilisini *B* kullanıcısına gönderir.

3. *B* bir gizli anahtar üretir *(K_S)*, ve bunu, *A* kullanıcısına $E_{KUe}[K_S]$ şeklinde şifreledikten sonra gönderir.

4. *E*, iletişimi durdurur ve $D_{K\,\mathrm{Re}}[E_{KU_e}[K_S]]$ hesaplaması ile gizli anahtarı öğrenir.

5. *E* kullanıcısı, $E_{KUa}[K_S]$ mesajını *A* kullanıcısına gönderir.



Şekil 4.4. Aktif ataklara karşı geliştirilmiş gizli anahtarın açık anahtarlı şifreleme kullanılarak dağıtılması

Sonuçta şu gerçekleşmiş olur, hem *A* kullanıcısı, hem de *B* kullanıcısı gizli anahtarı bilirler fakat bu anahtardan *E*`nin de haberdar olduğunu bilmezler. Dolayısıyla, *A* ve *B*, $K_S$ gizli anahtarını kullanarak mesajlaşmaya başlarlar. *E*`nin artık kanala aktif şekilde müdahale etmesine gerek yoktur; basit şekilde mesaj trafiğini dinleyip, elindeki gizli anahtar yardımıyla mesajları deşifreleyecek, *A* ve *B* kullanıcısının bu problemden haberi olmayacaktır. Böylece, bu basit iletişim sadece, tek tehlikenin ağın pasif olarak dinlenilmesi olduğu ortamlarda kullanılabilir.

4.2.2. Kimlik doğrulama ve güvenli gizli anahtar dağıtımı

Kimlik doğrulama protokolleri; iki veya daha fazla katılımcının bir açık ağ sisteminde iletişimlerini sağlamak adına bazı kriptografik amaçları gözetir. Bunlar temel olarak gizlilik, veri bütünlüğü, katılımcının kimlik doğrulaması, mesajın veya komutun kimlik doğrulaması ve anahtarın kimlik doğrulması olarak şekillenmiştir.

Bu kısımda D. R. Stinson' un araştırma ve önerileri ışığında kimlik doğrulama mekanizması ele alınmıştır (20).

Enformasyon güvenliği söz konusu olduğunda kimlik doğrulama mekanizması en önemli gerekliliklerden biridir. Diffie ve Hellman tarafından' da tartışıldığı gibi söz konusu imzanın, tüm katılımcılar tarafından kime ait olduğunun anlaşılması kolay olması gerekirken, kimliği tartışılan katılımcı dışında kimse tarafından üretilemeyecek yapıda olmalıdır.

Bununla birlikte, açık anahtarlı şifreleme sistemlerinin yapısının uygunluğu gereği bu işlemin yapılmasının kolay bir yolu vardır. *A* ve *B'nin* sistemi kullanan iki katılımcıyı, f_A fonksiyonu A ile iletişime geçmek isteyen katılımcıların mesaj şifreleme işlemlerinin tamamını simgelerken f_B fonksiyonu aynı işin B için yapılması için gerekli işlemlerin tamamını simgelesin. Basitliğin sağlanabilmesi için şifrelenmemiş düz metini oluşturan elemanlar kümesi *M* ile şifrelenmiş mesajları oluşturacak elemanların kümesi *C* birbirlerine eşit seçilmiş olsun. A'nın imzasını simgelemek üzere *Ma* (aynı zamanda bu imzaya bir kimlik numarası ve mesajın gönderildiği anı gösteren zaman bilgisi eklemek uygun olacaktır.) kullanılırsa, A katılımcısının, B katılımcısına $f_B(Ma)$ şifreli mesajının göndermesi yeterli olacaktır. Mesajın başında A katılımcısı tarafından gönderilecek $f_B f_A^{-1}(M)$ mesajı, B katılımcısı tarafından f_B^{-1} işlemi ile imzalı kısımla birlikte deşifre edildiğinde, $f_A^{-1}(M)$ kısmı B tarafından elde edilmiş olur. Mesajın A tarafından gönderildiğinin iddia edildiği bilindiğinden dolayı deşifre işleminden kalan bu kısım B tarafından A'nın açık anahtarı ile işleme tabi tutulur. Bu işlemin neticesinde A'nın dijital imzasının elde edileceği açıktır. f_A^{-1} işleminin A katılımcısı dışında kimse tarafından uygulanamayacağı bilindiğinden dolayı B katılımcısı mesajın A katılımcısından geldiğinden emin olacaktır.

Şekil 4.4., N. M. Reedham ve M. D. Shroede'in çalışmalarında işaret edilen, hem aktif hem de pasif ataklara karşı güvenlik sağlayan yaklaşımı göstermektedir (21). Burada, *A* ve *B*`nin bir önceki kısımda açıklanan herhangi bir yöntem kullanarak açık

anahtarlarını birbirlerine ulaştırdıkları kabul edilmektedir. Sonrasındaki adımlar şu şekilde gerçekleşir:

1. *A* kullanıcısı, *B*`nin açık anahtarını kullanarak, içerisinde *A*`nın bir tanımlayıcısı olan ID_A ve bu iletişimi özel olarak ifade etmek üzere bir nonce (N_1) içeren mesajı şifreler ve *B*`ye gönderir.

2. *B* kullanıcısı, *A*`nın açık anahtarını kullanarak, içerisinde *A*`nın gönderdiği (N_1) ve *B*`nin *A* gibi hazırladığı yeni bir $(N_2)$ içeren mesajı şifreler ve *A*`ya gönderir. Bu şekilde *A*, *B* kullanıcısının kimliğinden emin olur çünkü birinci mesajı deşifreleyebilecek tek kişi *B*`dir.

3. *A*, *B*`nin de emin olması için $N_2$`yi *B*`nin açık anahtarı ile şifreleyip gönderir.

4. *A* bir (K_S) gizli anahtarı belirler ve $M = E_{KUb}\left\lfloor E_{KR_a}\left[K_S\right]\right\rfloor$ şeklinde oluşturduğu mesajı *B*`ye gönderir. Mesajın, *B*`nin açık anahtarı ile şifrelenmesi, bu mesajı sadece *B*`nin okuyabileceğini, *A*`nın özel anahtarı ile şifrelenmiş olması da bu mesajın sadece *A* tarafından gönderilebileceğini garantiler.

Dikkat edilirse, bu yöntemin ilk üç adımının, Şekil 4.1.'de gösterilen yöntemin son üç adımıyla aynı olduğu görülür. Sonuç olarak, bu yöntem gizli anahtarların dağıtımı esnasında hem güvenliği hem de kimlik doğrulamayı sağlar.

4.2.3. Melez (hibrit) bir yöntem

Gizli anahtarların açık anahtarlı şifreleme yapılarından yararlanılarak dağıtılmasının bir diğer yolu , halen IBM mainframe`leri üzerinde kullanılan hibrit yapısıdır. Bu yöntem, tüm kullanıcı oturumlarında kullanılacak gizli oturum anahtarlarını bir asıl anahtar ile şifreleyerek dağıtacak olan bir anahtar dağıtım merkezi (KDC) kullanır ve bu merkez gizli anahtarları tüm kullanıcılara paylaştırır. Asıl anahtarların dağıtımı için de açık anahtarlı bir yöntem kullanılır. Aşağıda, bu üç seviyeli yöntemin kullanımını açıklanmaktadır.

• *Performans*: Sık sık anahtar değiştiren mesajlaşmadan ziyade özellikle işlem yapma amaçlı bir çok uygulama vardır. Oturum anahtarlarının açık anahtarlı şifreleme yöntemi ile dağıtılması, açık anahtarlı şifreleme ve deşifreleme esnasında hesaplamaların oldukça karmaşık olmasından dolayı sistemin büyük ölçüde performans kaybetmesine neden olmaktadır.

•*Geriye doğru uyumluluk*: Hibrit yöntem kolay bir şekilde mevcut olan KDC (anahtar dağıtım merkezi) üzerine inşa edilebilir.

Bir açık anahtar katmanının sisteme eklenmesi, güvenlik hususunda ve asıl anahtarların dağıtımı esnasında yüksek verim sağlar. Bu yöntem, bir KDC`nin bir çok kullanıcıya anahtar dağıttığı durumda büyük bir avantaj getirecektir.

4.3. Diffie-Hellman Anahtar Değişimi

İnsanlığa ilk duyurulan açık anahtar algoritması, Diffie ve Hellman`ın açık anahtarlı kriptografi olarak tanımladıkları 1976 yılında yayımladıkları "New Directions in Cryptography" isimli makalelerinde yer almıştır. Bir çok ticari uygulama bu anahtar değişimini kullanmıştır.

Algoritmanın amacı, iki kullanıcının bir anahtarı güvenli şekilde birbirlerine iletmeleri ve daha sonrasında da bu anahtar yardımı ile şifreli mesajları birbirlerine gönderebilmelerini sağlamaktır. Algoritma anahtar değişimi ile sınırlıdır.

Diffie-Hellman algoritması etkisini, ayrık logaritmik ifadelerin hesaplanmasının zorluğundan almaktadır. Kısaca, ayrık logaritmayı şu şekilde tanımlayabiliriz: Önce, bir asal sayı olan *p*'nin öyle bir temel kökü seçilir ki, bu sayının kuvvetleri 1`den *(p-1)*`e kadar olan tüm tamsayıları yaratabilir. Eğer *a*, asal sayı olan *p*`nin temel kökü ise, bu durumda sayılar:

$$a \bmod p, a^2 \bmod p, ..., a^{p-1} \bmod p$$

sayıları birbirinden farklı, ve 1`den *p-1*`e kadar olan tüm tamsayıları bir permütasyonla oluştururlar. Herhangi bir tamsayı *b* ve asal sayı olan *p*`nin bir ilkel

kökü olan *a* için, aşağıdaki eşitliği sağlayacak bir tane ve sadece bir tane olan *i* üssü bulunabilir:

$$b = a^i \bmod p \qquad 0 \le i \le (p-1)$$ koşulunu sağlayan.

i üssü, $a \bmod p$ merkezindeki *b*`nin, ayrık logaritması ya da indeksi olarak adlandırılır ve $ind_{a.b}(b)$ notasyonu ile ifade edilir. Şekil 4.5.'de Diffie – Helman anahtar değişimi yapısı verilmektedir.

Global Açık Elemanlar
* q (asal sayı)
* a (a < q ve a q' nun temel kökü)

A Kullanıcısının anahtar üretimi
* Özel X_A sayısı seçilir ($X_A < q$)
* Açık Y_A sayısı hesaplanır ($Y_A = a^{X_A} \bmod q$)

B Kullanıcısının anahtar üretimi
* Özel X_B sayısı seçilir ($X_B < q$)
* Açık Y_B sayısı hesaplanır ($Y_B = a^{X_B} \bmod q$)

A Kullanıcısının gizli anahtar üretimi
* $K = (Y_B)^{X_A} \bmod q$

B Kullanıcısının gizli anahtar üretimi
* $K = (Y_A)^{X_B} \bmod q$

Şekil 4.5. Diffie – Hellman anahtar değişimi yapısı

Bu altyapıyı da anladıktan sonra, Şekil 4.5.'de ifade edilmiş olan Diffie-Hellman anahtar değişimi açıklanabilir. Bu yöntemde, herkesçe bilinen iki sayı vardır: bir asal sayı olan *q* ve bu sayının ilkel kökü olan *a*. Anahtar değiştirmek isteyen *A* ve *B* adında iki kullanıcı olduğu düşünülürse *A* kullanıcısı $X_A<q$ olacak şekilde rasgele bir X_A tamsayısı seçer ve, $Y_A = a^{X_A} \bmod q$ değerini hesaplar. Benzer şekilde *B* kullanıcısı diğer kullanıcıdan bağımsız şekilde $X_B<q$ olacak bir X_B tamsayısı seçer ve o da, $Y_B = a^{X_B} \bmod q$ değerini hesaplar. Her iki kullanıcıda *X* değerini özel olarak saklar ve

Y değerini diğer taraf ile paylaşır. A kullanıcısı anahtarı $K = (Y_B)^{XA} \bmod q$, B kullanıcısı da, $K = (Y_A)^{X_S} \bmod q$ şeklinde hesaplarlar. Bu iki hesaplamanın sonucunun birbiri ile ilişkisi aşağıdaki modüler aritmetik kuralları ile ispatlanabilir:

$$\begin{aligned} K &= (Y_B)^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\ &= (\alpha^{X_B})^{X_A} \bmod q \\ &= (a^{X_A})^{X_B} \bmod q \\ &= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\ &= (Y_A)^{X_B} \bmod q \end{aligned} \qquad [4.3]$$

Bu sayede, iki tarafta bir gizli anahtarı değişmiş olur. Ayrıca, X_A ve X_B'nin gizli olmasından dolayı, herhangi bir saldırgan çalışmak için sadece şu değerleri bilir: q,a,Y_A,Y_B. Örneğin bir saldırganın B kullanıcısına ait olan gizli anahtarı bulmak için aşağıdaki hesaplamayı yapması gerekir:

$$X_B = ind_{a,q}(Y_B) \qquad [4.4]$$

Bu hesaplamanın başarısı sonrasında saldırgan gizli anahtarı B kullanıcısının hesapladığı gibi hesaplayabilecektir.

Aşağıdaki örnek, anahtar değişimi için, asal sayı olan $q=97$ ve ilkel kökü olarak da $a=5$ kullanarak algoritmayı açıklamaktadır. Burada A ve B kullanıcıları gizli anahtarlarını $X_A=36$ ve $X_B=58$ olarak seçmişlerdir. Sonuç olarak her ikisi de açık anahtarlarını şu şekilde hesaplarlar:

$$Y_A=5^{36}=50 \bmod 97 \qquad [4.5]$$

$$Y_B=5^{58}=44 \bmod 97 \qquad [4.6]$$

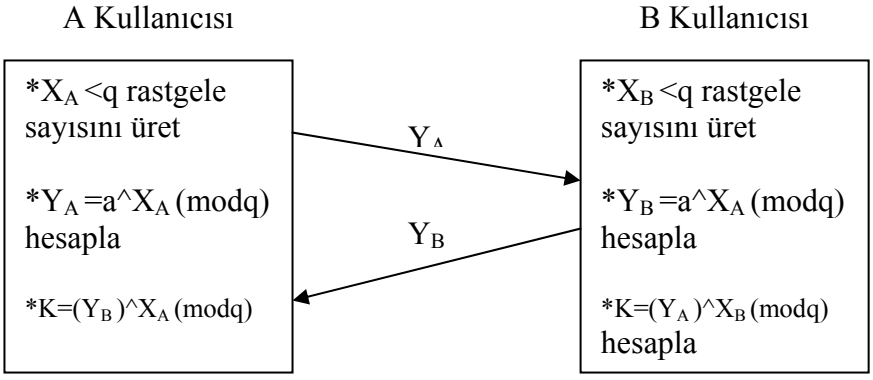
Açık anahtarlarını değiştikten sonra, gizli anahtarlarını da şu şekilde hesaplarlar:

$$K = (Y_B)^{X_A} \bmod 97 = 44^{36} = 75 \bmod 97 \qquad [4.7]$$

$$K = (Y_A)^{X_B} \bmod 97 = 44^{36} = 75 \bmod 97 \qquad [4.8]$$

{50,44} bilgisine sahip olan bir saldırgan, 75 değerini kolayca hesaplayamayacaktır ve bu hesapsal zorluk seçilen sayıların büyüklüğü ile orantılı olarak artacaktır.

Şekil 4.6., Diffie-Hellman hesaplamasını basit şekilde ifade etmektedir. Diffie-Hellman algoritmasının LAN (yerel network) kullanıcıları arasında kullanımı için bir diğer örnekte şu şekilde verilebilir: Söz konusu ağ altında çalışan her kullanıcı dayanıklı ve uzun birer X_A ve buna bağlı genel bir Y_A hesaplamış olsunlar. Kişilerin açık anahtarları ve herkesçe bilinen *q* ve *a* değerleri herkesin erişebileceği merkezi bir rehberde tutulduğu takdirde, herhangi bir anda bir *B* kullanıcısı mesajlaşmak istediği bir *A* kullanıcısının açık değerine ulaşabilecek ve onun için şifrelediği mesajı kendisine gönderebilecektir. Eğer merkezi rehber güvenilir ise, bu iletişim gizliliği ve kimlik denetimini sağlamış olacaktır. Tüm bunlara rağmen bu teknik, aktif tekrarlama gibi ataklara karşı korumasız kalır.



Şekil 4.6. Diffie – Helman hesaplamasının örnek gösterimi

5. HATA DENETİMİ

5.1. Hata Denetimi Yapısı ve Yöntemleri

Hata Denetimi tekniklerinin amacı, iletişim hattından veya göndericiden kaynaklanabilecek sorunlarla, mesajın bütünlüğünde meydana gelebilecek bozulmaların tespit edilmesidir. Bunu yapabilmek için gönderici, mesajın bir fonksiyonu olan ve kontrol-toplamı (checksum) olarak adlandırılan bir değeri hesaplayarak, mesajla birlikte alıcıya gönderir. Alıcı gelen mesajı aynı fonksiyon ile işleme sokarak kendi kontrol-toplamını oluşturur ve her iki toplamı karşılaştırır. Mesajın bütünlüğünde bir hata oluşmamışsa bu iki değer matematiksel olarak birbirine eşit olmalıdır. Aksi takdirde alıcı mesajın bozuk olduğuna hükmedecektir.

Örneğin, kontrol-toplamı fonksiyonunu; mesajı oluşturan tüm elemanların matematiksel olarak toplanarak, sonucun "*mod 256" (8 bitlik bir sayının işaretsiz olarak alabileceği maksimum değer)* işlemine tabii tutulması olarak seçilirse:

Orijinal Mesaj : 6,23,4
Mesaj ve kontrol-toplamı : 6,23,4;33
İletilen Mesaj : 6,27,4;33

Yukarıdaki örnekte iletilen mesajın ikinci byte değerinin bozularak "27" olarak algılandığı görülmektedir. Bununla birlikte alıcı, mesajla birlikte gelen kontrol-toplamı değerinin (33), aldığı mesajın kontrol-toplamından (37) farklı olduğunu anlayacak ve bozulmayı tespit edecektir. Kontrol-toplamının da iletişim kanalında bozulma ihtimali vardır. Bu durumda alıcı tarafından tüm mesajın hata içerdiği öngörülecektir. Yine de bu durum tehlikesiz-hata olarak adlandırılacaktır ve veri bütünlüğünden emin olmak amacı ile verilen bir ödün olarak düşünülebilir.

Bununla birlikte tehlikeli-hata olarak adlandırılan bir durum daha söz konusudur ve bu durumda mesajda ve/veya kontrol-toplamı değerinde meydana gelecek bozulmaların kontrol-toplamı değerini doğrulayacak şekilde olmasıdır. Böyle bir

olasılık oldukça düşük olmasına rağmen her zaman vardır ve bu hata olasılığını azaltmanın tek yolu kontrol-toplamında taşınan bilginin miktarını arttırmaktır. (Bir byte yerine daha büyük bir kontrol-toplamı.)

5.1.1. Daha karmaşık bir yapı gerekliliği

Önceki bölümde verilen örnekte mesajdaki bozulmanın, hata denetiminin kontrol-toplamı yardımıyla nasıl yapıldığı gösterilmiştir. Bu örnekte kullanılan kontrol-toplamı *mod256*'ya göre hesaplanmıştı:

Orijinal Mesaj : 6, 23, 4
Mesaj ve kontrol-toplamı : 6, 23, 4; 33
İletilen Mesaj : 6, 27, 4; 33

Bu mesajda meydana gelen bozulma oldukça basittir. Mesajın bütününde belirli sayıda ve rasgele oluşabilecek bazı hatalar, aşağıda gösterildiği gibi algılanamayacak nitelikte olabilir:

Orijinal Mesaj : 6, 23, 4
Mesaj ve kontrol-toplamı : 6, 23, 4; 33
İletilen Mesaj : 8, 20, 5; 33

Kontrol-toplamının etkisinin arttırılabilmesi, 8 bitlik bir değerde tutulmasının yerine 16 bitlik bir değerde tutulması ile mümkün olabilir. Böylece toplamın mod'u 256 dan 65536'ya çıkarılmış olacaktır. Dolayısı ile yukarıda sunulan durumun oluşma ihtimali 1/256 dan 1/65536'ya düşecektir. Bu temel olarak iyi bir fikirdir fakat söz konusu örnekte kullanılan formül basit bir toplama işleminden ibarettir. Bundan dolayı mesajda bulunan her bir byte, kontrol-toplamına sadece bir byte etki edecektir. Kontrol-toplamı değerinin megabyte'lar mertebesinde olduğu durum düşünüldüğünde yeterince uzun olmayan bir mesaj için kontrol-toplamının yüksek anlamlı byte'larının toplama işleminden etkilenmeyeceği açıktır. Bu problem

üstesinden gelmek, basit bir toplama işlemi yerine daha kompleks bir formül kullanılması ile mümkündür.

Bu noktada, daha güçlü bir kontrol-toplamı formülü geliştirebilmek için iki farklı iyileştirmeye ihtiyaç olduğu söylenebilir:

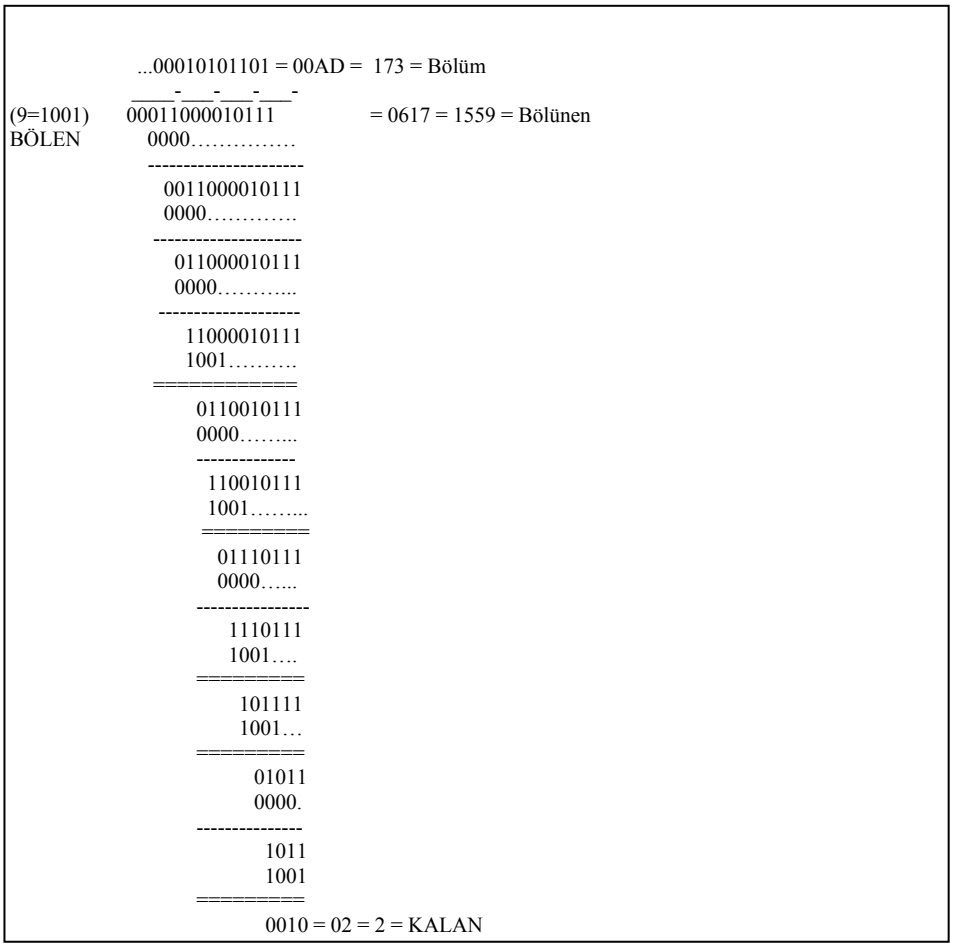
1) Kontrol-toplamının kapasitesi daha yüksek tutularak hata olasılığı düşürülebilir.
2) Daha karmaşık bir formül kullanarak mesajdaki her bir byte'ın kontrol toplamının bitlerini daha etkili bir biçimde değiştirmesi sağlanabilir.

5.2. Döngüsel Artıklık Denetimi (Cyclic Redundancy Check -CRC)

Daha karmaşık bir formülün bulunabilmesi için birçok fikrin geliştirilmesi mümkündür. Pi sayısı gibi bir sabitin basamakları çizelge olarak kullanılabileceği gibi mesajı oluşturan her bir byte kontrol-toplamını oluşturan her bir byte ile işleme sokulabilir veya bunlar gibi sayısız öneri sunulabilir.

CRC algoritmasının arkasında yatan fikir ise, tüm mesaja çok büyük ve ikili düzende yazılmış bir sayı, gibi davranmaktır. Bu büyük sayı daha önceden belirlenmiş ve yine ikili düzende yazılmış bir sayıya bölünerek, bu bölme işleminden kalan, kontrol-toplamının oluşturmasını sağlar. Alıcı da aynı işlemi gerçekleştirerek elde ettiği kalanı kontrol-toplamı ile karşılaştırır.

Mesajın bir önceki örnekteki gibi (6,23) olduğu farz edilsin. Tüm mesajın büyük bir sayı olarak düşünülmesi temel olduğu için CRC algoritması, bu mesajı, onaltılık sayı düzeninde (hexadecimal) 0617 olarak algılayacaktır. Bu sayının ikilik düzendeki karşılığı; 0000 - 0110 – 0001 – 0111 'dır. Kontrol-toplamının, bir byte genişlikte olduğu ve sabit bölenin 1001 ikili sayısı olduğu varsayılırsa bölme işlemi aşağıdaki gibi olacaktır:



Şekil 5.1. İkili Sistemde Bölme İşlemi Örneği

Ondalık olarak bu işlem; "1559 sayısının 9'a bölünerek bölüm, 173 ve kalan, 2" bulunması olarak açıklanabilir.

Mesajın bitlerindeki değişim bölüm değerini fazla değiştirmeyecektir fakat kalan değeri mesajdaki değişimden ciddi oranda etkilenerek değeri değişecektir. Hata denetlemede, bölme işleminin toplamaya göre avantajı işte bu değişimdir.

Sonuç olarak hesaplanan 4 bitlik kontrol- toplamı ile birlikte gönderilen mesaj hexadecimal olarak 06172 olacaktır.

5.2.1. Polinom aritmetiği

Bir önceki bölümde anlatılan, bölme işlemine dayanan düzen, CRC düzenine oldukça benzemektedir. CRC düzeninde; bölünen, bölen ve kalan değerleri pozitif

tamsayılar olarak görülmek yerine ikilik düzende temsil edilen bir polinomun katsayıları olarak düşünülmüştür.

CRC düzeninde de önceki bölümde anlatılan düzen gibi, tüm mesaj ikilik düzende yazılmış büyük bir sayı gibi düşünülür. Aynı şekilde bölünen, bölen ve kalan değerleri de ikilik düzende yazılır ve bir polinomun katsayıları olarak ifade edilir. Örneğin: ondalık olarak yazılan; 23 sayısı onaltılık düzende; 17 ve ikilik düzende; 10111 olarak ifade edilir ve bu değer katsayı katarı olarak düşünüldüğünde aşağıdaki polinoma karşılık gelir:

$$1(x^4) + 0(x^3) + 1(x^2) + 1(x^1) + 1(x^0) \qquad [5.1]$$

İfade daha basitleştirilirse:

$$x^4 + x^2 + x^1 + 1 \qquad [5.2]$$

elde edilir.

Bu tekniği kullanarak mesajın kendisi ve bölen, değeri polinom olarak ifade edilir ve bir önceki bölümde verilen bölme işlemi uygulanır. Örnek olarak ikili düzende yazılmış 1101 sayısı ile yine ikili düzende yazılmış 1011 sayısı ile çarpmak istediğimizi düşünelim. Bu işlem polinom aritmetiği ile aşağıdaki gibi yapılabilir:

$$\begin{aligned}&(x3 + x2 + x0)(x3 + x1 + x0)\\&= (x6 + x4 + x3 + x^5 + x^3 + x^2 + x^3 + x^1 + x^0)\\&= x^6 + x^5 + x^4 + 3x^3 + x^2 + x^1 + x^0\end{aligned} \qquad [5.3]$$

Bu noktada, x^3 ifadesinin katsayısının dağıtılarak sonucun ikili sistemde ifade edilebilecek bir hale getirilmesi gerekir. Bunun için x değeri 2'ye eşitmiş gibi düşünülerek:

$$3x^3 = 24 = (11000)_2 \equiv (x^4 + x^3) \qquad [5.4]$$

asıl denklemde yerine yazıldığında ve bu işlem tüm katsayılı ifadeler dağıtılana kadar yinelendiğinde aşağıdaki polinom elde edilecektir:

$$=x^7+x^3+x^2+x^1+x^0 \qquad [5.5]$$

Bu polinomun ikili sistemde ifadesi olan (10001111) (ondalık, 143) sayısının çarpımın doğru sonucu olduğunu görebiliriz.Yani:

(1101)*(1011)=(10001111)
13 * 11 = 143

bulunmuştur. Fakat CRC algoritmasında önemli olan sonucun matematiksel olarak doğruluğundan çok algoritmaya uygun olmasıdır.

Bu işlemde x'in üçüncü kuvvetinde karşılaşılan taşma durumu x'in 2 olduğu bilinerek giderilebilmiştir. Bununla birlikte x'in üçüncü derece kuvvetinde meydana gelen taşma işleminin giderilmesi sırasında bu kuvvetin katsayısının diğer kuvvetlerin katsayılarını değiştirdiği gözlenmiştir.

Yukarıda sunulan gerçek polinom aritmetiğinin bir başka versiyonu, polinomda taşmaya izin verilmeyecek şekilde geliştirilmiştir. Dolayısı ile bu yöntemde polinomdaki kuvvetlerin katsayıları birbirinden bağımsızdır yani birbirlerini etkilemezler. Bu yöntem "2 moduna göre polinom aritmetiği" (polynomial arithmetic mod 2) olarak bilinir. Taşmaya izin verilmeyeceğine göre her bir katsayı *mod 2* ye uyacak şekilde yeniden hesaplanır. "0" ve "1" lerden oluşacak hale getirilir. Dolayısı ile bir önceki örnekte $3x^3$ 'ün dağıtılmadan önceki hali bu yönteme göre hesaplanarak;

$$3 \equiv 1 \pmod 2$$

katsayısı da yerine konulursa:

$$(x^6 + x^5 + x^4 + 3x^3 + x^2 + x^1 + x^0) \equiv (x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + x^0)$$

denkliği yazılabilir. Görüldüğü gibi bu yöntemde taşmanın engellenmesi ile sonuç için bir sınır belirlenmesi mümkün olmuştur ve örnekte görülen 8 bitlik gerçek sonuç 7 bitlik bir ifade halinde kalmıştır.

2 moduna göre polinom aritmetiği, taşma durumu hesaplanmamış ikili sistem aritmetiğinin tam olarak aynısıdır. Şimdiye dek ikili sistem aritmetiği işlemlerinin polinom aritmetiği işlemleri ile ilişkisi sunularak anlatımın daha basit olması ve analitik olarak anlaşılabilirliğinin arttırılması amaçlanmıştır. Bundan sonraki bölümlerde polinom aritmetiği gösterim olarak kullanılmayacaktır. Bununla birlikte hesaplamalar, şimdiye kadar sunulan benzetmeye uygun düşünüldüğünde, anlaşılması daha kolay olacaktır.

5.2.2. Taşmasız ikili sayı sistemi aritmetiği (CRC aritmetiği)

Polinom aritmetiğinden vazgeçerek CRC hesaplamaları sırasında kullanılan taşmasız ikili sistem aritmetiğinin kullanılması ifadelerin sadeliği açısından ve dijital sistemlere uygulanabilirliği açısından daha uygun olacaktır. Bu aritmetik birçok kaynakta, direkt polinom aritmetiği olarak adlandırılsa da bu ve bundan sonraki bölümlerde CRC aritmetiği olarak kullanılmıştır.

CRC Aritmetiğinde Toplama: Toplam işlemi ikili sistemdekinin taşmaya izin verilmeden uygulanmasından ibarettir.
Örneğin:

```
  10011011
+ 11001010
----------
  01010001
```

Bit pozisyonlarına göre olası dört durum:

0+0=0
0+1=1
1+0=1
1+1=0 (taşma yok!)

olarak sıralanabilir.

CRC Aritmetiğinde Çıkarma: Toplama işleminde olduğu gibi ikilik sistemde taşma hesaplanmadan çıkarma işlemi yapmaktan ibarettir.

```
  10011011
-11001010
----------
  01010001
```

Bit pozisyonlarına göre olası dört durum:

0-0=0
0-1=1 (bir üst basamak etkilenmez)
1-0=1
1-1=0

Hem toplama hem de çıkarma işlemlerinin doğruluk çizelgeleri incelendiğinde bu işlemler için "0" ın etkisiz eleman olduğu görülmektedir.

Anlaşılabileceği gibi CRC aritmetiğinde, hem toplama hem de çıkarma işlemi mantıksal "harici veya" (exclusive or) işlemidir ki bu işlem sayının bit bazında, tersini almaktan ibarettir. Dolayısıyla CRC aritmetiğindeki toplama ve çıkarma işlemleri, işlem yükü bakımından oldukça efektiftir.

CRC aritmetiğinde, taşmaların iptal edilmesinden doğan bazı durumlar yadsınılmamalıdır. İkilik düzende 1010 sayısının 1001 sayısından daha büyük olduğu açık olmasına rağmen 1010 sayısından 1001 sayısını elde etmek çıkarma işlemi ile olduğu kadar toplama işlemi ile de mümkündür:

1001 = 1010 + 0011
1010 = 1010 + 0011

Buradan anlaşılabileceği gibi CRC aritmetiğinde derecelendirme yapmak mantıklı değildir.

CRC Aritmetiğinde Çarpma: Çarpma işlemi de gerçek aritmetikte olduğundan pek de farklı değildir. Tek fark çarpma işlemi sırasında uygulanan toplama işlemi taşma hesabı yapılmayan CRC aritmetik toplam işlemidir.

```
   1101
 x 1011
 -------
   1101
  1101.
 0000...
1101...
------------
1111111
```

CRC Aritmetiğinde Bölme: Bölme işlemi bilinen tanımından biraz farklıdır. Bu noktada CRC aritmetiğindeki sayıların büyüklüğü için şöyle bir tanımın verilmesi uygun olacaktır: X sayısının Y sayısına eşit veya daha büyük sayılabilmesi için en yüksek dereceli bitinin değerinin Y'nin en yüksek dereceli bitinin değerine eşit veya daha yüksek olması gerekir. Aşağıdaki örnekte bölme işleminin nasıl yapılacağı gösterilmiştir.

```
          1100001010
          10011
÷ _________________
          1001110110000
          10011……
          ---------------
           000010110000
           00000……..
           --------------
             00010110000
             00000……...
            -------------
               0010110000
               00000…….
              -------------
                 010110000
                 00000....
                ------------
                    10110000
                    10011.....
                   ------------
                     0101000
                     00000....
                    -----------
                       101000
                       10011.
                    -----------
                        01110
                        00000
                    -----------
                          1110 = Kalan.
```

Şekil 5.2. CRC Aritmetiğinde bölme örneği

Bölme ve çarpma işlemlerinden bahsettikten sonra CRC aritmetiğinde bölünebilirlik ve çarpılabilirlik kavramlarını da açıklamak gerekir. CRC aritmetiğinde, B sayısının bir A sayısının çarpanı olarak sayılabilmesi için; B sayısının bitlerinin çeşitli kaydırma operasyonları ile XOR işlemine tabii tutularak A sayısını üretebilmesi mümkün olmalıdır. Örnek olarak, A sayısının 0111010110 ve B sayısının 11 ikili değerlerine sahip olduğu varsayıldığında, A sayısı B sayısından aşağıdaki şekilde elde edilir:

```
  0111010110
= .......11.
+ ....11....
+ ...11.....
  .11.......
```

Bununla birlikte eğer A sayısı 0111010111 olarak alınsaydı B sayısı kullanılarak elde edilmesi mümkün olmazdı ve B sayısı A sayısına bölünebilir denilemezdi. Yukarıdaki işlem dikkatle incelendiğinde bunun nedeni daha net anlaşılacaktır.

5.2.3. Tam fonksiyonel CRC hesaplama örneği

CRC aritmetiğinin sunulmasının ardından CRC hesaplamalarının açıklanması yapılabilir. Bu hesaplamaların basitçe CRC aritmetiğinde sunulan bölme işlemine dayandığı söylenebilir.

CRC hesaplamalarının yapılabilmesi için öncelikle yapılması gereken şey bir bölen değerinin belirlenmesidir. Bu bölen değeri "üretici polinom" veya "polinomal" olarak adlandırılır. Her CRC algoritmasının kendine özgü belirlenmiş bir polinomal değeri mevcuttur.

Polinomal olarak herhangi bir değerin seçilmesi mümkün olmasına rağmen bazı polinomal değerlerin diğerlerine oranla daha efektif sonuçlar verdiği bilinmektedir. Bir sonraki bölümde bu husus incelenerek daha uygun polinomal değerin seçilmesi ayrıntılarıyla incelenecektir. Seçilen polinomal değerinin genişliği oldukça önemlidir ve tüm hesaplamaları etkileyecek niteliktedir. Genel olarak 16 veya 32 bitlik

değerlerin seçilerek bilgisayar işlemcilerinin hesaplama yapmalarının daha da kolaylaştırılması düşünülür.

CRC hesaplaması, ikili düzende büyük bir sayı olarak düşünülen mesajın polinomal değere bölünmesi ile CRC değerinin bulunmasından ibarettir. Bu adımların örneklenerek açıklanabilmesi için aşağıdaki değerler seçilmiştir:

Orijinal mesaj	:1101011011
Polinomal	:10011
Sıfır ilave edilen Mesaj	:11010110110000

Bölme işlemi CRC aritmetiğinde anlatıldığı gibi yapılır:

```
          11010110110000
          10011
÷_________________
          11010110110000 = Genişletilmiş mesaj (1101011011 + 0000)
          10011……..
          --------------------
            1001110110000
            10011……...
          --------------------
              00010110000
              00000……
            ------------------
               0010110000
               00000…….
             -----------------
                  010110000
                  00000……
               ----------------
                       10110000
                       10011….
              ----------------
                       0101000
                       00000…
             ----------------
                         101000
                         10011.
              ----------------
                         01110
                         00000
                         -------
                          1110        = Kalan = Kontrol-Toplam değeri!
                          1100001010 =Bölüm (Bölüm değeri hesaplamalar için gereksizdir.)
```

Şekil 5.3. Mesajın, polinomale CRC aritmetiği kullanılarak bölünmesi

Bölme işleminden elde edilen kalanın hesaplanması ile kontrol-toplamı değeri bulunmuştur. Bu değerin mesajın sonuna eklenmesi ile göndericinin hazırladığı mesaj 1101011011*1110* olacaktır.

Alıcı tarafında ise aşağıdaki iki işlemden biri gerçekleştirilebilir:

a) Mesaj ve kontrol-toplamı değeri birbirlerinden ayrılır. Mesajın sonuna kontrol-toplamı genişliğinde sıfır ilave edilerek göndericinin yaptığı işlem yenilenerek kontrol toplamı değeri hesaplanır.
b) Mesaj, kontrol-toplamı değerinden ayrılmadan bölme işlemine tabii tutularak, sonucun (kalan) sıfıra eşit olup olmadığı yani mesajın polinomala kalansız bölünüp bölünemediği kontrol edilir.

Her iki işlemde sonuç olarak istenileni sağlayarak mesajın bütünlüğünün bozulup bozulmadığını kontrol edebilecektir. Genellikle tercih edilen, ikinci yöntem olduğundan dolayı bundan sonraki anlatımlarda "b" ile sunulan teknik baz alınmıştır.

CRC algoritmasında yapılan işlemler aşağıdaki adımlarla özetlenebilir:

1. Genişliği belirlenen bir polinomal değerinin seçilmesi
2. Seçilen genişlik miktarındaki sıfırın mesaja ilave edilmesi.
3. CRC aritmetiği kullanarak oluşturulan mesajın polinomale bölünerek kalan değerinin kontrol-toplamı olarak hesaplanması

5.2.4. Polinomalin seçimi

Polinomalin seçimine geçilmezden önce, gönderilen mesajın, polinomalin tam katı olduğu düşünülmelidir. Çünkü bir önceki bölümde de anlatıldığı gibi mesaj gönderilmeden önce polinomal genişliği kadar "0" ilave edilmiş ve mesajın polinomale bölümünden kalan değer, mesaja ilave edilmiştir. İletilen mesaj alıcı kısmında M+E olarak algılanacaktır ki burada M:Alıcının gönderdiği mesajı E: ise

iletim hattında mesaja eklenen hata vektörünü göstermektedir. P polinomali temsil etmek üzere *M mod P = 0* olduğu bilinmektedir. Dolayısı ile hatanın algılanması *E mod P* işleminin sıfıra eşit olmadığı durumlar için söz konusudur. Hata denetlemek için en uygun polinomal, hatalı mesaj için gerçekleştirilen *E mod P* işleminin, sıfıra eşit olma ihtimalini minimum tutacak polinomaldir. Bu bakımdan seçimi etkileyecek, hata oluşumu çeşitleri aşağıda incelenmiştir:

Tek Bit Hatalar: Matematiksel olarak E vektörünün E =100..000 yapısında yazılması ile ifade edilir. Tek bitlik bir hatanın oluştuğunu anlayabilmek polinomalin en az iki bitlik uzunlukta seçilmesi ile mümkün olabilir.

İki Bit Hatalar: E, hata vektörünün iki adet 1 değeri içerdiği durumlarda E= 100.0001..00 formunda yazıldığı durumlardır. Böyle durumlar için seçilecek polinomalin çarpanlarının 11,101,1001,10001,100001,...v.b. yapısında olmaması daha uygun olacaktır.

Tek Sayıda Bit ile İfade Edilen Hatalar: Bu yapıdaki hataların tespiti için polinomalin çift sayıda bit (1) ile ifade edilmesi uygun olacaktır. Bunun nedeninin anlaşılması için aşağıdaki önermeler incelenirse:

1) CRC çarpımı basitçe, sabit bir değerin çeşitli offset değerleri verilerek yazmaçta bulunan değer ile XOR (exclusive or) işlemine tabii tutulmasıdır.
2) Yazmaçta bulunan değerin çift sayıda "1" bitinden oluştuğu düşünüldüğünde, XOR işlemi sonrası bu sayının içerdiği 1'lerin sayısı işlemden önce tek ise işlemden sonra da tek olacaktır.

Örnek olarak E vektörünün 111'e eşit olduğu düşünülsün. Polinomalin 11 olarak seçilmesi durumunda offset değerlerinin değiştirilerek XOR işleminin uygulanması ile E vektörünün, *E = E xor 011* ve *E = E xor 110* işleminin ardından yine tek sayıda 1 içermesi sağlanmış olacak, sıfırlanmayacaktır. Birçok popüler CRC polinomali çift sayıda "1" biti içerecek şekilde seçilmiştir. Ayrıca yukarıdakine benzer bir açıklama kullanan bir yöntem ile polinomalin, 11 sayısının çarpanı

olmasının tek sayıda bit ile ifade edilen hataları yakalamada daha başarılı olduğu Tanenbaum tarafından ortaya konulmuştur (22).

Burst Hataları: Hata vektörünün; E=00...01111...11100..0000 şeklinde verildiği hatalardır. Anlaşılacağı gibi hatanın belli bir süre boyunca mesajın tüm bitleri üzerinde etkili olduğu durumlarda ortaya çıkar. E vektörü; E=(1000...00)(1111...111) çarpımı şeklinde yazılabilir. Bu hatanın yakalanabilmesi için polinomalin en düşük anlamlı bitinin "1" olarak seçilmesi uygun olacaktır. Böylelikle ilk terim olan (100..000)'ın polinomal değeri ile sıfırlanması mümkün olmaz. Aynı zamanda polinomalin genişliğinin en az çarpanı oluşturan ikinci terimdeki "1" lerin sayısı kadar olması Tanenbaum tarafından önerilmiştir (22).

Son olarak, aşağıdaki çizelgede, bazı popüler CRC algoritmalarında kullanılan polinomal değerleri sunulmuştur:

Çizelge 5.1. Bilinen bazı CRC algoritmalarının polinomal değerleri

Polinomal Genişliği	***Polinomal Değeri***	***Kullanımı***
16 Bit	(16,12,5,0)	X-25 Standart
16 Bit	(16,15,2,0)	[CRC-16]
32 Bit	(32,26,23,22,16,12,11,10,8,7,5,4,2,1,0)	Ethernet

5.2.5. Doğrudan CRC uygulaması

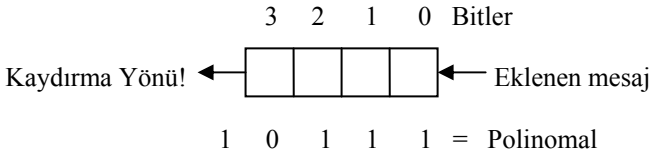
Bu bölüme kadar CRC algoritmasının teorisi anlatılmıştır. Bundan sonraki bölümlerde CRC algoritmasının bilgisayar sistemlerinde kullanılabilecek bir modeli çıkarılarak, ileriki bölümlerde ise protokole entegrasyonunu sağlamak amacıyla C++ dili ile ifade edilecektir.

CRC algoritmasının temeli, "CRC aritmetiğinde bölme" işlemine dayandığından dolayı uygulamayı geliştirmeye bu işlemden başlanılmalıdır. Normal bölme işleminin kullanılamamasının iki temel sebebi vardır. İlki CRC aritmetiğindeki

bölme işleminin işlemcinin tanıdığı aritmetik bölme işleminden yapısal farkı. İkincisi ise tüm mesajın büyük bir sayı olarak düşünülmesinden dolayı bölünen değerinin çok büyük olmasıdır ki günümüz bilgisayarlarının yazmaçları bu derece geniş bir sayıyı tutabilecek kapasitede de değildir.

Dolayısı ile bölme işlemini gerçekleştirirken, öncelikle bölünen değerini tutacak yazmacın, mesajın yüksek öncelikle bitinden başlamak üzere beslenmesi gerekir. Herhangi bir karışıklığın olmaması açısından bundan sonra verilecek örneklerde, mesajın formatının, byte' lar dan oluştuğu ve bu byte'ların en anlamlı bitlerinin 7. bit olduğu düşünülmelidir.

Örneği sunmadan önce polinomal değerinin 10111 olduğu ve dolayısı ile ihtiyaç duyulan yazmaç genişliğinin 4 bit olduğu varsayılsın. Bu şartlar altında



Yukarıda gösterilen yapıda dikkat edilmesi gereken bir başka nokta önceki bölümlerde de hatırlatıldığı gibi, eklenen mesajın polinomal uzunluğu kadar “0” ile genişletilmiş olmasıdır.

Çizelge 5.2. Bölme işleminin adımları

Bölme işlemini gerçekleştirmek için:

*Yazmaç değerini sıfırla

*Mesajın sonuna polinomal genişliği kadar "0" yazıp genişleterek işleme hazırla.

*Döngüye Başla

- Yazmaçtaki bitleri sola doğru 1 bit kaydırırken bit0 pozisyonunu mesaj katarından alınan (en anlamlı bit) ile besle.
- Eğer bir önceki adımda yazmaçtan taşma olmuşsa (yani sola kayarak yazmaçtan çıkan bit "1" değerinde ise) :

  Yazmaç=Yazmaç xor Polinomal

  işlemini yap.
- Mesaj Sonu ise Döngüden Çık.

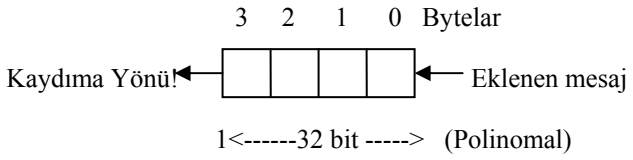
Çizelge 5.2.'de gösterilen algoritma, bölme işleminin en basit şekilde uygulamasını göstermektedir. Algoritma sonunda yazmaçta bulunan değer, kalan değeri olacak ve kontrol-toplamı olarak mesaja eklenecektir. (İkinci adımda mesaja eklenen "0" ların yerine).

5.2.6. Tablo ile sürülen CRC uygulaması

Yukarıda çok basitçe gösterilen uygulama CRC sisteminin tüm sistematiğini içinde barındırmasından dolayı oldukça önemlidir. Fakat anlaşılabilirliğinin arttırılması için basitleştirildiğinden dolayı, bilgisayarda direkt uygulamak için pek te uygun değildir. Bunun en önemli nedeni işlemlerin bit bazında gerçekleştirilmesidir. Dolayısı ile algoritmanın bu şekilde C programlama diline aktarılması çok hantal bir rutin doğuracaktır. Operasyonun hızlandırılması için mesajın, bir bitten daha büyük parçalardan oluşacak şekilde işlenmesi uygun olacaktır. Bilgisayarlardaki adresleme mantığı düşünüldüğünde en uygun genişlikler; 4bit (nibble), 8bit (byte), 16bit (word)

ve 32 bit (double word) olmaktadır.

Bu amaçla daha önce 4 bit genişliğinde seçilen polinomal değerinin 32 bit seçildiği düşünülürse:



Not:Polinomalin genişliği en baştaki kılavuz biti ile birlikte 33 bittir.

Buradaki gösterim hemen hemen bir önceki gösterimin aynısıdır. Yalnızca daha önce bit numarasını gösteren kutular artık byte numarasını göstermektedir.

Daha önce sunulan basit algoritmadaki mantık geçerli olduğuna göre en anlamlı byte olan byte[3] ün bitleri aynı şekilde incelenebilir. Bu byte'taki bitler aşağıdaki gibi gösterilirse:

$$\text{byte[3]} \rightarrow \; t7\ t6\ t5\ t4\ t3\ t2\ t1\ t0 \qquad [5.6]$$

Basit algoritmada en anlamlı bit olan t7 nin durumu xor işleminin yapılıp yapılamayacağına karar vermekteydi. Aynı şekilde düşünüldüğünde t7=1 ise yazmaçtaki değer ile polinomal xor işlemine tabii tutulacaktır. Polinomalin bitlerinin (3 numaralı byte için) g7,g6,...g0 olarak gösterildiği düşünülürse algoritmanın ilk adımı aşağıdaki gibi olacaktır:

$$\begin{array}{r} t6\ \ t5\ \ t4\ \ t3\ \ t2\ \ t1\ \ t0\ ?? \\ + t7 * (g7\ g6\ g5\ g4\ g3\ g2\ g1\ g0) \end{array} \qquad [5.7]$$

[Kalan: + xor işelmini simgelemektedir]

İkinci adımda ise xor işleminin yapılıp yapılmayacağına t6 + t7*g7 bitinin değeri karar verecektir. Bu eşitliğe bakarak yeni en yüksek dereceli bitin oluşturulması için kendinden önceki iki bitin değerinin bilinmesi gerektiği söylenebilir. Aynı şekilde bir sonraki işlemi belirlemek için t7,t6 ve t5 bitlerinin bilinmesi gerekir. Bu şekilde genelleştirilerek; k. işlemin belirlenmesi için yazmaçtaki, en yüksek anlamlı k adet bitin değerinin bilinmesi gerekir denilebilir.

Sekizinci adımdaki işlemin hesaplanmak istendiği varsayılırsa yazmaçtaki en yüksek anlamlı 8 bitin iterasyona girmesi gerekir. Eğer bu işlemlerin daha önceden hesaplanmış ve kaydedilmiş değerlerle yapıldığı düşünülürse tablo ile CRC hesaplama yönteminin temeli anlaşılmış olur. Bu durumda üç önemli nokta belirtilebilir:

*Artık yazmaçtaki en yüksek dereceli byte 'ın bir anlamı yoktur. Polinomalin kaç kere kaydırma ve xor işlemlerinden geçirildiği de bir anlam ifade etmez.

*Geri kalan tüm bitler bir pozisyon sola kaydırılacaklardır.(Bir byte sola)

*Yazmaçtaki değer, daha önceden hesaplanmış bir byte uzunluğundaki kontrol değerine uygun olarak polinomal ile bir dizi xor işlemine tabii tutulur.

Bu işlemin daha iyi anlaşılabilmesi için, tablo oluşturma işleminin mantığı, bir örnekle vurgulanabilir:

Yazmaçtaki değerin 1001 1001 olduğu, ve polinomalin 1100 olduğu düşünülürse:
Kalan değerini bulmak bir önceki bölümde anlatılan basit algoritmanın adımları izlendiğinde

```
10011001
1100          →0 kaydırma
--------------
01011001
 1100         →1 bit sağa kaydırma
--------------
00111001
  1100        →2 bit sağa kaydırma
--------------
00001001
    1100      →4 bit sağa kaydırma
--------------
00000101      →Kalan değeri
```

Bu işlemin adım adım yapılması yerine yazmaçtaki ilk değerin “1” bitleri kullanılarak oluşturacak bir tablo ile polinomalin uygun miktarlarda kaydırılması ile elde edilecek 8 bitlik değer, bir seferde yazmaçtaki ilk değerle xor işlemine tabii tutulması ile de sonuca ulaşılabilir. Verilen örnekte yazmaçtaki ilk değer (10011001)

için kaydırma değerleri (0,1,2,4) olmalıdır. Yani:

```
1100          →0 kaydırma
 1100         →1 kaydırma
  1100        →2 kaydırma
    1100      →4 kaydırma
+_______
10011100      →tabloda bulunması gereken değer
```

olacaktır. Elde edilen bu değer direkt yazmaçtaki ilk değer ile xor işlemine tabii tutulursa:

```
10011001
10011100
--------------
00000101 →Kalan değeri
```

aynı kalan değerinin elde edildiği görülmektedir.
Tüm bu anlatılanların ışığında aşağıdaki algoritma oluşturulabilir:

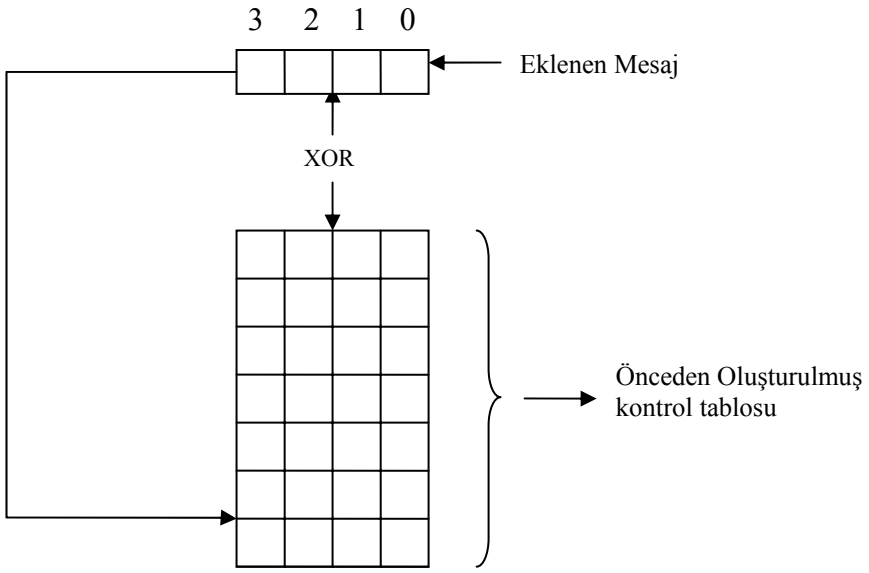
Çizelge 5.3. Tablo algoritmasının adımları

- Döngüye Başla
    - *Yazmacın en yüksek anlamlı byte değerini tespit et.
    - Bu değeri kullanarak kontrol byte' ını oluştur.
    - Kontrol byte' ındaki değeri kullanarak polinomalde uygun kaydırma ve toplama işlemleri yap ve elde edilen yeni polinomal ile yazmaçtaki değeri xor işleminden geçir.
    - Yazmaçtaki değeri mesaj ile besleyerek bir byte sola kaydır.
    - Eğer bir önceki adımda yazmaçtan taşma olmuşsa (yani sola kayarak yazmaçtan çıkan bit "1" değerinde ise) :

      Yazmaç=Yazmaç xor Polinomal

      işlemini yap.
- Mesaj Sonu ise Döngüden Çık.

Yukarıdaki algoritmanın şematik gösterimi aşağıdaki gibi olmalıdır:



Şekil 5.4. Tablo algoritmasının şematik gösterimi

Algoritmanın C/C++ ile ifadesi ise aşağıdaki şekilde olacaktır:

```
r=0;
while (len--)
  {
   byte t = (r >> 24) & 0xFF;
   r = (r << 8) | *p++;
   r^=tablo[t];
   }
```

Burada *len*: mesajın byte cinsinden uzunluğunu, *t*: geçici bir değişkeni, *tablo*: oluşturulan tabloyu göstermekte ve *p*: mesajın eklenecek byte'ının hafızadaki yerini gösteren işaretçidir.

5.3. Algoritmayı Tanımlamak

Algoritmanın yapısının oluşturulması için gerekli olan adımların belirtilmesi yerinde olacaktır. Bu amaçla, öncelikle CRC algoritmalarını birbirinden ayırt eden noktaların belirtilmesi gerekir. Aşağıdaki belirtilen hususlar algoritmanın oluşturulmasında baz alınmalıdır:

- Polinomalin genişliği

- Polinomalin değeri
- Yazmaç'ın başlangıç değeri
- Yazmaçta kalan son değer ile xor işleminin yapılacağı değer

5.3.1. CRC algoritmaları için parametrik model

Bir önceki bölümde amaca uygun CRC algoritması geliştirmek için göz önünde bulundurulması gereken bazı hususlar sunulmuştu. Bu bölümde ise CRC algoritması için kullanılacak parametreler genel olarak ele alınmış ve yaygın olarak kullanılan isimleri ile belirtilmiştir.

İsim: Algoritmanın temel özelliklerini içerisinde barındıran bir isim.

Width: Polinomalin fiziksel ve bit cinsinden genişliğinin bir eksiği.

Poly: Polinomalin değeri. Bu değer aslında ikili sistemde seçilmiş olup 16'lık sistemde ifade edilmelidir. "Poly" değeri belirlenirken polinomalin en anlamlı biti ihmal edilir. Örneğin polinomal değeri 10110 ise poly değeri 0x06 olarak düşünülmelidir.

Init: Algoritma başladığında yazmaçta bulunan başlangıç değeri.

RefIn: Bu değer, algoritmaya giren byte değeri için en anlamlı bitin yerleşimi bakımından farklılık gösteren işlemci mimarilerinin seçilmesine imkan veren bir bayrak değeridir. Big-Endian olarak bilinen ve byte içerisinde en anlamlı bitin 7.bit olduğu mimari için; yanlış (FALSE), Little-Endian olarak bilinen ve byte içerisinde an anlamlı bitin 0. bit olduğu mimari için; doğru (TRUE) değerlerine sahip olmalıdır.

RefOut: Bu değer, yazmaçta bulunan son değerin direkt olarak xor işlemine mi gireceğini yoksa önce yansıma (reflection) işlemine mi tabii tutulacağını belirleyen bayrak değeridir. Doğru (TRUE) ise yansıma işlemi yapılır. Yanlış (FALSE) ise doğrudan xor işlemine geçilir.

XorOut: Yazmaçta kalan son değer ile xor işlemine tabii tutulacak değeri barındırır.

Bu temel parametre değerleri popüler CRC algoritmalarından CRC-16 algoritması için aşağıda gösterildiği gibidir:

İsim : CRC-16
Width : 16
Poly : 8005
Init : 0000
RefIn : TRUE
RefOut : TRUE
XorOut : 0000

Aşağıdaki çizelgede yaygın olarak kullanılan bazı CRC algoritmalarının parametrelerini gösterilmektedir:

Çizelge 5.4. Bilinen bazı CRC Algoritmalarının parametreleri

İsim	**CRC-16**	**CRC-16/CITT**	**XMODEM**	**ARC**	**CRC-32**
Width	16	16	16	16	32
Poly	8005	1021	8408	8005	04C11DB7
Init	0000	FFFF	0000	0000	FFFFFFFF
RefIn	TRUE	FALSE	TRUE	TRUE	TRUE
RefOut	TRUE	FALSE	TRUE	TRUE	TRUE
XorOut	0000	0000	0000	0000	FFFFFFFF

6. PROGRAMIN YAPISI VE ÇALIŞMASI

Bu bölümden itibaren geliştirilmeye çalışılan protokol daha yakından incelenerek daha önceki bölümlerde verilen teorik bilgiler uygulamaya koyulacaktır.

Güvenilirliği, yaygın kullanımı ve esnek yapısı göz önüne alındığında RSA algoritmasının geliştirilecek protokolde kullanılması uygun görülmüştür.

Açık anahtarlı şifreleme hiç şüphesiz uygulamalı matematiğin 20. yüzyılda yaşadığı en büyük gelişmelerden biridir. Geleneksel şifrelemede "anahtar" olarak tanımlanmış bir sayı hem şifreleme hem de deşifreleme işlemleri için kullanılır. Bir A kullanıcısı B kullanıcısına geleneksel şifreleme yöntemi kullanarak bir mesaj göndermek istediğinde öncelikle her iki kullanıcı da sadece kendilerinin sahip olacağı bir anahtar üzerinde mutabık kalmalıdır. Daha sonra A kullanıcısı bu anahtarı mesajı şifrelemek için ve B kullanıcısı ise aynı anahtar ile mesajı deşifrelemek için kullanacaktır. Böyle bir iletişimin güvenli olduğunu söyleyebilmek için iletişim hattını dinleyebilecek üçüncü bir C kullanıcısının şifrelenmiş mesajı ele geçirebilmesine rağmen onu çözümlemesinin mümkün olmaması gerekir. Aşağıdaki şartları yerine getirebilen bir metodun güvenli olduğundan bahsedilebilir:

1. Anahtar oluşturma işlemi makul sayılabilecek derecede kolay olmalıdır.
2. Anahtar yeterince uzun olmalı ve bir üçüncü kişinin tahminini önleyebilecek derecede rasgele seçilmelidir.
3. Şifrelenmiş mesaj, orijinal mesajın bir parçası bilinse dahi çözülememelidir.
4. Anahtar tamamıyla gizli tutulmalıdır.

Geleneksel şifreleme yöntemini esas alan bir çok algoritma mevcuttur. Bazılarının diğerlerine göre daha üstün olduğu söylenebilse de hepsinin en önemli problemi iletişime katılacak N kullanıcı düşünüldüğünde her bir mesaj aktarımı için bir anahtara ihtiyaç duyulacağından N(N-1)/2 adet anahtarın üretilmesi gerekir. Kullanıcı sayısının artması anahtar üretme ve anahtar dağıtım işlemlerinin zorlaşması demektir.

Açık anahtarlı şifreleme sistemi düşünülürse her kullanıcının kendine ait, biri şifreleme işlemlerinde diğeri deşifreleme işlemlerinde kullanılmak üzere iki farklı anahtarı vardır ve açık anahtar olarak adlandırılan anahtar diğer N-1 kişiye duyurulur. Diğer anahtar ise özel anahtar olarak adlandırılır ve gizli tutulur.

Açık anahtarlı şifreleme sisteminin temel olarak aşağıdaki özellikleri barındırması gerekir:

1. Anahtar çiftlerinin üretilmesi makul sayılabilecek derecede kolay olmalıdır.
2. Anahtarlar yeterince uzun olmalı ve bir üçüncü kişinin tahminini önleyebilecek derecede rasgele seçilmelidir.
3. Şifrelenmiş mesaj, orijinal mesajın bir parçası bilinse dahi çözülememelidir.
4. Özel anahtar tamamıyla gizli tutulmalıdır.

A kullanıcısının B kullanıcısına açık anahtarlı şifreleme yöntemi ile bir mesaj göndermesi aşağıdaki adımlar izlenilerek yapılmalıdır:

1. A, geleneksel şifreleme metotları yardımı ile rasgele bir anahtar seçer. Bu anahtar oturum anahtarı olarak kullanılacaktır.
2. A, B' nin açık anahtarını oturum anahtarını şifreleme için kullanır.
3. A, şifrelenmiş oturum anahtarını B' ye gönderir.
4. A, oturum anahtarını kullanarak bir mesajı şifreler ve B' ye gönderir.
5. B özel anahtarını kullanarak A'nın gönderdiği şifrelenmiş oturum anahtarını deşifreleyerek oturum anahtarını elde eder.
6. B şifrelenmiş mesajı oturum anahtarını kullanarak deşifre eder.

Bu algoritma her ne kadar mesajın içeriğinin bir üçüncü kişiden korunmasını sağlasa da mesajın beklenen kişiden yani *A* kullanıcısından gelip gelmediğini sorgulamamaktadır. Bu şemaya göre bir *C* kullanıcısının da *A* ile aynı adımları izleyerek *B* ile konuşması mümkündür. Bunu önlemek amacı ile 2. ve 5. adımlar aşağıdaki gibi modifiye edilmelidir:

2. *A* oturum anahtarını önce kendi özel anahtarını ve daha sonra *B*' nin açık anahtarını kullanarak şifreler.
5. *B* önce kendi özel anahtarını ve daha sonra *A*' nın açık anahtarını kullanarak şifrelenmiş oturum anahtarı bilgisini deşifre ederek oturum anahtarını elde eder.

Bu adımlar takip edildiğinde eğer *A*' nın özel anahtarı ele geçirilmemişse *B* mesajın *A* dan geldiğine emin olacaktır.

Bu noktada *A* kullanıcısının niçin direkt *B* nin açık anahtarını kullanarak mesajı şifrelemek yerine bir oturum anahtarı kullandığı düşünülebilir. Bunun temel iki nedeni vardır:

1. Açık anahtarlı şifreleme sistemi geleneksel yönteme nazaran daha yavaş kalmaktadır ve tüm mesajın açık anahtar kullanılarak şifrelenmesi ve deşifrelenmesi daha çok zaman alacaktır.
2. İletişimi izleyen üçüncü bir *C* kullanıcısı orijinal mesajın bir kısmını biliyor olabilir ve daha önce herkese duyurulan açık anahtarın ve bu bilginin yardımı ile orijinal mesajı elde etmesi kolaylaşır.

Tüm şifreleme algoritmalarının bir zaafı vardır ve tabi ki önerilen bu metodun bir istisna oluşturması beklenmemelidir. Bahsedilen protokolde gözlenebilecek ilk zayıf nokta oturuma başlarken *A* kullanıcısının ürettiği oturum anahtarının yeterince rasgele olmaması olabilir. İletişimi dinleyen *C* kullanıcısı üretilebilecek rasgele sayıları makul bir miktarda hesaplayıp bu sayılarla bir tablo oluşturur, *B* nin açık anahtarı ile bu tablodaki anahtarları şifreler ve oluşturduğu bu mesajları *A*' nın ürettiği ile karşılaştırırsa oturum anahtarını elde etmiş olacaktır. Bu anlatılan yöntemin hemen hemen aynısı popüler Netscape tarayıcısının şifreleme sisteminin kırılması için kullanılmıştır.

6.1. RSA Algoritmasının Açıklanması

p ve *q* birbirinden farklı ve büyük iki asal sayı olsun. Bir tamsayının asal olup olmadığı Fermat' ın teoremi kullanılarak test edilebilir. Bu teorem her pozitif *a* tamsayısı ile asallığı sorgulanan sayı arasında aşağıdaki ilişkinin olması gerekliliğini arar:

$$a^{p-1} \equiv 1 \ (mod\ p) \qquad [6.1]$$

p tamsayısı bu denklik testinden yeterli miktarda farklı a değeri için geçerse asal olma olasılığı yüksektir denir.

p ve *q* asal sayılarının bu şekilde oluşturulmalarının ardından iki büyük *d* ve *e* tamsayıları bulunur. Her ikisinin de *(p-1)(q-1)* çarpımından küçük olması ve aşağıdaki eşitliğe uygun olması gerekir;

$$de \equiv 1 \ (mod\ (p-1)(q-1)) \qquad [6.2]$$

Bunu gerçekleştirmenin en kolay yolu *e* yi rasgele seçerek *OBEB(e, (p-1)(q-1))* , *d* ve *s* değerlerini Öklit algoritması kullanarak hesaplamaktır. Böylece aşağıdaki eşitlik oluşturulabilecektir;

$$de - s(p-1)(q-1) = gcd(e, (p-1)(q-1)) \qquad [6.3]$$

Bu değerler yardımıyla açık anahtar; *(e,pq)* ve özel anahtar; *(d,pq)* oluşturulur. *e* ve *d* üs olarak, *pq* çarpımı ise modül olarak kullanılır. *pq* çarpımı yani modül açık anahtar ile birlikte duyurulur fakat *p* ve *q* değerleri gizli tutulur. Aslında *p* ve *q* değerleri şifreleme veya deşifreleme işlemlerinde direkt olarak kullanılmamaktadır fakat bu sayıların elde edilmesi ile *e* ve *d* değerlerinin bulunması çok kolaydır.

m değeri [0,*pq*) aralığındaki şifrelenmek istenen mesajı simgelesin. RSA algoritması kullanılarak bu değer [0,*pq*) aralığındaki şifrelenmiş *c* değerini aşağıdaki gibi oluşturur:

$$c \equiv m^e \ (mod\ pq) \qquad [6.4]$$

Deşifreleme işlemi ise benzer şekilde *d* sayısı kullanılarak yapılır.

$$m \equiv c^d \ (mod\ pq) \qquad [6.5]$$

Şifreleme ve deşifreleme işleminin matematiksel anlamda da birbirinin tersi olduğu görülmektedir ve aşağıdaki gibi ispatlanabilir:

$$(m^e)^d \equiv (m^d)^e \equiv m^{de} \equiv m \ (mod\ pq) \qquad [6.6]$$

d ve *e* değerleri aşağıdaki gibi oluşturulduğundan

$$de = 1 + s(p-1)(q-1) \qquad [6.7]$$

Eğer *p* sayısı *m* sayısını bölemiyorsa Fermat teoremine göre:

$$m^{p-1} \equiv 1 \ (mod\ p) \qquad [6.8]$$

Her iki tarafa $s(q-1)$ ile üs alma ve *m* ile çarpma işlemleri uygulandığında;

$$m^{de} = m^{1 + s(p-1)(q-1)} \equiv m \ (mod\ p) \qquad [6.9]$$

bu eşitlik aynı zamanda *m*'nin *p* sayısına bölünebildiği durumlarda da geçerlidir. Benzer şekilde:

$$m^{de} \equiv m \ (mod\ q) \qquad [6.10]$$

Dolayısı ile hem *p*'nin hem de *q*'nun $(m^{de} - m)$' i böldüğü anlaşılır. *p* ve *q*'nun aralarında asal olduğu bilindiğinden:

$$m^{de} \equiv m \ (mod\ pq) \qquad [6.11]$$

yazılabilir.

6.2. RSA Algoritmasının C++ ile Uygulaması

Bu bölüme kadar geliştirilmesi planlanan protokol için güvenlik ihtiyacını karşılayabilecek yöntemler incelenmiş ve bu yöntemler arasında en uygun görülen RSA algoritmasının uygulanması için gerekecek tüm matematiksel ve kuramsal

bilgiler verilmiştir. Bu bölümde RSA algoritmasının; Microsoft Visual C++ 6.0 yardımı ile Windows işletim sistemi altında herhangi bir hazır şifreleme kütüphanesi kullanmaksızın uygulaması anlatılacaktır. Okuyucuların, temel programlama bilgisi, C ve C++ dillerine aşina oldukları varsayılmıştır.

Daha öncede ayrıntılarıyla açıklandığı gibi RSA açık anahtarlı şifreleme algoritması etkisini kullandığı anahtarların uzunluğundan almaktadır. Bu bakımdan RSA için en temel ihtiyacın büyük tamsayılarla modüler aritmetik işlemleri yapabilmek olduğu söylenebilir. Fakat bilindiği gibi günümüzde bilgisayar sistemlerinde kullanılan işlemcilerin yazmaçları dahi algoritmayı karşılayacak uzunlukta tamsayıları donanımsal anlamda destekleyemezler. Kullanılan derleyicinin ve çalışılan işlemcinin 32 bitlik işlemler için tasarlandığı düşünülürse daha büyük tamsayılar oluşturmak ve bu tamsayılarla modüler işlemler gerçekleştirmek için yazılımsal desteğe ihtiyaç olacaktır.

Bu ihtiyacı karşılamak amacıyla büyük sayıları oluşturmayı ve işlem yapabilmeyi sağlayacak bir c++ sınıfı yazılmıştır. Öncelikle, "large_int" ismi verilen bu sınıfın, ayrıntılarıyla anlatımı sunulacak, böylece ileriki kısımlardaki büyük sayıların kullanımı daha rahat anlaşılabilecektir.

Bu sınıfın tanımı ve fonksiyonların prototipi Ek-1'de sunulmuştur.

6.2.1. Büyük sayı sınıfının açıklanması

Bu sınıf yapı itibariyle işaretsiz 16 bitlik tamsayıların oluşturduğu kümenin tekbir tamsayıymış gibi işlemlere girmesini sağlayarak bu sayede lüzumlu algoritmaların işlenmesine olanak verecek şekilde yazılmıştır. Dolayısı ile sınıfın temel birimi;
*unsigned short *value;*
şeklinde tanımlanan value göstericisidir. İhtiyaç duyulan büyüklükte bir sayı oluşturabilmek için 16 bit cinsinden uzunluğunun tutulacağı
unsigned int uzunluk;
sınıfın ikinci önemli değişkenidir denebilir.

Bu sınıfa ait bir nesne oluşturulurken kullanılacak iki farklı kurucu fonksiyondan ilki oluşturulacak sayının uzunluğunu, 16 bitlik bir tamsayının katı olarak beklerken diğeri ise yine bu sınıftan bir tam katı olarak verilmesini bekler.

```
large_int::large_int(unsigned len){
  uzunluk = len;
  value = new unsigned short[len]; }

large_int::large_int(const large_int &n) {
  uzunluk = n.uzunluk;
  value = new unsigned short[uzunluk];
  unsigned i;
  for (i = 0; i < uzunluk; i++)
    value[i] = n.value[i];
}
```

verilen uzunluk değeri linker, tarafından uygun kurucu fonksiyona gönderilerek gerekli hafıza alanının ayrılmasında kullanılır.

Aşağıda gösterilen yıkıcı fonksiyon ise ayrılan bu alanın tekrar kullanılabilir hale getirilmesi için boşaltılmasını sağlar.

```
large_int::~large_int(){
delete [] value;}
```

Ayrıca, büyük sayılarda özellikle çarpma işlemlerinde kullanılmak üzere sayıyı verilen bit uzunluğunda sola kaydırmaya yarayan shift fonksiyonu tanımlanmıştır. Sınıfta, büyük sayıların birbirleriyle işlemlerini sağlayacak hemen hemen tüm matematiksel operatörlerin aşırı yüklemeleri de tanımlanmıştır. Bu sayede programın anlaşılabilirliği arttırılmıştır. Programda özellikle mod alma işlemlerinde çokça kullanılan "%" operatörünün de diğer aşırı yüklenmiş operatörler gibi hem 16 bitlik değerler hem de büyük sayı cinsinden nesneler için işlem yapabilecek iki farklı aşırı yüklenmiş fonksiyonu vardır.

Divide, Edit, Dump gibi üye fonksiyonlar bu aşırı yüklenmiş operatörlerin kullanımı için yazılmış olmalarına rağmen "public" olarak tanımlandıklarından harici olarak da erişilebileceklerdir.

RSA algoritması için en çok kullanılması gereken işlemlerden biride üs alma işlemi olacağından sınıfa, büyük sayılar için üs alma işlemini gerçekleştirecek "power" fonksiyonu da eklenmiştir. Bu fonksiyonun tüm parametreleri büyük sayı cinsinden verildiğinden yapılacak üs alma işleminin uzunluk sınırlaması bulunmamaktadır.

```
void large_int::Power(const large_int &base, const large_int &exponent,
  const large_int &modulus, large_int &result){
  large_int r(2*base.uzunluk+1);
  r = 1;
  bool one = true;
  unsigned i = exponent.uzunluk;
  while (i-- != 0)  {
    unsigned bit = 1 << 15;
    do    {
      if (!one)      {
        large_int n(r);
        r *= n;
        r %= modulus;      }
      if (exponent.value[i] & bit)      {
        r *= base;
        r %= modulus;
        one = false;      }
      bit >>= 1;
    } while (bit != 0);
  }
result = r;
}
```

Yukarıda gösterildiği gibi bu fonksiyon tamamen RSA için özelleştirilerek verilen taban, üs ve modül değerlerine göre yaptığı hesaplamaları, sonuç büyük sayı değişkenine atamaktadır.

Yapılacak işlemler için uzunluğu taban uzunluğunun iki katından bir fazla olan r büyük sayı değişkeni oluşturulmuştur. İşlemin kabarıklığının önlenmesi amacıyla matematiksel önbilgi bölümünde verilen modüler aritmetiğin dağılma özelliğinden yararlanılmış ve bu sayede tüm üs alma işlemini yaptıktan sonra modül alma işlemi yapmak yerine her bir üs alma işleminden sonra modül alma işlemi yapılarak sayının çok fazla büyümesi önlenmiştir. İkinci önemli nokta, üs alma işleminin tekrarlı kare alma yöntemi ile basitleştirilmesidir. Bunu yapabilmek için üssün bitleri en değerli bitten başlamak üzere test edilir. Her bir 2' nin tam üssü için tabanın karesi alınırken

sonuçta artan değerler kadar taban değerinin sonuçla çarpımı yapılır. Böylelikle sadece artan değerler için normal üs alma işlemi yapılmış olur.

6.2.2. RSA fonksiyonlarının açıklanması

Bu bölümde RSA şifrelemenin ihtiyaç duyduğu algoritmaların bulunduğu fonksiyonların açıklamaları yapılacaktır. Öncelikle şifreleme için gerekli anahtarların üretilmesi daha sonra bu anahtarlar kullanılarak bir mesajın şifrelenebilmesi için kullanılan fonksiyonlar anlatılacaktır.

Anahtar üretimi göz önüne alındığında ihtiyaç duyulacak ilk şey asal sayıların üretimi olacaktır. Bunu gerçekleştirmek için öncelikle rasgele bir sayı seçilmekte daha sonra bu sayının komşuluklarında asal bir sayı aranmaktadır. Bulunan sayıların asal olup olmadığının test edilebilmesi için kullanılan AsalTest() fonksiyonu burada açıklanacak, rasgele sayı seçme işlemi ise daha sonra ayrıntılarıyla anlatılacaktır.

```
static bool AsalTest(const large_int &p){
 large_int pminus1(p);
 pminus1 -= 1;
 unsigned count = 101;
 while (--count != 0)  {
  large_int r(p.uzunluk);
  large_int x(p.uzunluk);    {
   for (unsigned i = 0; i < x.uzunluk; i++)
    x.value[i] = rand() << 8 | rand();    }
  x %= p;
  if (x != 0)    {
   large_int::Power(x, pminus1, p, r);
   if (r != 1)
    return false;    }  }
    return true;
   }
```

Fonksiyonda daha önceki bölümlerde anlatılan Fermat teoremi uygulanmaktadır. Bu fonksiyonda bir sayının asal sayılabilmesi için asallık testine 100 kez girmesi gerekmektedir. İterasyonun arttırılması sayının asal olma ihtimalini, dolayısıyla kesinliği arttıracaktır.

AsalTest fonksiyonu anahtarların üretimleri sırasında AsalUret ve AsalUret fonksiyonunu kullanan AnahtarUret fonksiyonları tarafından çağırılmaktadır.

AsalUret fonksiyonu parametre olarak aldığı büyük sayı türünden değişkene önce rasgele bir değer atar. Daha sonra AsalTest fonksiyonunu çağırarak bu rasgele sayının asallığını test eder. Çift sayıların asal olması mümkün olmayacağından sadece tek sayılar için deneme yapılır. Bu bakımdan asallığı AsalTest fonksiyonu tarafından onaylanmayan sayılar iki arttırılarak tekrar teste tabi tutulur. Bu şekilde asal bir sayı bulununcaya kadar döngüye devam edilir.

Anahtar üretimi sırasında karşılaşılan çok önemli bir başka husus aralarında asal sayıların bulunmasıdır. Hatırlanacağı gibi d ve e sabitlerinin Φ ile aralarında asal olacak şekilde seçilmeleri gerekiyordu. Bu durumun kontrolünün ise Euclid algoritmasıyla sağlandığı daha önce anlatılmıştı. Bu denetleme programda EuclideanAlgorithm fonksiyonu tarafından yapılmakta ve anahtar üretimi sırasında AnahtarUret fonksiyonu tarafından çağırılarak kullanılmaktadır. Bu fonksiyon işleyiş olarak matematiksel önbilgi bölümünde verilen algoritmanın adımlarını aynen izlemektedir. Böylece girdi olarak aldığı iki büyük sayının Ortak Bölenlerinin en Büyüğü' nü bularak bir diğer büyük sayı olan g ye atamaktadır.

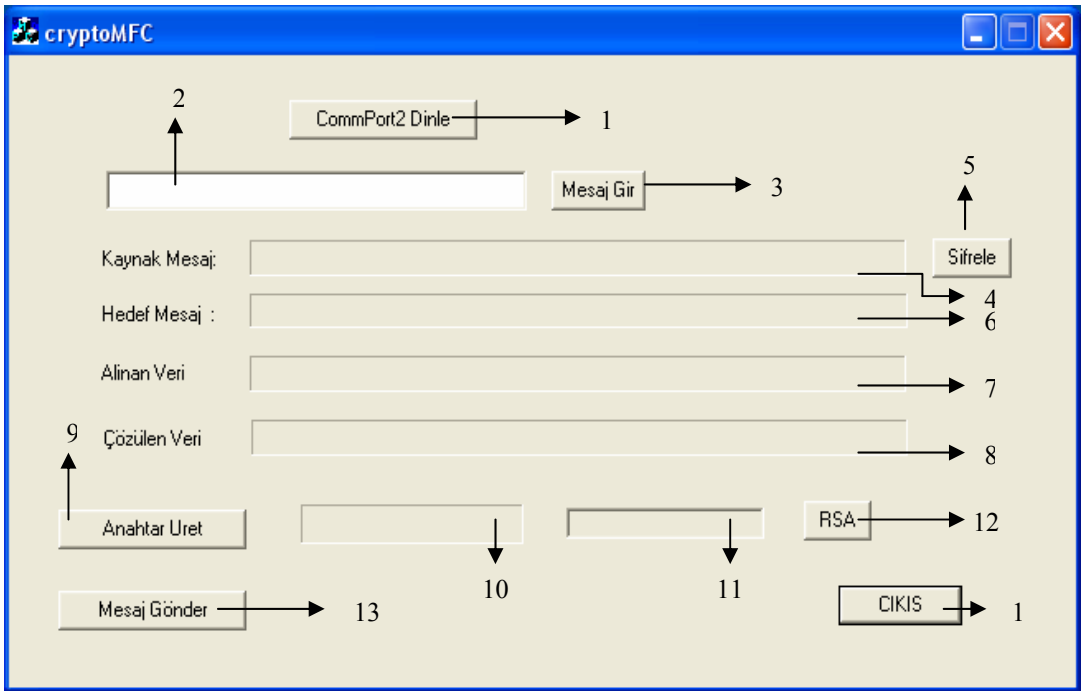
6.3. Programın Çalışması

6.3.1. Arabirim

Tasarlanan protokolün Visual Studio .NET C++ 7.0 ile yazılan uygulaması, seri port üzerinden yarattığı iletişim kanalını, simülasyon yapmak için kullanmaktadır. Programın tam anlamı ile çalışabilmesi için Windows 9x,Me,XP veya 2000 tabanlı ve 2 adet seri (COMM) kanala sahip bir bilgisayar kullanılmalıdır. 1 numaralı seri iletişim kanalı protokole uygun olarak hazırlanan veriyi göndermek için kullanılırken, 2 numaralı seri iletişim kanalı bu veriyi almak için kullanılır. Anlaşılacağı gibi program, aslında bir host ve birçok slave cihazdan oluşabilecek

protokolün simülasyonunu tek bir bilgisayar üzerinde yapılabilmesine imkan sağlayacak şekilde düşünülmüştür.

Program çalıştırıldığında aşağıdaki ana arabirim ile açılır:



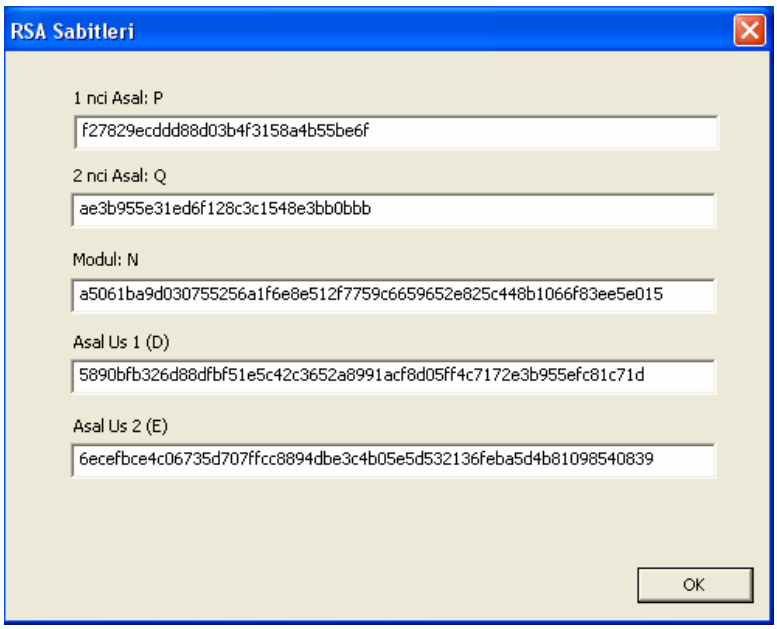
Şekil 6.1. Programın ana arabirim ekran görüntüsü

Şekilde gösterilen arabirimde, numaralandırılarak gösterilen elemanların açıklamaları aşağıda verilmiştir:

1. 2 numaralı iletişim protokolünün, program için rezerv edilmesini sağlayan butondur. Hazırlanan verinin 1 numaralı kanal üzerinden gönderilmesinden evvel bu buton ile COMM 2 kanalı hazır hale getirilmelidir.
2. Kullanıcının giriş yapması için yerleştirilmiş düzenleme kutusu. Girilen mesaj en fazla anahtar uzunluğu kadar olmalıdır.(256 bit için 32 karakter).
3. Düzenleme kutusuna girilen mesajın protokole girilmesini sağlayan buton.
4. Düzenleme kutusunda hazırlanarak "Mesaj Gir" butonu ile protokole girilen mesajın kaynak mesaj olarak gösterilmesini sağlayan etiket kutusu.

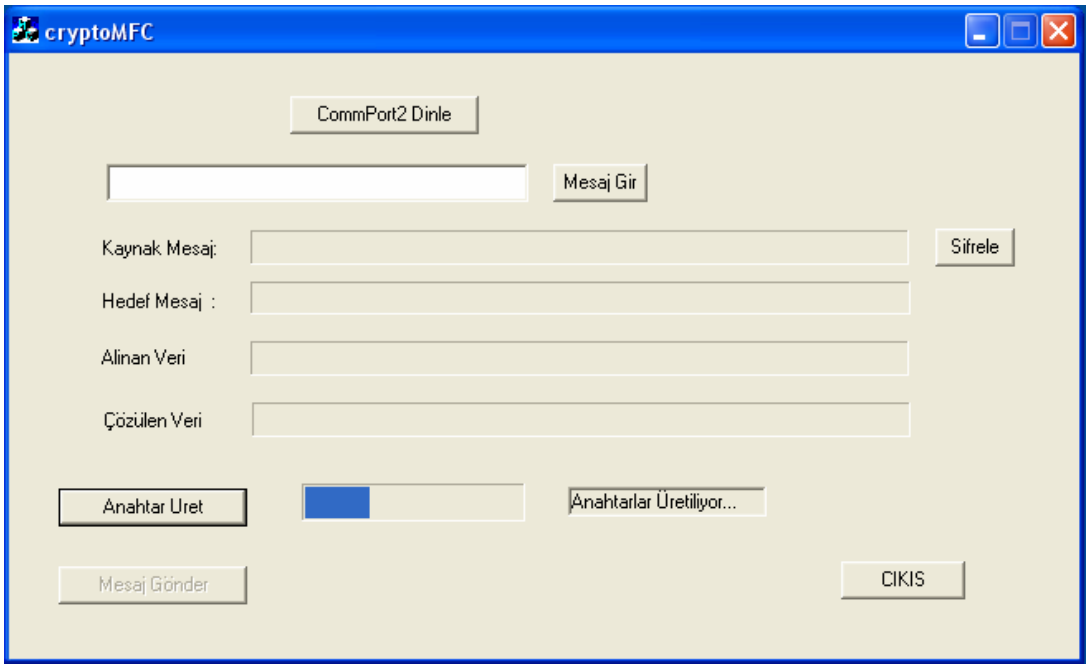
5. Girilen mesajın RSA algoritmasına uygun olarak şifrelenerek gönderilmeye hazır hale getirilmesini sağlayan buton. Bu buton kullanılmadan önce RSA şifreleme için gereken anahtarların üretilmesi gerekir.
6. Şifrelenerek gönderilmeye hazır hale getirilen mesajın son halini gösteren etiket kutusu.
7. 2. iletişim kanalından alınan verinin gösterildiği etiket kutusu.
8. Alınan verinin deşifreleme işleminin ardından elde edilen mesajın gösterildiği etiket kutusu.
9. RSA algoritmasının uygulanabilmesi için ihtiyaç duyulan anahtar değerlerinin üretilmesini sağlayan buton. Yeni bir iletişim oturumu gerektiren durumlar için anahtarlar yeniden üretilmelidir.
10. Anahtar üretme işleminin hangi aşamada olduğunun anlaşılmasını sağlayan durum gösterici.
11. Anahtar üretme işleminin fazlarını kullanıcıya bildiren etiket kutusu.
12. "Anahtar Üret" butonu yardımıyla oluşturulan anahtar değerlerinin ve diğer RSA sabitlerinin yer aldığı RSA Sabitleri diyalog kutusunun açılmasını sağlayan buton. Bu buton şifreleme işlemlerinin gerçekleştirilebilmesi için gerekli olan anahtarlar üretilinceye kadar görünmeyecektir.
13. Şifrelenmiş verinin 1 numaralı iletişim kanalı üzerinden gönderilmesini sağlayan buton. Bu buton, başarılı bir şifreleme işlemi yapılana kadar erişilebilir değildir.
14. Programın sonlandırılmasını sağlayan buton.

RSA sabitlerinin gösterildiği diyalog kutusu ise Şekil 6.2.'de gösterilmiştir.



Şekil 6.2. RSA sabitleri diyalog kutusu ekran görüntüsü

6.3.2. Anahtarların üretilmesi



Şekil 6.3. Anahtarların üretilmesi sırasında programın ekran görüntüsü.

Şifreleme ve Deşifreleme işlemlerinin gerçekleştirilebilmesi için öncelikle Genel ve Özel anahtarların üretilmesi gerekir. Programda bu işlemi yapmak için 9 numara ile

gösterilen “Anahtar Üret” butonu kullanılmalıdır. Oturuma başlamadan önce bu işlemin yapılması gerekir.

“Anahtar Üret” butonuna basıldığında öncelikle aşağıdaki fonksiyon çağırılır:

```
void CCryptoMFCDlg::OnBUTTONAnahtarUret()
{
CProgressCtrl myCtrl;
RSA_Button.ShowWindow(SW_HIDE);
progress_control.StepIt();
SetDlgItemText(IDC_STATIC_data_deneme,"Anahtarlar
Üretiliyor...");
AnahtarUret(d1, e1, n1);
SetDlgItemText(IDC_STATIC_data_deneme,"Sayılar Uretildi
!!!");
RSA_Button.ShowWindow(SW_SHOW);
}
```

Öncelikle, daha önce oluşturulan RSA sabitleri geçersiz olduğu için RSA butonu gizlenmektedir. Ardından işlemin başladığının kullanıcıya bildirilmesi amacıyla durum gösterici bir aşama ilerletilmiş ve “Anahtarlar Üretiliyor” mesajı gösterilmiştir. Global olarak tanımlanan, büyük sayı sınıfından türetilmiş: d1,e1 ve n1 nesneleri uygun değerlerin atanması amacıyla “AnahtarUret()” fonksiyonuna parametre olarak gönderilmiştir. Bu fonksiyon aşağıdaki gösterilmektedir:

```
void AnahtarUret(large_int &d, large_int &e, large_int &n)
{
      char *KeyBuffer;
      char strBuffer[50];
  large_int p(d.uzunluk/2);
  AsalUret(p);
      ...
      //…Sayıyı RSA Diyalog kutusunda gösterecek şekilde stringe ata

  large_int q(d.uzunluk/2);
  AsalUret(q);
      ...
      //…Sayıyı RSA Diyalog kutusunda gösterecek şekilde stringe ata


  large_int pp(p);  pp -= 1;
  large_int qq(q);  qq -= 1;
  large_int pq(d.uzunluk);  pq = pp;  pq *= qq;  n = p;  n *= q;
      ...
      //…Sayıyı RSA Diyalog kutusunda gösterecek şekilde stringe ata

  Random(d);
  d %= pq;
```

```
  large_int temp(d.uzunluk);
  large_int g(d.uzunluk);

  while (true){
    EuclideanAlgorithm(d, pq, e, temp, g);
    if (g == 1)
      break;
    d += 1;
  }
      ...
      //…Sayıyı RSA Diyalog kutusunda gösterecek şekilde stringe ata


  prog_control->StepIt();
}
```

Daha önce RSA şifrelemede anlatıldığı gibi öncelikle p ve q asal sayılarının bulunması gereklidir. Daha sonra bu sayılar kullanılarak anahtar üretimi tamamlanır. Burada çağırılan "AsalUret" fonksiyonu rasgele seçilen bir sayının üzerinde yaptığı değişikliklerle asal bir sayı üretmeyi amaçlamaktadır. Aşağıdaki yapıda verilen bu fonksiyonun iki önemli safhası vardır:

```
static void AsalUret(large_int &p)
{
  Random(p);
  p.value[p.uzunluk-1] |= 0x8000;
  p.value[0] |= 1;
  while (!AsalTest(p))
    p += 2;
}
```

İlki "Random()" fonksiyonu ile rasgele bir sayı seçilmesi, ikincisi ise bu sayıdan hareketle "AsalTest()" fonksiyonu yardımıyla asal bir sayı oluşturulmasıdır.

Bölüm 3.7 de RSA algoritmasının güvenliği anlatılmış ve Random fonksiyonunun önemi belirtilmiştir. Bu bakımdan standart Windows fonksiyonları yeterli değildir. Aşağıdaki Random ve random_ozel fonksiyonları yazmaç değerlerinin değişmediği durumlarda dahi algoritma için yeterli sonuçlar vermektedir.

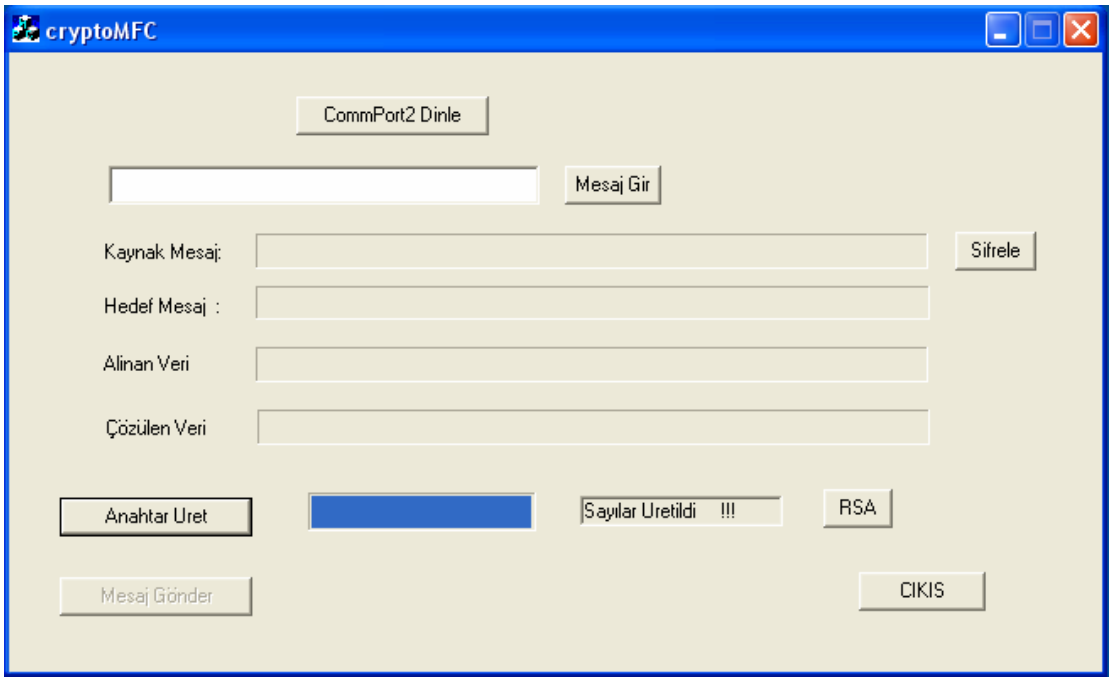
```
double random_ozel(long *idum)
{
      int j;long k;double temp;
      if(*idum <= 0 || !iy)   {
            if(-(*idum) < 1)
                  *idum = 1;
```

```
            else
                  *idum = -(*idum);
            for(j = NTAB + 7; j >= 0; j--){
                  k = (*idum) / IQ;
                  *idum = IA * (*idum - k * IQ) - IR * k;
                  if(*idum < 0)
                        *idum += IM;
                  if(j < NTAB)
                        iv[j] = *idum;    }
            iy = iv[0];
      }

      k = (*idum) / IQ;
      *idum = IA * (*idum - k * IQ) - IR * k;
      if(*idum < 0)
            *idum += IM;
      j = iy / NDIV;
      iy = iv[j];
      iv[j] = *idum;
      if((temp = AM * iy) > RNMX)
            return RNMX;
      else
            return temp;
}

void Random(large_int &x)
{
int random_buffer;
double vrb_random;
unsigned short random_value;

      for (unsigned i = 0; i < x.uzunluk; i++){
      srand( (unsigned)time( NULL ) );
      random_buffer=rand()+(0x5)*i;
      vrb_random = random_ozel((long *)&random_buffer);
      random_buffer=(int)(1000000000*vrb_random);
            random_value=(unsigned short)(random_buffer);
            random_value +=(unsigned short)(random_buffer>>16);

      x.value[i] = random_value;
      }
}
```

Bu fonksiyonlar yardımıyla üretilen sayıdan asal sayı elde edilebilmesi için AsalTest fonksiyonu çağırılmakta ve bu fonksiyon, Fermat teoremini kullanarak verilen sayının asal olup olmadığını test etmektedir. Programda sayının asal sayılabilmesi için Fermat teoremine uygunluğunun 100 defa test edilmesi uygun görülmüştür. Bu miktarın azaltılması anahtarların üretilmesi işleminin hızlandırılmasını sağlayacaktır.

```
static bool AsalTest(const large_int &p)
{
  large_int pminus1(p);
  pminus1 -= 1;
  unsigned count = 101;
  while (--count != 0)  {
        if(count==80 || count==50 || count==30 || count==10)
              prog_control->StepIt();

    large_int r(p.uzunluk);
    large_int x(p.uzunluk);
    {
      for (unsigned i = 0; i < x.uzunluk; i++)
        x.value[i] = rand() << 8 | rand();
    }
    x %= p;
    if (x != 0) {
      large_int::Power(x, pminus1, p, r);
      if (r != 1)
        return false;
    }
  }
  return true;
}
```

Tüm bu işlemlerin tamamlanması ile gerekli RSA sabitleri ve anahtarlar üretilmiş olur. Bu aşamada program Şekil 6.4.'te gösterildiği gibi görülecektir.

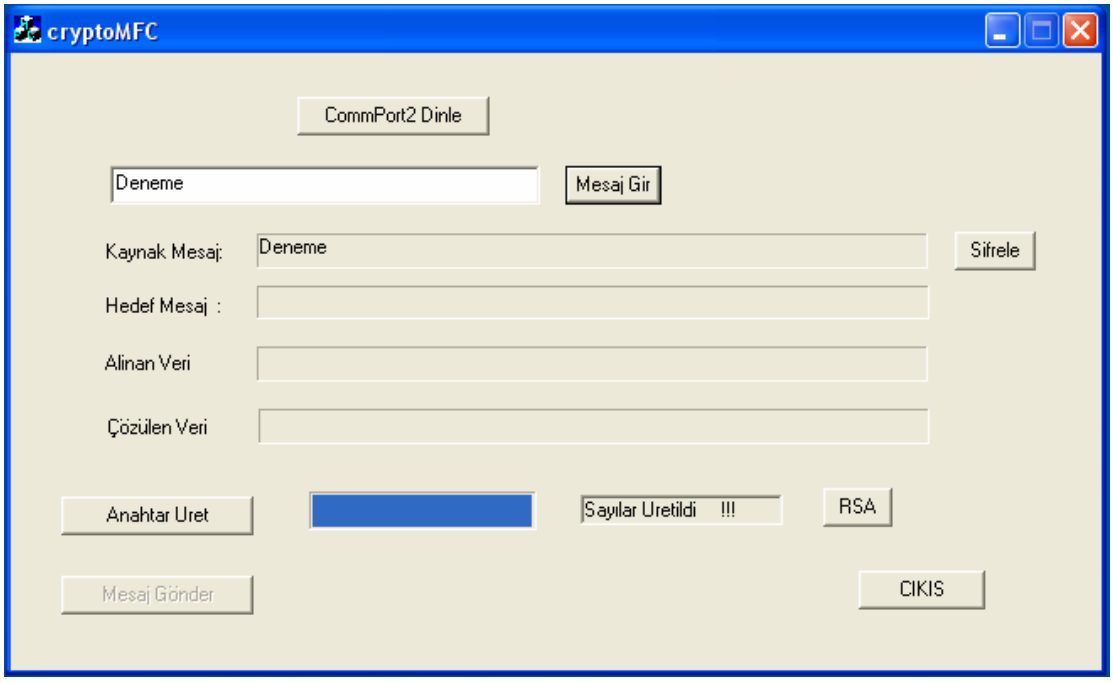


Şekil 6.4. Anahtar üretme işleminin tamamlanmasının ardından programın ekran görüntüsü.

RSA sabitlerinin başarı ile üretilmesinin ardından RSA butonu aktif hale gelecektir. Bu buton kullanılarak Şekil 6.2.'de gösterilen RSA Sabitleri Diyalog kutusu açılarak üretilen sabitler görüntülenebilir.

6.3.3. Gönderilecek mesajın programa girilmesi ve şifrelenmesi

Şifreleme işlemleri için gereken anahtarların oluşturulmasının ardından gönderilmek istenen mesaj veya komut programa girilmelidir. Bunun için Şekil 6.1.'de "2" ile gösterilen düzenleme kutusuna mesaj/komut yazılmalı ve "3" ile gösterilen buton kullanılarak programa girilmelidir. Bu işlemin ardından girilen mesaj, program tarafından kaynak mesaj olarak kabul edilecektir. Programın ekran görüntüsü ise Şekil 6.5.'te gösterildiği gibi olmalıdır.



Şekil 6.5. Gönderilecek mesajın programa girilmesi anında ekran görüntüsü

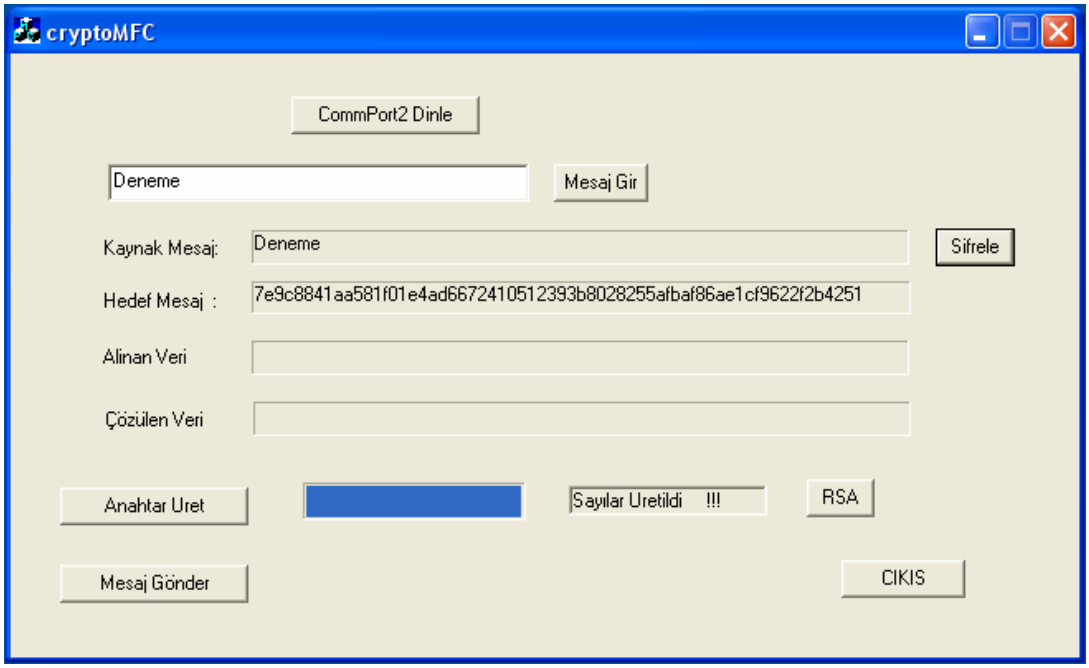
"Mesaj Gir" butonuna basıldığında çağırılan fonksiyon aşağıda gösterilmektedir:

```
void CCryptoMFCDlg::OnBnClickedButtonMesajGir(){
      CString str;
      GetDlgItemText(IDC_EDIT_DATA,str);
      size_t msg_lenght;
      msg_lenght=strlen(str);
            if(msg_lenght>32){
```

```
      MessageBox("Mesaj uzunlugu blok uzunlugunu gecmemelidir! \n
(Blok uzunlugu=32 karakter)","Error!",MB_ICONERROR);
                        return;}
      if(str[0])  {
            char string[30];
            strcpy(string,str);
            //burada editboxtan alınan string büyük sayı sınıfına
            // aktarılmaktadır
            for(int i=0,k=0;i<msg_lenght;i+=2,k++){
            kaynak_msj.value[15-k]=(unsigned short)string[i];
            kaynak_msj.value[15-k]<<=8;
            kaynak_msj.value[15-k]|=(unsigned short)string[i+1];
            }
            if(msg_lenght<32){
                  for(int i=0;i<(32-msg_lenght);i++){
            if((i%2)==0 && i!=1)kaynak_msj.value[(i/2)]&=0x00FF;
      else if((i%2)==1 || i==1)kaynak_msj.value[(i/2)]&=0xFF00;
                  }}
      SetDlgItemText(IDC_STATIC_kaynak,string);
      }
}
```

Bu fonksiyonda, programa girilen mesajın metin formatında alınması işlemleri yapılmaktadır. Aynı zamanda bu fonksiyonda dikkat edilmesi gereken asıl nokta alınan metinin büyük sayı sınıfından türetilmiş kaynak_msj nesnesine bir döngü ile atanmasıdır. Böylelikle mesaj protokole girilmiş olur.

Girilen mesajın şifrelenerek gönderilmeye hazırlanması bu aşamada mümkündür. Dolayısı ile Şekil 6.1.'de 5 ile numaralanan "Şifrele" butonu kullanılarak mesaj şifrelenir. Aşağıdaki Şekil 6.6.'da programın bu adımdaki ekran görüntüsü verilmektedir.



Şekil 6.6. Girilen mesajın şifrelenmesi sonrası programın ekran görüntüsü

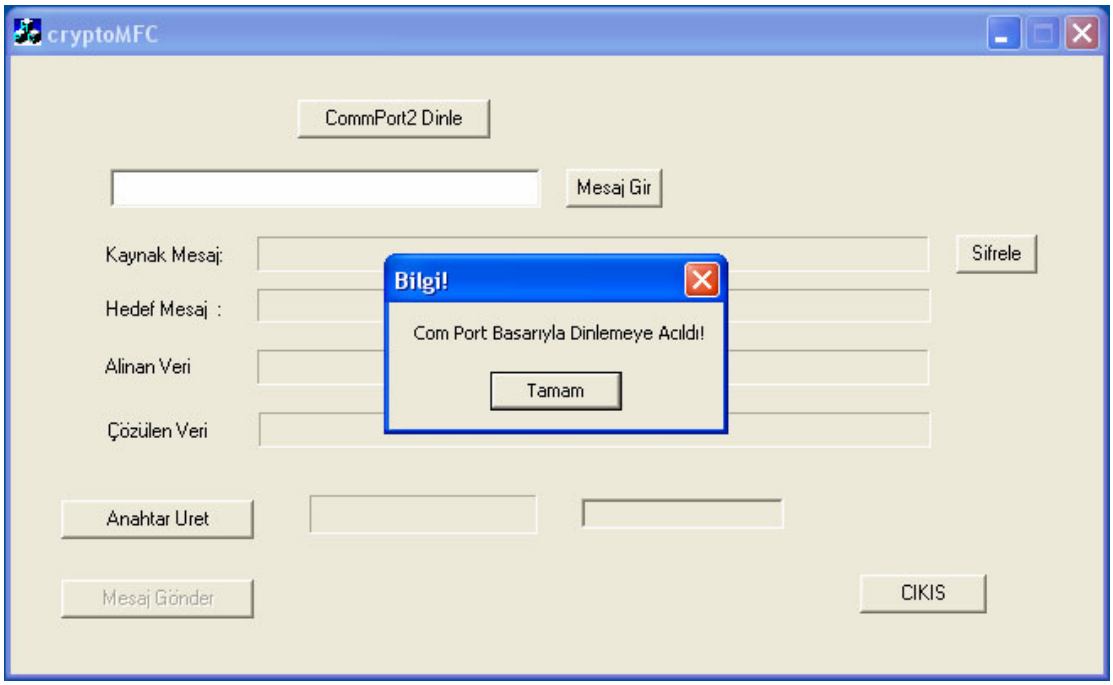
"Şifrele" butonunun çağırdığı fonksiyon aşağıda gösterilmektedir:

```
void CCryptoMFCDlg::OnBnClickedButton2()
{
EncryptDecrypt(result1,kaynak_msj,e1,n1);
      for(unsigned int i=0;i<(2*result1.uzunluk);i++){
            *(str_buffer+i)=*(pbuffer-i);
      }
str_buffer[i+1]=0;
StringDump(str_buffer, strOutput,16);
SetDlgItemText(IDC_STATIC_hedef,strOutput);
MsjGonderButton.EnableWindow(1);
}
```

Bu fonksiyonda dikkat edilmesi gereken nokta EncryptDecrypt() rutinin çağırılmasıdır. Bu rutin bölüm 6.2.2 de verilen ve açıklanan fonksiyonları çağırarak şifreleme ve deşifreleme işlemini yapmaktadır. RSA algoritmasının yapısından hatırlanabileceği gibi şifreleme işlemi için "e" sabiti deşifreleme işlemi içinse "d" sabitinin bu rutine parametre olarak verilmesi gerekmektedir.

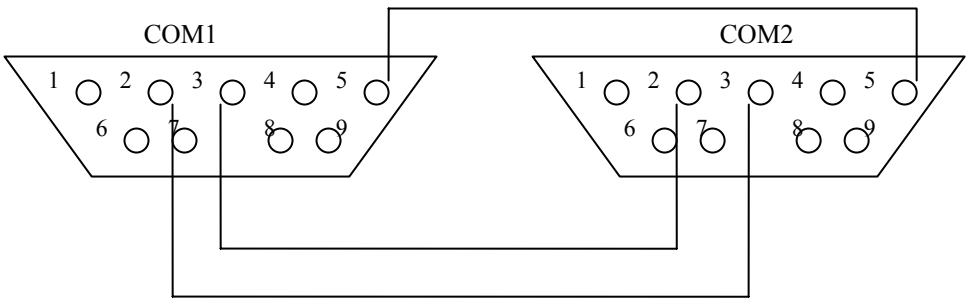
6.4. İletişim Kanallarının Hazırlanması

Daha önce, programın, mesajı göndermek için 1 numaralı iletişim kanalını, mesajı almak içinse 2 numaralı iletişim kanalını kullandığı belirtilmişti. Programın, bir kanaldan gönderdiği şifreli mesajı diğer kanaldan tekrar alarak deşifrelemesi tamamen simülasyon amaçlı olmasından dolayı; 1 numaralı iletişim kanalı programın yüklenmesiyle birlikte hazır hale getirilirken, 2 numaralı iletişim kanalının açılması kullanıcı tarafından yapılacak şekilde dizayn edilmiştir. Dolayısı ile simülasyon işlemine başlamadan önce Şekil 6.1.'de 1 ile numaralanan buton kullanılarak 2 numaralı iletişim kanalı hazırlanmalıdır. Aşağıdaki şekilde bu aşamada programın ekran görüntüsü verilmektedir.



Şekil 6.7. 2 numaralı iletişim kanalı hazırlandığında programın ekran görüntüsü

Simülasyon işleminin doğru bir şekilde yapılabilmesi için yazılımla yapılacak ayarlamaların yanında donanımsal bir bağlantıya da ihtiyaç vardır. Yapılması gereken 1 ve 2 numaralı COMM portların TX- Verici kanalının RX –Alıcı kanalı ile bağlanmasıdır.



Şekil 6.8. Seri iletişim kanallarının simülasyon için hazırlanması

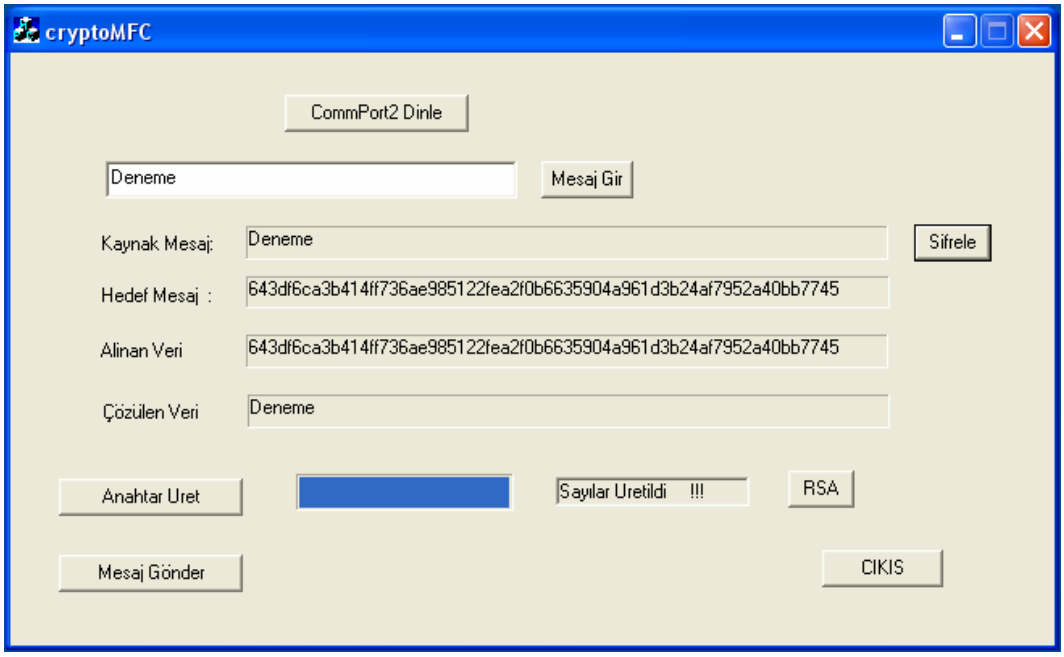
İletişim kanalının dinlemeye açılmasını sağlayan kod aşağıda verilmektedir:

```
void CCryptoMFCDlg::OnButtonComm2()
{
	if(flComPortAcildi==SUCCESS){
MessageBox("Com Port Zaten Dinlemeye Acılmıs! ","Bilgi!",MB_OK);
	return;
	}
	if(InitCommPort("COM2",CBR_9600,8,NOPARITY,ONESTOPBIT)==TRUE){
		hThread =
CreateThread(NULL,0,ReadCommPort,NULL,0,&hThreadId);
		flComPortAcildi=SUCCESS;
MessageBox("Com Port Basarıyla Dinlemeye Acıldı! ","Bilgi!", MB_OK);
	}
	else
MessageBox("Com Port Dinlemeye Acılamadı! ","Error!",MB_ICONERROR);
}
```

Bu fonksiyondan da anlaşılacağı gibi InitCommPort() fonksiyonuna gönderilen parametre değerleri ile (Bu parametre değerleri 1 numaralı iletişim kanalı için de aynıdır.) hazırlanmış ve yeni bir “thread” ile 2 numaralı iletişim kanalına ulaşan verinin program tarafından algılanması sağlanmıştır.

6.5. Şifreli Mesajın Gönderilmesi

Bir önceki bölümde anlatıldığı şekilde hazırlanan iletişim kanallarının üzerinden simülasyon işlemine başlamak artık mümkündür. Bu amaçla Şekil 6.1.’de 13 ile numaralandırılan buton kullanılarak şifrelenen mesajın gönderilmesi ve aşağıdaki şekilde belirtilen ekran görüntüsünün gözlenmesi beklenmelidir:



Şekil 6.9. Simülasyon sonrası şifrelenen verinin alınıp deşifrelenmesi sonrası ekran görüntüsü

2 numaralı İletişim kanalının dinlemeye açılması bir önceki bölümde anlatılmıştı. Alınan verinin deşifrelenmesi ise aşağıdaki algoritmanın içinde yapılmaktadır:

```
LRESULT CCryptoMFCDlg::WindowProc(UINT message, WPARAM wParam,
LPARAM lParam){
	switch(message)
	{
	case WM_CLOSE:
		{
			if(TerminateThread(hThread,0))
				break;
			else
			{
				if(flComPortAcildi==SUCCESS){
	MessageBox("Unable to terminate rx thread attached to comm
port!","Error!",MB_ICONERROR);
				return 0	}
			}
		};
	case WM_RXCHAR:
		{
			char flag=0;
			data[strCount]=(char)wParam;
			SetDlgItemText(IDC_STATIC_data_received,data);
			strCount++;
			if(strCount==63){
			//Deşifreleme işlemi!
```

```
                    EncryptDecrypt(result2,result1,d1,n1);
                    for(unsigned int i=0;i<(2*result2.uzunluk);i++){
                          *(str_buffer+i)=*(fbuffer-i);
                                }
                          str_buffer[i+1]=0;
      SetDlgItemText(IDC_STATIC_data_received_decrypted,str_buffer);
                    }
                };
      }
      return CDialog::WindowProc(message, wParam, lParam);
}
```

Bu algoritmada dikkat edilmesi gereken nokta; "WM_RXCHAR" olayının gerçekleşmesi ve 64 karakterin seri kanaldan alınmasının ardından çağırılan EncryptDecrypt() fonksiyonudur. Daha önce de anlatılan bu fonksiyon hem şifreleme hem deşifreleme işlemlerini gerçekleştirmekle görevlidir. Burada deşifreleme işleminin yapılacağını bu fonksiyona gönderilen "d" sabiti belirlemektedir.

7. SONUÇ VE ÖNERİLER

Bu çalışmada veri iletişiminde karşılaşılabilecek temel problemler ele alınmıştır. Bu problemler; veri iletişiminde güvenlik, iletişime giren tarafların kimliklerinin doğruluğu ve iletişim hattında kasıtlı olarak oluşturulabilecek veya gürültüden kaynaklanabilecek bozulmalar ile belirtilebilecek üç ana başlıkta incelenerek söz konusu hata ve eksikliklerin giderildiği bir "protokol modeli" sunulmaya çalışılmıştır.

Sunulan bu protokolün güvenlik ve kimlik doğrulama ihtiyaçları açık anahtarlı şifreleme algoritmalarının en bilinenlerinden olan RSA algoritması ile çözülmesi öngörülmüştür. Veride oluşabilecek hataların denetlenmesi içinse basit bir kontrol-toplamı algoritması kullanmak yerine "Döngüsel Artıklık Denetimi" olarak ta bilinen CRC algoritması uygun görülmüştür.

Bu üç ana hedef ile geliştirilen model, herhangi bir platforma uygulanabilmesinin yanında, Windows işletim sistemine ve iki adet seri iletişim kanalına sahip bilgisayarlarda çalıştırılabilecek bir örnek program bu modele paralel olarak geliştirilmiş ve ek olarak sunulmuştur. Bu programda 256 bitlik anahtarlar kullanarak şifreleme/deşifreleme işlemleri, iletişim kanalları üzerinden simüle edilmektedir.

Algoritmalar üzerinde yapılacak optimizasyon ile programın çalışması hızlandırılabileceği gibi daha uzun anahtarlar kullanılarak güvenliğin arttırılması mümkündür. Ayrıca ortaya konulan bu modelin gerçek bir kontrol sistemine uygulanarak sonuçların değerlendirilmesi ve protokolün bu sonuçlara göre yeniden şekillendirilmesi yeni bir çalışmanın konusu olabilir. Değerlendirme platformunun RF tabanlı seçilmesi hata denetleme algoritmasının değerlendirilmesinde etkili olacaktır.

KAYNAKLAR

1. Diffie, W. and Hellman, M.E., "New directions in cryptography" , **IEEE Trans. Inform.Theory,** 22 (6): 644-654 (1976).

2. Nguyen, H., Aziz, S.M., Ryan, T., "A low bandwidth communication protocol for remote control applications" , **IEEE Region 10 Conference,** C: 37-40 (2004).

3. Wolisz, A., Schieferdecker, I., Walch, M., "An integrated approach to the design of communication protocols" , **Distributed Computing Systems,** 19: 411-418 (1993).

4. Martin, G., "Mathematical games" , **Scientific America,** 237: 18-28 (1977).

5. Rivest, R.L., Shamir, A., Adleman, L.A., "A method for obtaining digital signatures and public-key cryptosystem" , **Communications of the ACM,** 21 (2): 120-126 (1978).

6. Solovay, A. and Strassen, V., "A fast monte-carlo test for primality" , **SIAM Journal on Computing,** 6 (1): 84-86 (1977).

7. Boneh, M., Franklin, A., "Efficient generation of shared RSA keys" , **In Proceedings Crypto,** 97: 425-439 (1997).

8. Manindra, A. and Somenath, B., "Primality and identity testing via chinese remaindering" , **In Proceedings of 40th FOCS,** 1: 202-209 (1999).

9. Joye, P., Paillier, K. and Vaudenay, S., "Efficient generation of prime numbers. in proceedings of cryptographic hardware and embedded systems" , **Lecture Notes in Computer Science**, 1965 (1): 340-354 (2000).

10. Rabin, O., "Probabilistic algorithm for testing primality" , **Journal of Number Theory**, 12: 128-138 (1980).

11. Riemann, M., "Hypothesis and tests for primality" , **J. Comput. System Sci.**, 13: 300-317 (1976).

12. Damgard, P., Landrock, C. and Pomerance, C., "Average case error estimates for the strong probable prime test" , **Math. Computing**, 61 (203): 177-194 (1993).

13. Weinberger, P. J., Rothschild, L. P., "Factoring polynomials over algebraic number fields" , **ACM Trans. Math. Software,** 2 (2): 335-350 (1976).

14. Berrizbeitia, P., Berry, T. G., "Cubic reciprocity and generalized Lucas-Lehmer test for primality of A^3 (n±1)" , **Proceedings of AMS,** 127: 1923-1925 (1999).

15. Kocher, P., “Timing attacks on implementation of Diffie–Hellman RSA, DSS and other systems” , **In Advances in Cryptology,** 1 (7): 43-50 (1996).

16. Popek, G. J. and Kline, C. S., “Encryption and secure computer networks” , **ACM Computing Surveys**, 11 (4): 335-356 (1979).

17. Kohnfelder, L. A., “A method for certification” , **Massachusetts Institute of Technology**, Cambridge, 274-331 (1978).

18. Denning, D. E., “Cryptography and data security” , **Communications of the ACM**, 33: 155-159 (1983).

19. Merkle, R. C., “Secrecy, authentication and public key systems” , **UMI Research Pres**, Ann Arbor, 99-278 (1979).

20. Stinson, D. R., “Cryptography theory and practice” , **CRC Pres. Trs.**, 17(2): 417-422 (1995).

21. Needham, R. and Shroeder, M., “Using encryption for authentication in large networks of computer” , **Communications of the ACM**, 21: 993-999 (1978).

22. Tanenbaum, A. S., “Computer networks” , **Prentice Hall**, New York, 128-132 (1981).

E K L E R

EK 1 – Programda kullanılan başlık dosyaları

Fonksiyon ve Sınıf Tanımlamaları

largenum.h:

```
/*****************************************************************
********
*Header File: largenum.H
*Tanımlayan: Ahmet Çağrı Bölücek
*Tarih: 17.10.2004
*Son Değiştirme tarihi: 23.03.2005
*Açıklama: Bu başlık dosyası büyük sayı sınıfının tanımını içerir. Diğer tüm
işlemler bu *türden sayılar kullanılarak yapılacağından bu başlık dosyası nesne
oluşturulmadan önce koda *dahil edilmelidir.
*****************************************************************
*******/
 #ifndef H__large_int          //Başlık dosyasının birden fazla defa eklenmemesini ve
 #define H__large_int          // böylece link sırasında sorun oluşmasını engelleyen
        kontrol.

extern void numeric_overflow(void);      //İşlemde taşma olduğunda çağırılacak
                                         rutinin //prototipi
class large_int                          //Sınıf Tanımı
{
  private:
    unsigned short remainder(unsigned short);
    void shift(unsigned);
  public:
    unsigned short *value;
    bool IsZero(void) const;
    int Compare(const large_int &) const;
    int Compare(unsigned short) const;
    unsigned uzunluk;
    large_int(unsigned);                 //Aşırı Yüklenmiş Kurucu
Fonksiyonlar
    large_int(const large_int &);
    ~large_int();                        //Yıkıcı Fonksiyon
    void operator = (const large_int &); //Aşırı yüklenmiş operatörlerin
prototip
    void operator = (unsigned short);    //tanımlamaları.
    void operator += (const large_int &);
    void operator += (unsigned short);
    void operator -= (const large_int &);
    void operator -= (unsigned short);
    void operator *= (const large_int &);
    void operator *= (unsigned short);
    void operator /= (const large_int &);
```

```
   void operator /= (unsigned short);
   void operator %= (const large_int &);
   void operator %= (unsigned short);
   static void Divide(const large_int &, const large_int &, large_int &, large_int &);
   char *edit(char *) const;
   bool scan(const char *&);
   void dump() const;
   bool large_int::operator == (const large_int &n) const {return Compare(n) == 0;}
   bool large_int::operator != (const large_int &n) const {return Compare(n) != 0;}
   bool large_int::operator >  (const large_int &n) const {return Compare(n) >  0;}
   bool large_int::operator >= (const large_int &n) const {return Compare(n) >= 0;}
   bool large_int::operator <  (const large_int &n) const {return Compare(n) <  0;}
   bool large_int::operator <= (const large_int &n) const {return Compare(n) <= 0;}
   bool large_int::operator == (unsigned short n) const {return Compare(n) == 0;}
   bool large_int::operator != (unsigned short n) const {return Compare(n) != 0;}
   bool large_int::operator >  (unsigned short n) const {return Compare(n) >  0;}
   bool large_int::operator >= (unsigned short n) const {return Compare(n) >= 0;}
   bool large_int::operator <  (unsigned short n) const {return Compare(n) <  0;}
   bool large_int::operator <= (unsigned short n) const {return Compare(n) <= 0;}
   static void Power(const large_int &, const large_int &, const large_int &,large_int
&);
};

#endif
```

rsa.h:

```
/*****************************************************************
********
*Header File: rsa.H
*Tanımlayan: Ahmet Çağrı Bölücek
*Tarih: 17.10.2004
*Son Değiştirme tarihi: 23.03.2005
*Açıklama:Bu başlık dosyası program içinde kullanılacak Anahtar Üretme işlemini
yapan *AnahtarUret(...) fonksiyonunun ve Şifreleme/Deşifreleme işlemlerini yapan
EncryptDecrypt *(...) fonksiyonunun prototiplerini içerir.
*****************************************************************
*******/
#ifndef H__RSA              //Başlık dosyasının birden fazla defa eklenmemesini ve
#define H__RSA              // böylece link sırasında sorun oluşmasını engelleyen
kontrol.
#include "largenum.h"
void AnahtarUret(large_int &d, large_int &e, large_int &n);
void Random(large_int &x);
inline void EncryptDecrypt(large_int &result, const large_int &source,
  const large_int &e, const large_int &n){
  large_int::Power(source, e, n, result);
}
```

```
#endif
```

euclid.h:

```
/*****************************************************************
********
*Header File: euclid.H
*Tanımlayan: Ahmet Çağrı Bölücek
*Tarih: 17.10.2004
*Son Değiştirme tarihi: 23.03.2005
*Açıklama: Bu başlık dosyası RSA algoritmasında önemli bir yeri bulunan,
aralarında asal *sayıların bulunması sırasında kullanılan euclid algortimasının
uygulandığı fonksiyon olan EuclideanAlgorithm(...) fonksiyonunun prototipini içerir.
*****************************************************************
*******/

#ifndef H__EUCLID          //Başlık dosyasının birden fazla defa eklenmemesini ve
#define H__EUCLID          // böylece link sırasında sorun oluşmasını engelleyen
kontrol.

#include "largenum.h"

typedef large_int UITYPE;  //nesne tanımlamaları sırasında UITYPE değeri de
kullanılabilir.

void EuclideanAlgorithm(const UITYPE &x, const UITYPE &y, UITYPE &a,
  UITYPE &b, UITYPE &g);
void GreatestCommonDivisor(const UITYPE &x, const UITYPE &y, UITYPE &g);

#endif
```

EK 2 – Programın Kaynak Kodları

Fonksiyonlar

```
/*****************************************************************
*Program İsmi: largenum.cpp
*Programlayan: Ahmet Çağrı Bölücek
*Tarih: 10.12.2004
*Son Değiştirme tarihi: 23.03.2005
*Açıklama: Bu dosyada largenum.h başlık dosyasında tanımları verilen
fonksiyonların kodları *bulunmaktadır.
*****************************************************************/

#include "stdafx.h"
#include "cryptoMFC.h"
#include "cryptoMFCDlg.h"
#include <stdio.h>
#include "largenum.h"

large_int::large_int(unsigned len){          //32 bitlik uzunluk değeri alan kurucu
fonksiyon
  uzunluk = len;
  value = new unsigned short[len]; }

large_int::large_int(const large_int &n) {   //büyük sayı türünden uzunluk değeri
alan kurucu
  uzunluk = n.uzunluk;                       //fonksiyon
  value = new unsigned short[uzunluk]; unsigned i;
  for (i = 0; i < uzunluk; i++)
    value[i] = n.value[i]; }

large_int::~large_int(){                     //Hafızada ayırılan alanı nesnenin
kullanımının  delete [] value;          // bitmesiyle boşaltan yıkıcı fonksiyon
}

void large_int::operator = (const large_int &n){    //= operatörünü aşırı yükleyen
fonksiyon
  unsigned i;
  for (i = 0; i < uzunluk && i < n.uzunluk; i++)
            value[i] = n.value[i];
  for (; i < uzunluk; i++)
            value[i] = 0;
  for (; i < n.uzunluk; i++)  {
    if (n.value[i] != 0)                     //Taşma olup olmadığı
denetleniyor.
          numeric_overflow();          //varsa numeric_overflow() fonksiyonuna
  }}                                         //dallan.
```

```
void large_int::operator = (unsigned short n){     //= operatörü 16 bitlik değer alacak
  value[0] = n;  unsigned i;                              //şekilde aşırı yükleniyor.
  for (i = 1; i < uzunluk; i++)
              value[i] = 0;}
void large_int::operator += (const large_int &n){    //+= operatörü (a+=b → a=a+b)
  unsigned i;                                        //büyük sayı girdisi alacak şekilde
  unsigned long carry = 0;                                // aşırı yükleniyor.
  for (i = 0; i < uzunluk; i++)  {
    unsigned long sum = value[i] + (i < n.uzunluk ? n.value[i] : 0) + carry;
    value[i] = sum;
    carry = sum >> 16;
  }
  if (carry != 0)                                    //Taşma olup olmadığı denetleniyor.
    numeric_overflow();                  //varsa numeric_overflow() fonksiyonuna
  for (; i < n.uzunluk; i++)                              //dallan
  {
    if (n.value[i] != 0)
      numeric_overflow();  }}

void large_int::operator += (unsigned short n) {      // += operatörü 16 bitlik değer
alacak  value[0] += n;                                    // şekilde aşırı yükleniyor
  if (value[0] < n)
  {
    unsigned i;
    for (i = 1; i < uzunluk; i++)
    {
      if (++value[i] != 0)
        return;
    }
    numeric_overflow();  }}

void large_int::operator -= (const large_int &n){              //-= operatörü (a-=b → a
= a - b)
  unsigned i;                                                  //büyük sayı girdisi alacak
şekilde
  unsigned long borrow = 0;                                    // aşırı yükleniyor.
  for (i = 0; i < uzunluk; i++)  {
    unsigned long subtrahend = (i < n.uzunluk ? n.value[i] : 0) + borrow;
    borrow = (unsigned long)(value[i]) < subtrahend;
    value[i] -= subtrahend;  }
  if (borrow != 0)
    numeric_overflow();
  for (; i < n.uzunluk; i++) {
    if (n.value[i] != 0)
      numeric_overflow();  }}
```

```
void large_int::operator -= (unsigned short n){       // -= operatörü 16 bitlik değer
alacak
 if (value[0] >= n)                                   // şekilde aşırı yükleniyor
  value[0] -= n;
 else  {
  value[0] -= n;
  for (unsigned i = 1; i < uzunluk; i++){
   if (--value[i] != 0xFFFF)
     return;
  }
  numeric_overflow();  }}

void large_int::operator *= (const large_int &n){            //*= operatörü (a*=b → a=a*b)
 unsigned i;                                                 //büyük sayı girdisi alacak
şekilde
 unsigned short *multiplier = new unsigned short[uzunluk];        // aşırı yükleniyor
 for (i = 0; i < uzunluk; i++){
  multiplier[i] = value[i];
  value[i] = 0;  }
 for (i = 0; i < uzunluk; i++){
  unsigned j;
  for (j = 0; j < n.uzunluk; j++){
   unsigned long product = multiplier[i] * n.value[j];
   unsigned k = i + j;
   while (product != 0){
    if (k >= uzunluk)                                      //Taşma Durumu Denetleniyor.
      numeric_overflow();
    product += value[k];
    value[k] = product;
    product >>= 16;
    k++;    }   }  }
 delete [] multiplier;            //Çarpan değerini tutmak için ayrılan alan boşaltılıyor.
 }

void large_int::operator *= (unsigned short n){       // -= operatörü 16 bitlik değer
alacak
 unsigned i;                                                 // şekilde aşırı yükleniyor
 unsigned long product = 0;
 for (i = 0; i < uzunluk; i++)  {
  product += n * value[i];
  value[i] = product;
  product >>= 16;  }
 if (product != 0)
  numeric_overflow();}
```

```
unsigned short large_int::remainder(unsigned short n){      //Bölme işleminden kalanı
bulan
  unsigned i = uzunluk;                                     // fonksiyon. Aynı zamanda % ve
/=
  unsigned rem = 0;                                         //operatörünün aşırı yüklenmesi
sırasında
  while (i-- != 0) {                                        //da kullanılacak.
    unsigned long dividend = (unsigned long) rem << 16 |  (unsigned long) value[i];
    value[i] = dividend / n;
    rem = dividend % n;  }
  return rem;}

void large_int::operator /= (unsigned short n){
  (void) remainder(n);}

void large_int::operator %= (unsigned short n){
  *this = remainder(n);}

int large_int::Compare(const large_int &n) const{           //İki büyük sayının
uzunluklarının
  unsigned i;                              //karşılaştırmasını yaparak girdi olarak verilen
sayı,
  if (uzunluk > n.uzunluk)  {              // sözkonusu nesneden daha büyükse -1, nesne
                                           sayıdan //daha büyükse 1 değerini döndürür.
    for (i = uzunluk-1; i >= n.uzunluk; i--){
      if (value[i] != 0)
        return 1;
                  }
                                      }
  else if (n.uzunluk > uzunluk)  {
    for (i = n.uzunluk-1; i >= uzunluk; i--)    {
      if (n.value[i] != 0)
        return -1; }}
  else
    i = uzunluk-1;
  while (true)  {
    if (value[i] > n.value[i])
      return 1;
    if (value[i] < n.value[i])
      return -1;
    if (i == 0)
      return 0;
    i--; }}
int large_int::Compare(unsigned short n) const{     // Bu fonksiyon bir önceki
karşılaştırma
  unsigned i;                                       //fonksiyonunun 16 bitlik bir sayı
için
```

```
 for (i = uzunluk-1; i >= 1; i--)                    //aşırı yüklenmiş halidir.
 {
  if (value[i] != 0)
    return 1;
 }
 return value[0] > n ? 1 : value[0] < n ? -1 : 0;}

bool large_int::IsZero(void) const          //Sınıfa ait nesnenin değerinin 0 (sıfır)
olup
{                                           //olmadığını denetleyerek 0 olduğu
durumda
 unsigned i;                                //"true" olmadığı durumda "false" değeri
 for (i = 0; i < uzunluk; i++) {            // döndürür. Bu fonksiyon özellikle asal
sayı
        if (value[i] != 0)                  // oluştururken kullanılacaktır.
       return false;
 }
 return true;}
char *large_int::edit(char *s) const{       //Bu fonksiyon sözkonusu büyük sayı
nesnesinin
 large_int n(*this);                        //ascii karakterlerle ifade edilebilmesini
sağlar
 unsigned i = 0;                            //Sayı bir string halinde ifade edilerek
ekranda
 do                                         //Gösterimi kolaylaşır.
  s[i++] = n.remainder(10) + '0';
 while (!n.IsZero());
 s[i] = 0;
 unsigned j;
 for (j = 0; --i > j; j++)
 {
  char c = s[i];
  s[i] = s[j];
  s[j] = c;
 }
 return s; }

bool large_int::scan(const char *&s) {      //Bu fonksiyon bir karakter dizisi
içerisinde 8
 const char *t = s;                         //bitlik değer aramaktadır. Bu
fonksiyonun bir
 bool found = false;                        //büyük sayı için kullanılmasından evvel
edit
 while (*t == ' ' || *t == '\t')            //fonksiyonu kullanılarak sayının ascii
olarak
  t++;                                      //ifade edilmesi gerekir.
 *this = 0;
```

```
 while ('0' <= *t && *t <= '9')
 {
  found = true;
  *this *= 10;
  *this += (unsigned short) (*t++ - '0');   }
 s = t;
 return found; }

void large_int::shift(unsigned bit){            //Bu fonksiyon ait olduğu büyük sayı
nesnesi
 for (unsigned i = 0; i < uzunluk; i++)  {      //için bit kaydırma işlemi yapar.
Böylelikle 2'ye
   unsigned long x = value[i] << 1 | bit;       //bölme veya 2' ile çarpma işlemleri çok
daha
   value[i] = x;                                //hızlı yapılabilir.
   bit = x >> 16;   }
 if (bit != 0)                                  //Taşma durumu kontrol edilir.
   numeric_overflow();
}

void large_int::Divide(const large_int &dividend, const large_int &divisor, large_int
&quotient,  large_int &remainder){
 if (divisor.IsZero())                  //Bu fonksiyonda bölme işlemi büyük sayının
   numeric_overflow();                  // 16 bitlik değerler halinde ele alınarak temel
 remainder = 0;                         //matematik kurallarının uygulanmasıyla
yapılmaktadır
 quotient = 0;
 unsigned i = dividend.uzunluk;
 while (i-- != 0)  {
   unsigned bit = 16;
   while (bit-- != 0)     {
     remainder.shift(dividend.value[i] >> bit & 1);
     if (divisor <= remainder)      {
       quotient.shift(1);
       remainder -= divisor;
          }
     else
       quotient.shift(false);
          }
                          }
}

void large_int::operator /= (const large_int &n) {    //Yukarıda 16 bitlik bir değer için
verilen
 large_int remainder(n.uzunluk);                      // "/=" operatörü burada büyük
sayı
```

```
 large_int dividend(*this);                    //türünden bir bölen için aşırı
yüklenmiştir
 Divide(dividend, n, *this, remainder);
}

void large_int::operator %= (const large_int &n) {
 large_int quotient(uzunluk);                  //Özellikle mod alma işlemlerinde
kullanılacak
 large_int dividend(*this);                    //% operatörü büyük sayı türünden bölen
için
 Divide(dividend, n, quotient, *this);         //aşırı yüklenmiştir.
}

void large_int::Power(const large_int &base, const large_int &exponent,
 const large_int &modul, large_int &sonuc){
 large_int r(2*base.uzunluk+1);
 r = 1;                                        //Bu fonksiyon büyüksayı türünden verilen
taban ve
 bool one = true;                              //üs degerlerine göre üs alma işlemi yapmakta
yine
 unsigned i = exponent.uzunluk;                // verilen modüle indigeyerek sonuç değerine
atama
 while (i-- != 0)  {                           //yapmaktadır. Modül alma işlemi sayının
büyümesini    unsigned bit = 1 << 15;          // engellemek amacıyla ara
değerler için uygulanmıştır.

   do    {
    if (!one)      {
      large_int n(r);
      r *= n;
      r %= modul;      }
    if (exponent.value[i] & bit) {
      r *= base;
      r %= modul;
      one = false;      }
    bit >>= 1;
   } while (bit != 0);
 }
sonuc = r;}

void large_int::dump() const{
 unsigned i;
 for (i = 0; i < uzunluk; i++)
   printf(" %x", value[i]);
 putchar('\n');
}
```

ÖZGEÇMİŞ

1979 yılında Sivas'ta doğdu. 2002 yılında Gazi Üniversitesi Elektrik-Elektronik mühendisliği bölümünden mezun oldu. Aynı sene tıbbi cihazlar üreten özel bir şirketin Ar-Ge bölümünde proje geliştirici olarak göreve başladı. Halen bu pozisyondaki görevini sürdürmektedir.