KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

VERİ GÜVENLİĞİ

BİTİRME PROJESİ

Onur BUDAK
Batuhan EKİCİ
Hakan AKSOY

2018-2019 BAHAR DÖNEMİ

KARADENİZ TEKNİK ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

VERİ GÜVENLİĞİ

BİTİRME PROJESİ

Onur BUDAK
Batuhan EKİCİ
Hakan AKSOY

2018-2019 BAHAR DÖNEMİ

IEEE Etik Kuralları
IEEE Code of Ethics

Mesleğime karşı şahsi sorumluluğumu kabul ederek, hizmet ettiğim toplumlara ve üyelerine en yüksek etik ve mesleki davranışta bulunmaya söz verdiğimi ve aşağıdaki etik kurallarını kabul ettiğimi ifade ederim:

1. Kamu güvenliği, sağlığı ve refahı ile uyumlu kararlar vermenin sorumluluğunu kabul etmek ve kamu veya çevreyi tehdit edebilecek faktörleri derhal açıklamak;
2. Mümkün olabilecek çıkar çatışması, ister gerçekten var olması isterse sadece algı olması, durumlarından kaçınmak. Çıkar çatışması olması durumunda, etkilenen taraflara durumu bildirmek;
3. Mevcut verilere dayalı tahminlerde ve fikir beyan etmelerde gerçekçi ve dürüst olmak;
4. Her türlü rüşveti reddetmek;
5. Mütenasip uygulamalarını ve muhtemel sonuçlarını gözeterek teknoloji anlayışını geliştirmek;
6. Teknik yeterliliklerimizi sürdürmek ve geliştirmek, yeterli eğitim veya tecrübe olması veya işin zorluk sınırları ifade edilmesi durumunda ancak başkaları için teknolojik sorumlulukları üstlenmek;
7. Teknik bir çalışma hakkında yansız bir eleştiri için uğraşmak, eleştiriyi kabul etmek ve eleştiriyi yapmak; hatları kabul etmek ve düzeltmek; diğer katkı sunanların emeklerini ifade etmek;
8. Bütün kişilere adilane davranmak; ırk, din, cinsiyet, yaş, milliyet, cinsi tercih, cinsiyet kimliği, veya cinsiyet ifadesi üzerinden ayırımcılık yapma durumuna girişmemek;
9. Yanlış veya kötü amaçlı eylemler sonucu kimsenin yaralanması, mülklerinin zarar görmesi, itibarlarının veya istihdamlarının zedelenmesi durumlarının oluşmasından kaçınmak;
10. Meslektaşlara ve yardımcı personele mesleki gelişimlerinde yardımcı olmak ve onları desteklemek.

IEEE Yönetim Kurulu tarafından Ağustos 1990'da onaylanmıştır.

ÖNSÖZ

Günümüzdeki internet ortamında sıklıkla ihtiyaç duyulan SSL (Secure Secret Layer) içinde bulunan "Simetrik Şifreleme" alanında yaptığımız algoritma tasarımını tamamlayıp, bu yapının koda implementasyonunu gerçekleştirmeye çalıştık.

Öncelikle tez konusunu seçerken isteklerimizi göz önünde bulundurup bize yardımcı olan Doç. Dr. Güzin ULUTAŞ ve Prof. Dr. Mustafa ULUTAŞ hocamıza teşekkür ederiz.

Proje tasarımını gerçekleştirirken yardımlarını esirgemeyen TÜBİTAK TÜSSİDE çalışanları Ahmet Çakmak ve Öznur Kalkar'a, bölümümüz Arş. Gör. Ercüment Öztürk hocamıza, bu zorlu tez sürecinde bizden desteğini bir an için esirgemeyen değerli sınıf ve ev arkadaşımız Mehmet Alperen Çubuk'a ve tüm eğitim hayatımız boyunca bizden maddi ve manevi desteklerini esirgemeyen her zaman yanımızda olan sevgili ailelerimize teşekkürlerimizi bir borç biliriz.

Onur BUDAK
Batuhan EKİCİ
Hakan AKSOY
Trabzon 2019

İÇİNDEKİLER

ÖZET

Proje kapsamında “Simetrik Şifreleme” algoritmalarının tasarımını gerçeklemeye çalışırken, birçok parametreye dikkat ederek algoritmalarda iyileştirmeler yaptık. Tasarımların görsel halini “flowgorithm” uygulamasını kullanarak gerçekleştirdik.

Projenin ilk kısmında şifreleme algoritmalarının implementasyonu için Eclipse ve Visual Studio IDE’lerini kullanarak gerekli testleri yapmıştık. Algoritmalarda üretilen anahtarın uzunluğu ile bağlantılı olarak BruteForce süre değerlerini inceledik.

Algoritmaların uygulamalara implementasyonundan sonra, bir istemci-sunucu uygulamasından şifreleme algoritmalarını test etmeye çalıştık. Bu süreçte ilk olarak açık metni şifreleyip gönderimi sağladık. Gönderilen şifreli metni “Wireshark” üzerinden test edip, gelen şifreli metne yaptığımız deşifreleme işlemiyle açık metni elde ettik.

Projenin ilerleyen kısmı için SSL protokolünün özelliklerini barındıran bir yapının implementasyonu için protokol içinde bulunan alt katmanları detaylı bir şekilde inceledik. Bu süreçte hocamızın tavsiyesiyle “SSL & TLS Essentials - Securing the Web” kitabı üzerinden Handshake-Alert-Record protokolleriyle ilgili araştırmalarımızı tamamladık.

Daha sonra 2 farklı platformda yazdığımız şifreleme algoritmalarını ortak bir programlama dili üzerinde toplayarak, algoritmaların kütüphaneleştirme işlemi üzerinde çalışmalar yaptık.

Tasarlanılan kütüphane ile birlikte şifrelemelerin doğruluğunu kontrol ettik. Test işleminden sonra uygulamayı protokolleştirme için soket programlama yapısı hakkında araştırmaları yaptık. İlk önce anahtarın her iki taraf üzerinde bilindiği kabul edilerek şifreleme ve deşifreleme işlemlerini sunucu-istemci arasında gerçekleştirdik.

Son olarak şifreleme algoritmalarında kullanılan anahtar paylaşımını sağlamak ve hash algoritması için gerekli kodlama işlemlerini uygulamaya dahil ettikten sonra iletişimi güvenli hale getirdik.

1. GENEL BİLGİLER

1.1. Giriş

Günümüzde veri güvenliği konusundaki yapılan çalışmaların gün geçtikçe daha çok arttığı görülmektedir. Daha sonra tasarım konusunu belirlemek için, bu konuda yapılan çalışmaları araştırmaya başlamıştık.

Bu süreçte bölümümüzde verilen "Şifreleme" dersine giderek konular hakkında bilgi edinmeye başladık. Ders konuları ilk önce Simetrik Şifreleme konusunda olması, bizi bu konuda tasarım konusu belirlemeye etti. Şifreleme konusunda kötü niyetli insanlara karşı savunulması gereken anahtarın iyi belirlenmesi gerekir. Kapalı metnin ve anahtarın frekans analizine karşı dayanıklılığı önemlidir.

Bu konuda yapılabilecek şifreleme yöntemleri; blok şifreleme yapısı (Block cipher – n'ye n) ve Kelime şifreleme yapısı (Stream cipher – 1'e 1) başlıkları altında düşünülebilir. Blok şifrelemede anahtarın aynı olması gerektiği unutulmamalıdır. Bu yapıda giriş ile çıkış arasındaki ilişki kopmaktadır. Şifrelemede kullanılan döngüler bu özelliğe katkı sağlar.

Simetrik algoritma yapısı olarak DES (Data Encryption Standard) ve AES (Advanced Encryption Standard) örnek verilebilir. DES algoritması NIST (National Institute of Standards and Technology) tarafından yayınlanmış ve FIPS (Federal Information Processing Standards) standartıdır.

Şifreleme ve deşifreleme arasındaki fark, anahtarların veriliş sırasıdır. AES yapısının kullanımı DES'ten farklı olarak tablo tutma işlemi yapmamaktadır. Bunun yerine GF (Galois Field) yapısını kullanmaktadır. Bu şifreleme lineerlik SubWord ile bozulabilirken, anahtarın döngü durumu buna karşılık vermektedir. Bu sebepten dolayı zayıf anahtar görülememektedir. Yani bir bitlik değişimin etkisi oldukça fazladır.

1.2. SSL' e Giriş

Netscape tarafından 1994 yılında SSL piyasaya sürüldü ve bir sonraki yıl IETF tarafından standart olarak kabul edildi. Aslında standardın asıl ismi TLS olmasına rağmen genellikle SSL kullanımı tercih edilmektedir. İlk zamanlar sadece HTTP trafiğini şifreleme amaçlı geliştirilmiş olsa da günümüzde TCP tabanlı tüm servisleri şifreleme amaçlı kullanılabiliyor.

1.2.1. SSL Nedir?

Açılımı "**Secure Sockets Layer**" olan SSL Türkçe anlamı ise "**Güvenli Giriş Katmanı**" demektir. SSL, sunucu ile client arasındaki iletişimde verilerin şifrelenmesi işlemidir. En yaygın ve bilinen kullanımı ise web sitesindeki veri alışverişi sırasında Server ile internet tarayıcısı("browser") arasındaki iletişimi şifrelemesidir. Güvenli veri iletişimi için birçok web sitesi SSL teknolojisini kullanmaktadır. SSL bağlantısı üzerinden paylaşılan bilgiler sadece uygun anahtar değeri ile okunabilir. Dışarıdan yapılan müdahaleler ile bu bilgilere ulaşılsa bile, bu bilgiler şifreli olduklarından tamamen okunamaz haldedirler.

SSL kullanmayan sistemlerde 2 temel güvenlik problemi söz konusudur. Bunlar;

- Doğru sunucuya bağlanıp bağlanamadığımızı bilememek.
- İnternet üzerinden paylaşacağımız verilerin 3. kişiler tarafından ele geçirilip geçirilmediğini bilememek.

SSL in varlığını iki şekilde anlaşılabilir. Birincisi **http://'** nin **https://** olmasından İkincisi ise, url kısmında çıkan kilit simgesinden anlarız. SSL bağlantısı olduğunda tarayıcının url kısmında bir kilit simgesi çıkar. SSL bağlantısı olmadığı zaman ise bu kilit simgesi görünmez.

(SSL kullanmayan sistem)

(SSL kullanan sistem)

HTTPS, HTTP'nin iletim (Transport) katmanının güvenli hale getirilerek gizliliğin sağlanması ve dijital sertifikalarla istemcinin, sunucunun kimliğini doğrulaması ile çalışır.

1.2.2. SSL Sertifikaları

SSL, web sunucusunu tanımak için, dijital olarak imzalanan sertifikalar kullanır. Sertifika, aslında, o organizasyon hakkında bazı bilgiler içeren bir veri dosyasıdır. Aynı zamanda da, kuruluşun açık-kapalı anahtar çiftinin "açık" anahtarı ("public key") da sertifika içinde yer alır. Sertifikalar, "güvenilir" sertifika kuruluşları tarafından dağıtılır. Sunuculara verilen SSL sertifikaları, istemcilerin bağlandıkları sunucuların kimlik doğrulamasının yapılarak sahteciliğin önlenmesini ve istemci-sunucu iletişiminde verilerin şifrelenmesini sağlar.

Sertifika sahibi olmak için web sunucusunun bir "Sertifika İmzalama İsteği (CSR)" oluşturması gereklidir. Bu CSR oluşturma süreci sunucuda açık ve kapalı anahtarların oluşturulmasını sağlayacaktır. Güvenilir bir sertifika sağlayıcısına gönderilen CSR dosyası, sunucuya ait açık anahtarı barındırmaktadır. Sertifika otoritesi tarafından bu CSR dosyasına ait sertifikayı oluşturur ve bu sertifika web sunucusuna yüklenir.

İhtiyaca göre 3 farklı SSL sertifikası vardır bunlar:

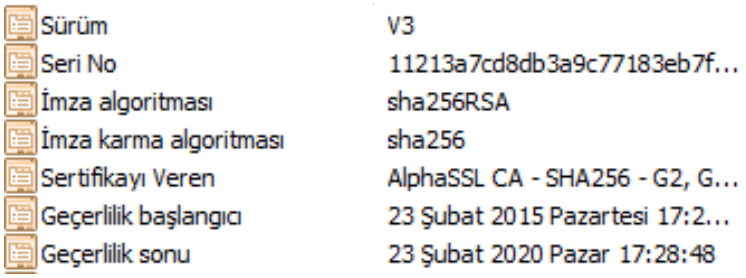
SSL sertifikası: Tek bir sunucu adı için verilen bir sertifikadır. Sertifikanın içinde adı yazılı olan sunucuya yüklenerek SSL güvenliği sağlar.

SAN SSL sertifikası: Birden fazla sunucu veya alan adını (domain) içerebilen bir SSL sertifikasıdır. Ortalama olarak 10-15 farklı sunucu adını içerebilir.

Wildcard SSL sertifikası: Tek bir alan adı için alınabilen ve o alan adının tüm alt alan adlarını kapsayan bir sertifikadır.

Wilcard ile SAN SSL sertifikası arasındaki fark aynı alan adı için kullanılan birçok sunucu sertifikası için Wildcard SSL sertifikaları; aynı kurumun farklı alan adlarına ait sunucular için ise SAN SSL sertifikaları kullanılabilir.

SSL sertifikalarında kriptografik olarak bir şifreleme algoritması ve bir özetleme ("hashing") algoritması kullanılmaktadır. Yaygın olarak kullanılmakta olan RSA, DSA, ECC gibi asimetrik şifreleme algoritmaları ve MD5, SHA-1, SHA-256 gibi özetleme algoritmaları SSL sertifikalarının da temelini oluşturur.



E-devlet'in RSA asimetrik şifreleme algoritması ve SHA-256 özetleme algoritmasını kullanmaktadır.

1.2.3. SSL Bağlantısının Oluşması

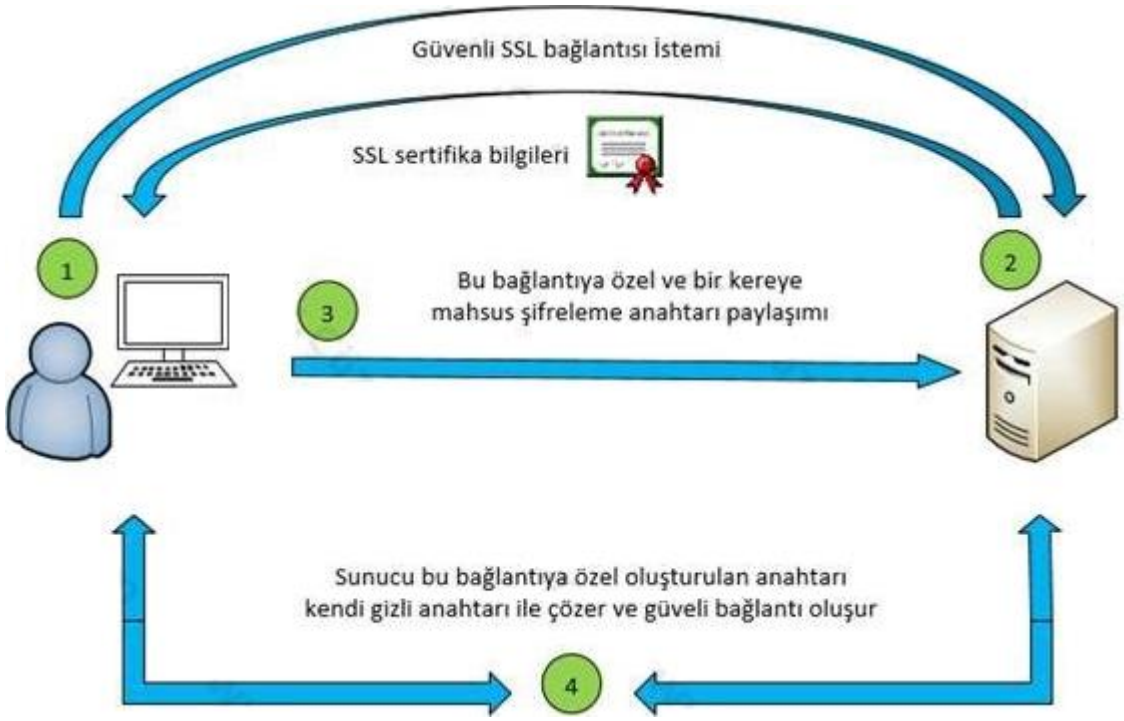
Adım 1: İstemci, bilgisayar ağı üzerinde bağlanmak istediği sunucuya güvenli olarak bağlanma talebi (https) gönderir. Bu işlem, varsayılan ayarların etkin olduğu durumda, SSL portu olarak tabir edilen 443 portu üzerinden yapılır. Sunucu yapılandırmasına bağlı olarak farklı bir port numarası da kullanılabilir.

Adım 2: Ardından, sunucu SSL sertifikasını istemciye gönderir. Bunun üzerine istemci bu sertifikayı üreten kurumun sağladığı çevrimiçi doğrulama servislerini kullanarak bu sertifikanın geçerliliğini kontrol eder. Ayrıca, bağlantı sırasında kullanılmakta olan web tarayıcısı da bu sertifikayı üreten kurumun güvenilirliğini tarayıcıda önceden tanımlı olan kurumlarla kıyaslayarak kontrol eder. Bu aşamada sunucu, açık anahtarını istemciyle SSL sertifikası aracılığıyla paylaşmış olur.

Adım 3: Eğer tüm kontroller doğrulanırsa, istemci sunucuya tek kullanımlık simetrik oturum anahtarını ("session key" veya "encryption key"), sunucunun açık anahtarıyla şifreleyerek gönderir.

Adım 4: Sunucu, gizli anahtarını kullanarak şifreyi çözer ve tek kullanımlık simetrik oturum anahtarını alarak, istemciyle arasında güvenli bir SSL bağlantısı kurar. Böylece, bağlantı süresince bu iki cihaz arasındaki tüm iletişim güvenli bir şekilde devam eder.

SSL bağlantısının oluşmasının görsel anlatımı



1.3. Şifreleme Algoritmaları

Bir göndericinin alıcıya açık ağ üzerinden bir ileti göndermek istediği zaman, açık ağda gönderilen iletiler üçüncü şahıslar tarafından dinlenme ve değiştirilme tehdidi altındadır. Burada söz konusu ileti düz metin (plaintext)'dir. Bir iletinin içeriğini saklamak üzere yapılan gizleme işlemi de şifrelemedir (encryption). Bu işlem düz metini anahtar kullanarak şifreli metine dönüştürmektedir. Şifrelenmiş bir ileti şifreli metindir (ciphertext). Şifreli metini düz metine geri çevirme işlemi şifre çözümüdür (decrypt). Şekil 1'de bu süreç gösterilmiştir.

Şekil 1 Şifreleme ve şifreyi çözme işlemleri

Simetrik Şifreleme: Şifreleme ve şifre çözmenin daha önceden paylaşılmış bir anahtar ile yapıldığı şifreleme türüdür. 128 veya 256 bitlik anahtarlar kullanılabilir. Web sunucusu ve tarayıcı arasında SSL Handshake sırasında tasarlanan oturum (session key), simetriktir.

Asimetrik Şifreleme: Şifreleme ve şifre çözmenin ayrı anahtarlar ile yapıldığı şifreleme türüdür. Burada herkes public (açık) anahtar ile şifreli mesaj gönderebilmekte, yalnız private (kapalı) anahtar sahibi tarafından bu mesajlar çözümlenebilmektedir.

Asimetrik ve simetrik şifrelemenin nasıl kullanıldığı aşağıda belirtilmiştir.

- Sunucu tarayıcıya asimetrik public anahtarını gönderir.
- Tarayıcı simetrik oturum anahtarını oluşturur. Sunucu asimetrik anahtarı ile oturum anahtarını şifreler ve web sunucusuna gönderir.
- Sunucu kendi private anahtarı ile şifreyi çözer ve simetrik oturum anahtarını elde eder.
- Sunucu ve tarayıcı bu simetrik anahtarı kullanarak güvenli bir oturum başlatabilir.

1.3.1 Simetrik şifreleme algoritmaları

Simetrik şifreleme algoritmaları şifreleme ve şifre çözme işlemleri için tek bir gizli anahtar kullanmaktadır. Bu durum veri şifreleme için matematiksel açıdan daha az problem çıkaran bir yaklaşımdır ve çok kullanılan bir yöntemdir. Bu tip algoritmalarda şifreleme işlemi gerçekleştirildikten sonra şifreli metni alıcıya gönderirken şifreli metinle birlikte gizli anahtarı da alıcıya güvenli bir şekilde göndermek gerekmektedir. Simetrik şifreleme algoritmaları çok hızlı bir şekilde şifreleme ve şifre çözme işlemlerini gerçekleştirebilmektedir. Tablo 1'de çeşitli simetrik şifreleme algoritmaları hakkında bilgiler verilmiştir.

Tablo 1: Bazı simetrik şifreleme algoritmaları bilgileri

Algoritmanın Adı	Geliştiren	Tarihi	Tipi (Blok Uzunluğu)	Anahtar Uzunluğu	Döngü Sayısı	Çözülme Durumu	Kullanım Koşulları
DES (Data Encryption Standard)	IBM (ABD)	1977	Feistel Blok (64 bit)	56 bit (parity ile 64 bit)	16	Sağlam; 8 döngülü çeşidi çözülebiliyor; 16 zayıf anahtar	Serbest
IDEA (International Data Encryption Algorithm)	Lai-Massey, ETH Zurich (İsviçre)	1992	Blok (64 bit)	128 bit	8	Sağlam; 2^{51} zayıf anahtar	Ticari faaliyetler hariç serbest
RC2 (Rivest's Cipher veya Ron's Code2)	Rivest, RSA Data Security (ABD)	1992	Katar	2048 bite kadar	Bilinmiyor	Zayıflık bulunmadı	Algoritma RSA tarafından saklı tutuluyor
RC5 (Rivest's Cipher veya Ron's Code5)	Rivest, RSA Data Security (ABD)	1995	Blok (32, 64 veya 128 bit)	2048 bite kadar	255'e kadar	64 bit blok ve 12 döngü ile diferansiyel ve doğrusal şifre çözüme dayanıklı	Serbest
GOST 28147-89	I.A.Zabotin, G.P.Glazkov, V.B.Isaeva (Sovyetler Birliği)	1989	Feistel Blok (64 bit)	256 bit; 512 bit tanımlanabilir sübsitüsyon; 610 bit etken gizli bilgi	32	SSCB tarafından bütün gizlilik derecelerindeki bilgiler için uygun görülmüştür	Serbest
ASEKAL-21	Aselsan (Türkiye)	-	Doğrusal olmayan katar	57 bit ?	-	Ulusal olarak onaylanmış algoritma	Aselsan 2101, 2010 veri ve ses şifreleme birimlerinde kullanılıyor

1.3.2. Şifrelemede kullanılan anahtar boyutları

Anahtarın deneme-yanılma yöntemiyle bulunmasını engellemek için, kullanılan anahtarların uzunluğunun mümkün olduğunca büyük olması gerekmektedir. Tablo 2'de farklı anahtar boyları için, saniyede bir milyon, bir milyar ve bir trilyon şifre deneyebilen bilgisayarlar için anahtar çözme süreleri verilmiştir.

Tablo 2 Farklı anahtar boyutları için anahtar çözme süreleri

Anahtar Uzunluğu (n)	**Olası değer sayısı (2^n)**	**10^6 şifre/s hızında ortalama çözme süresi**	**10^9 şifre/s hızında ortalama çözme süresi**	**10^{12} şifre/s hızında ortalama çözme süresi**
32 bit	~$4x10^9$	36 dak	2.16 s	2.16 ms
40 bit	~10^{12}	6 gün	9 dak	1 s
56 bit	~$7.2x10^{16}$	1142 yıl	1 yıl 2 ay	10 saat
64 bit	$1.8x10^{19}$	292 000 yıl	292 yıl	3.5 ay
128 bit	$1.7x10^{38}$	$5.4x10^{24}$ yıl	$5.4x10^{21}$ yıl	$5.4x10^{18}$ yıl

1.3.3. Sayısal imza

Sayısal imza elektronik mesaja eklenmiş bilgidir. Çift anahtarlı bir şifreleme algoritmasıyla hazırlanan sayısal imza, hem gönderilen bilginin sayısal içeriğinin değiştirilmediğinin hem de gönderen tarafın kimliğinin ispatlanması için kullanılır ve gönderilecek mesajdan üretilen "mesaj özetinin" sayısal içeriği, gönderen tarafın kendi özel anahtarına bağlı olarak oluşturulur. Sayısal imzanın doğruluğunu kanıtlamak için mesajı alan taraf, kendisine gelen mesajın ve sayısal imzanın sayısal içeriği ile gönderen tarafın açık anahtarını kullanmaktadır.

1.3.4. Şifreleme Modları

Simetrik anahtar şifreleme, modern blok şifreler kullanılarak yapılabilir. DES veya AES kullanılan herhangi bir boyuttaki metni şifrelemek için operasyon modları tasarlanmıştır.

Burada blok seviyesindeki desenler korunur. Blok bağımsızlığı, anahtar bilinmeden bazı şifreli metin blokları değiştirmesi için fırsatları içinde barındırdığı için güvenlik sorunu teşkil edilmektedir. Aktarımdaki tek bir bit hatası, ilgili blokta birkaç hata oluşturabilir. Bu yönden hata yayılımı konusunda, hatanın diğer bloklar üzerinde herhangi bir etkisi yoktur. ECB'de dolgulama (padding) işlemi vardır.

Şifreleme modlarının tablo halinde incelenmesi

Operation Mode	*Description*	*Type of Result*	*Data Unit Size*
ECB	Each n-bit block is encrypted independently with the same cipher key.	Block cipher	n
CBC	Same as ECB, but each block is first exclusive-ored with the previous ciphertext.	Block cipher	n
CFB	Each r-bit block is exclusive-ored with an r-bit key, which is part of previous cipher text	Stream cipher	$r \leq n$
OFB	Same as CFB, but the shift register is updated by the previous r-bit key.	Stream cipher	$r \leq n$
CTR	Same as OFB, but a counter is used instead of a shift register.	Stream cipher	n

Şifreleme modlarında kullanılan blok şifreleme yönteminin girişten gelen veriye bağımlılığı kötü bir özelliktir ve şifrenin çözülme oranını arttırmaktadır. Şifreleme yaparken kullanılan uzayın büyüklüğü, şifrelemenin kuvvetini etkiler.

Uygulanan şifreleme modunun etkisiyle şifrelenen açık metnin geri dönüşü bir o kadar önemli bir noktadır. Bu noktada CTR (Counter Mode Encryption) modunun geriye dönüşü bulunmamaktadır. Burada aslında mükemmelinde tekrar etmeyen bir sayı üretecine dayanmaktadır. Yani matematiksel olarak bir fonksiyon sürekli sayı üretmekte ancak bu üretilen sayılar kendini tekrar etmemektedir. Ancak bu şekildeki fonksiyonun da ürettiği sayıların mümkün olduğunca rastgele olması ve önceden kestirilemez olması istenir.

1.4. Kullanılan Programlama Dilleri ve Platformlar

Java James Gosling tarafından geliştirilmeye başlanmıştır. Sun Microsystem Java'yı ilk olarak 1995 yılında piyasaya sunmuştur. Java günümüzde neredeyse her platformda kullanılmaktadır. Nesneye dayalı, açık kodlu ve yüksek seviyeli bir programlama dilidir. Java dili C ve C++ dan birçok sözdizimi ve yapı kullanmasına rağmen daha basit ve anlaşılabilirdir. Java platformu aynı yazılımın birçok değişik bilgisayar ortamında veya değişik tür makinalarda çalışması fikri ile geliştirilmiştir. Platformdan (Unix, Microsoft, MacOS) bağımsızdır. İnternet üzerinden uygulama geliştirme yapılabilir. Java yazılımlarını geliştirmek için JDK (Java Development Kit) programının kurulumu yapılması gerekmektedir. En çok tercih edilen programlama dillerinden biridir.

Eclipse IBM Kanada'nın bir projesi olarak 2001 yılında Object Technology International tarafından piyasa sunulmuştur. Açık kaynak kod geliştirme ortamıdır. Temel olarak Java ile ilgili bir teknoloji ise de C, C++, Rubby gibi programlama dilleri içinde kullanılır. Basit ara yüzü ve kolay kullanımı ile çok kısa bir zamanda en çok kullanılanlar arasına girmiştir. Java da olduğu gibi Eclipse de platformdan bağımsızdır. Unix, Microsoft, MacOS işletim sistemlerini desteklemektedir.

Yazılım sektörü içerisinde en sık kullanılan iki yazılım dili olan C ve C++ etkileşimi ile türetilmiştir. Ayrıca C#, ortak platformlarda taşınabilir bir (portable language) programlama dili olan Java ile pek çok açıdan benzerlik taşımaktadır. En önemli özelliği ise .Net Framework platformu için hazırlanmış tamamen nesne yönelimli bir yazılım dilidir. Yani nesneler önceden sınıflar halinde yazılıdır. Programcıya sadece o nesneyi sürüklemek ve sonrasında nesneyi amaca uygun çalıştıracak kod satırlarını yazmak kalır.

Microsoft Visual Studio, Microsoft tarafından geliştirilen bir tümleşik geliştirme ortamıdır (IDE). Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework ve Microsoft Silverlight tarafından desteklenen tüm platformlar için yönetilen kod ile birlikte yerel kod ve Windows Forms uygulamaları, web siteleri, web uygulamaları ve web servisleri ile birlikte konsol ve grafiksel kullanıcı arayüzü uygulamaları geliştirmek için kullanılır.

1.5. İstemci – Sunucu (Client – Server) Mimarisi

Ağdaki her bilgisayarın rolü farklıdır. Bu roller istemci (client) ve sunucu (server) olabilir. İstemci genel olarak kullanıcı ihtiyaçlarını etkileşime sokmak için sunum hizmetleri, veri tabanı hizmetleri ve bağlantı sağlayan bir yapıdır.

Sunucu ise iş prosedürleriyle ilgili arabirimlerin yanı sıra bağlantı ve veri tabanı hizmetleri sağlayan ayrıca istemciye göre daha yüksek donanım özelliklerine sahip bir istasyondur. Bu mimaride merkezde sunucu bilgisayar bulunur ve kaynakların kullanımını, kullanıcıların yetkilerini ayarlamakla sorumludur. İstemci bilgisayarların tümü sunucu bilgisayara bağlıdır ve sunucunun verdiği yetkiler doğrultusunda işlemler gerçekleştirilir. İstemci tek kullanıcılı gibi davranırken sunucu çok kullanıcılı yapıya sahiptir. Bunun nedeni istemci tek bir hizmeti almak için sunucuya istek gönderirken, sunucu tarafında ise birden çok istemcinin hizmet talebinin sağlanması amaçlanmaktadır.

Genellikle, birden çok istemci programı, ortak bir sunucu programının hizmetlerini paylaşır. Hem istemci programları hem de sunucu programları genellikle daha büyük bir programın veya uygulamanın bir parçasıdır. Örneğin web tarayıcıları (Web'in gönderilmesi sayfalar veya dosyalar) internetteki başka bir bilgisayarda bulunan web sunucusundan (teknik olarak Köprü Metni Aktarım Protokolü veya http sunucusu olarak adlandırılır) servis talep eden bir istemci programıdır.

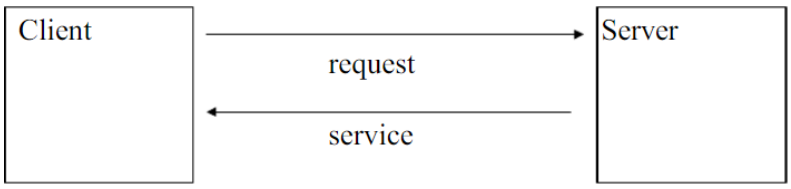
Sunucu yapısı gereği çoklu-thread yapısını kullanmaktadır. Bunun için öncelikle thread'i açıklanması gerekir. Thread, bir program içerisinde farklı kod kısımlarını aynı zamanda çalıştırılmasının bir yoludur. Bunun yanında socket kullanımıda önemlidir. Socket, süreçler arası iletişim için kullanılır. Interprocess iletişimi istemci-sunucu modeline dayanır. Bu durumda, istemci-sunucu birbiriyle etkileşime giren uygulamalardır. İstemci ve sunucu arasındaki etkileşim için bir bağlantı gereklidir. Soket programlaması, etkileşime girecek uygulamalar arasındaki bağlantıyı kurmaya yarar. Kısacası ağ iletişiminin gerçekleşmesini sağlayan yapıdır.

Sunucu – istemci yapısı ile internetteki herhangi bir siteye bağlanırken bu yapıdan faydalanılır. Sitenin sunucusu varsayılan olarak belirtilmiş olan porta gelen bağlantı istekleri için sürekli dinler. Gelen bağlantıları diğer bir port ile meydana getirdiği soket ile tanımlar. Bundan dolayı sunucu sürekli varsayılan portu dinlemiş olur.

İstemci bağlantı isteğinde bulunur. Eğer sunucu, istemcinin isteğini kabul ederse ona bir soket nesnesi açar ve farklı bir porttan kendisiyle iletişim kurmasını sağlar. Sunucu tarafında sürekli olarak bir dinleyici (Listener) soketi bulunur. Bu soketler belli bir port üzerinde dinleme yaparlar. Tüm bağlantı isteklerini bu port üzerinden kabul eder ve ya reddeder. Wireshark programı kullanırken yukarıda bahsedilen yapıların da kullanımı gerekmektedir.

Açık metni şifreli olarak gönderen taraf ile şifreli metni alıp şifresini çözen alıcı taraf (İstemci-sunucu) arasında oluşan şifreli iletişimi görebilmek adına sıkça kullanılan wireshark programı kullanılmıştır. Wireshark, bağlanılan internetin tüm protokolleri kullanarak ne tip veriler işlendiğini görebilmemize olanak sağlayan bir programdır. Anlık olarak da, daha önce kaydedilmiş veriler de tekrar tekrar gözden geçirilebilir.

Wireshark, Windows bilgisayara kurulduğu zaman WinPcaP adında bir araçla beraber gelir. WinPcaP, o an bağlı olunan internet bağlantısını, bilgisayar donanımı olan Ethernet kartı üzerinden izlemesine yardımcı olur. Aslında işi WinPcaP halleder ama rahat kullanılması ve grafiksel olması için Wireshark arayüzü kullanılır. Bu program ile şifrelenen metin ağ üzerinde incelenerek doğruluğu test edilmiştir. Burada inceleme yaparken ağ üzerinde başka paketlerde gönderilmiştir. İnceleme yaparken buna dikkat edilmiştir. Ayrıca inceleme esnasında görmek istedinen şifreli metinde de kayıplar oluşabilmektedir.



İstemci-Sunucu mimarisinde gelişen olayların özeti

- Kullanıcı herhangi bir işlem yapmak için istemci yapısını çalıştırır.
- İstemci sunucuya bağlanır.
- İstemci sunucuya istek gönderir.
- Sunucu bu isteği analiz eder.
- Sunucu isteğin sonucunu bulur.
- Sunucu bulduğu sonucu istemciye iletir.
- İstemci sonuçları kullanıcıya gösterir.
- Bu işlemler gerektiği kadar tekrarlanır.

2. YAPILAN ÇALIŞMALAR

2.1. Gereksinim Analizi

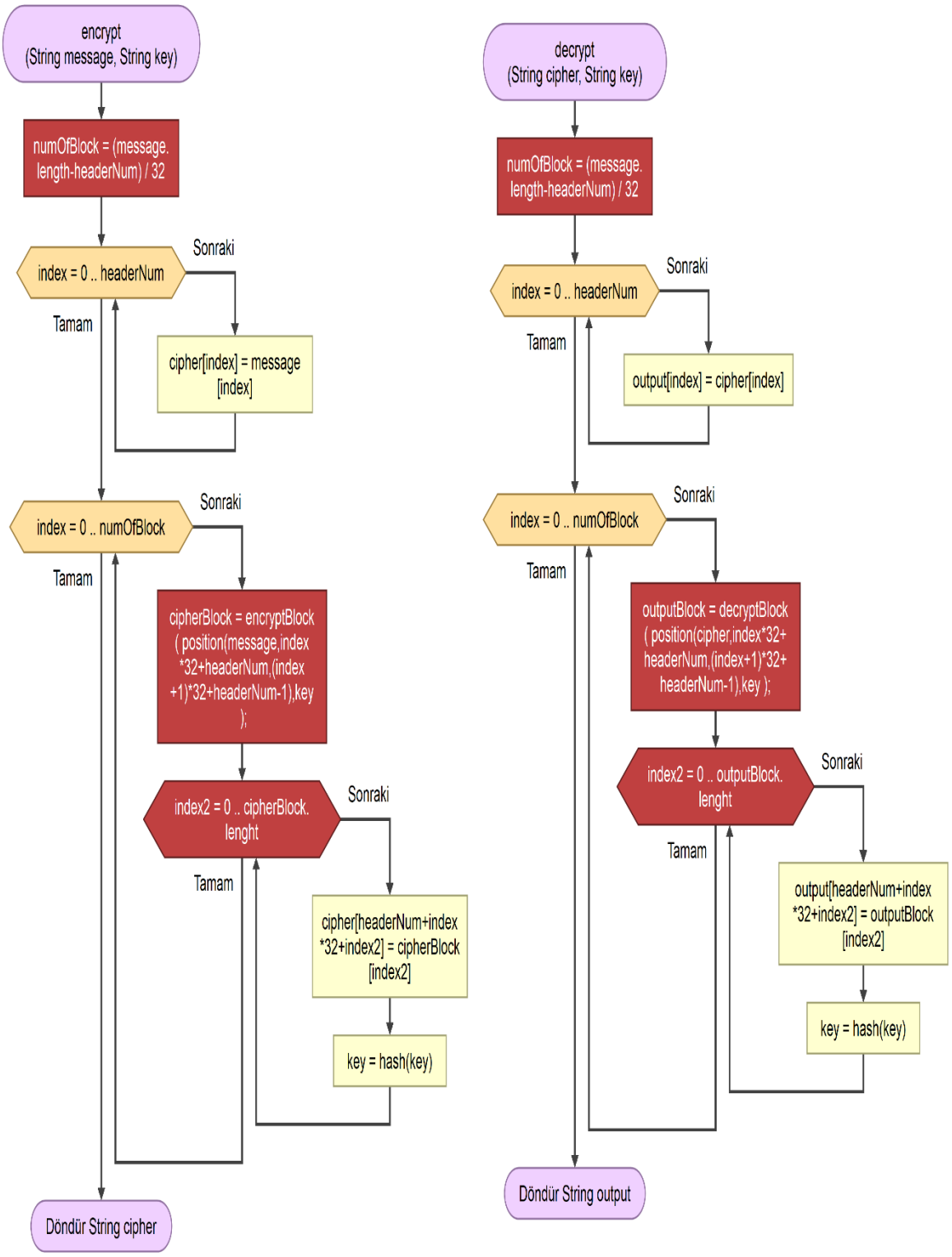
Algoritma Tasarımı:

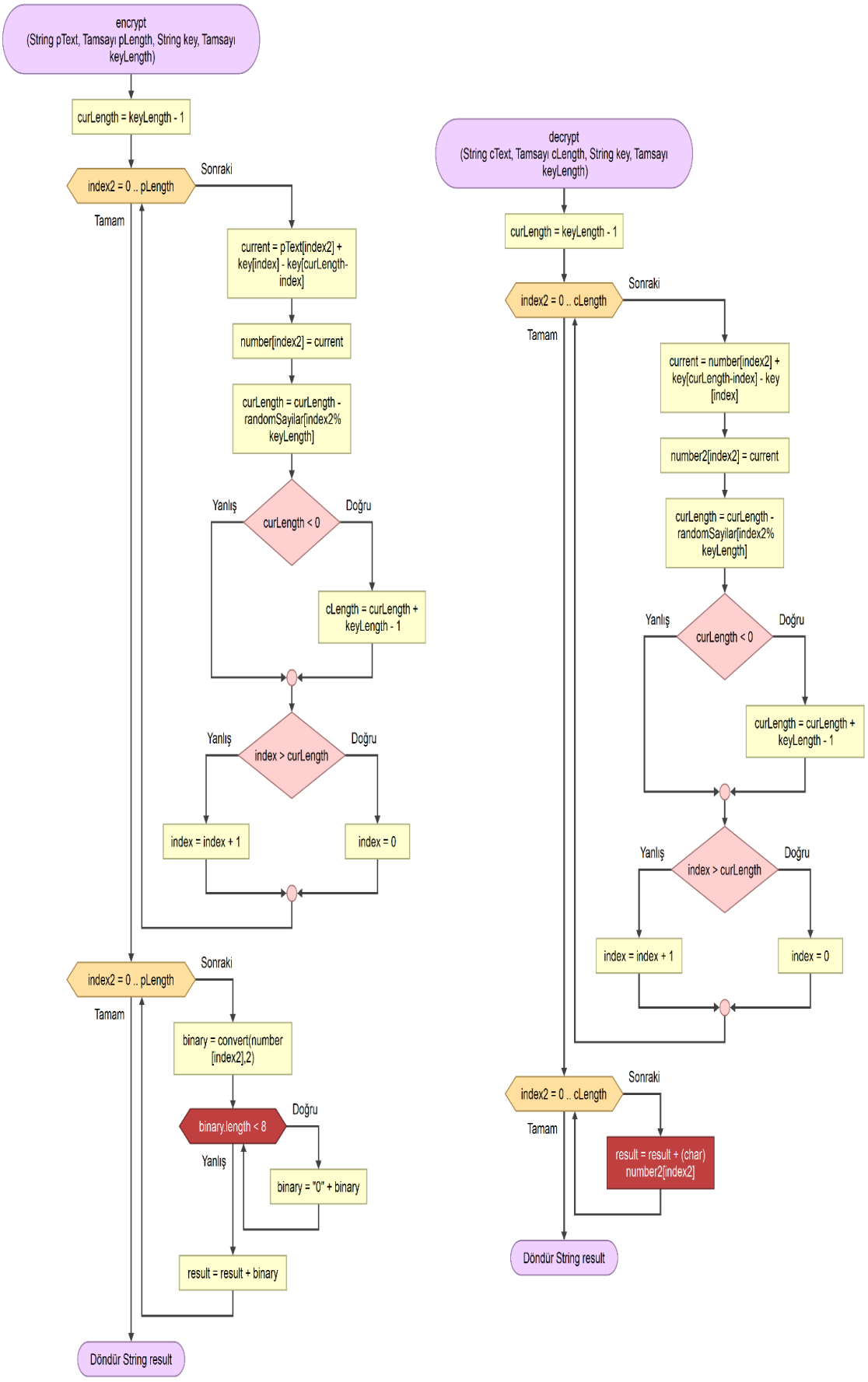
- SSL (Secure Socket Layer) protokolüne benzer, günümüz bilgisayarlarının kıramayacağı güçlükte şifreleme yapan algoritmaların gerçeklenmesi
- Algoritmanın optimizasyonu
- Algoritmanın bir programlama dili ile implementasyonu
- Şifreleme algoritmalarının deşifreleme doğruluğu, kararlılık ve hız testi
- Algoritmaya şifreleme tekniklerinde kullanılan test yaklaşımlarına karşı dayanıklılık kazanması
- Şifreleme yöntemlerinde anahtar için hash (özet fonksiyon) kullanımı

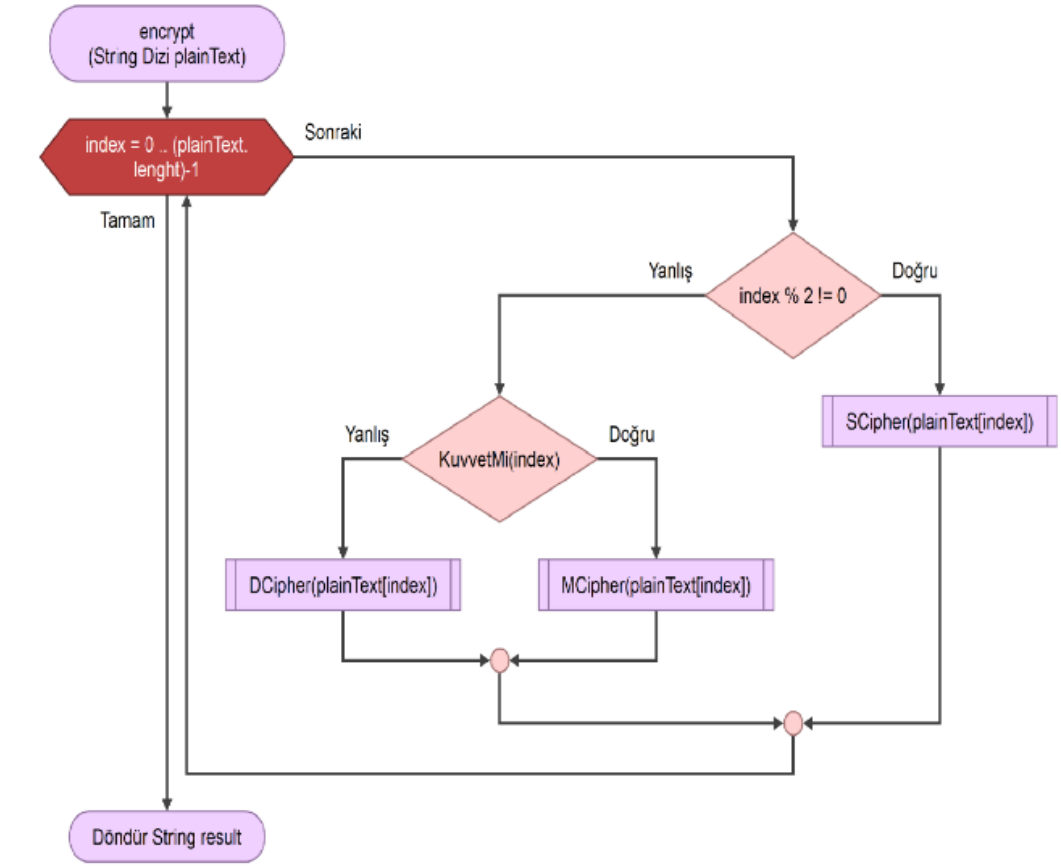
Client&Server:

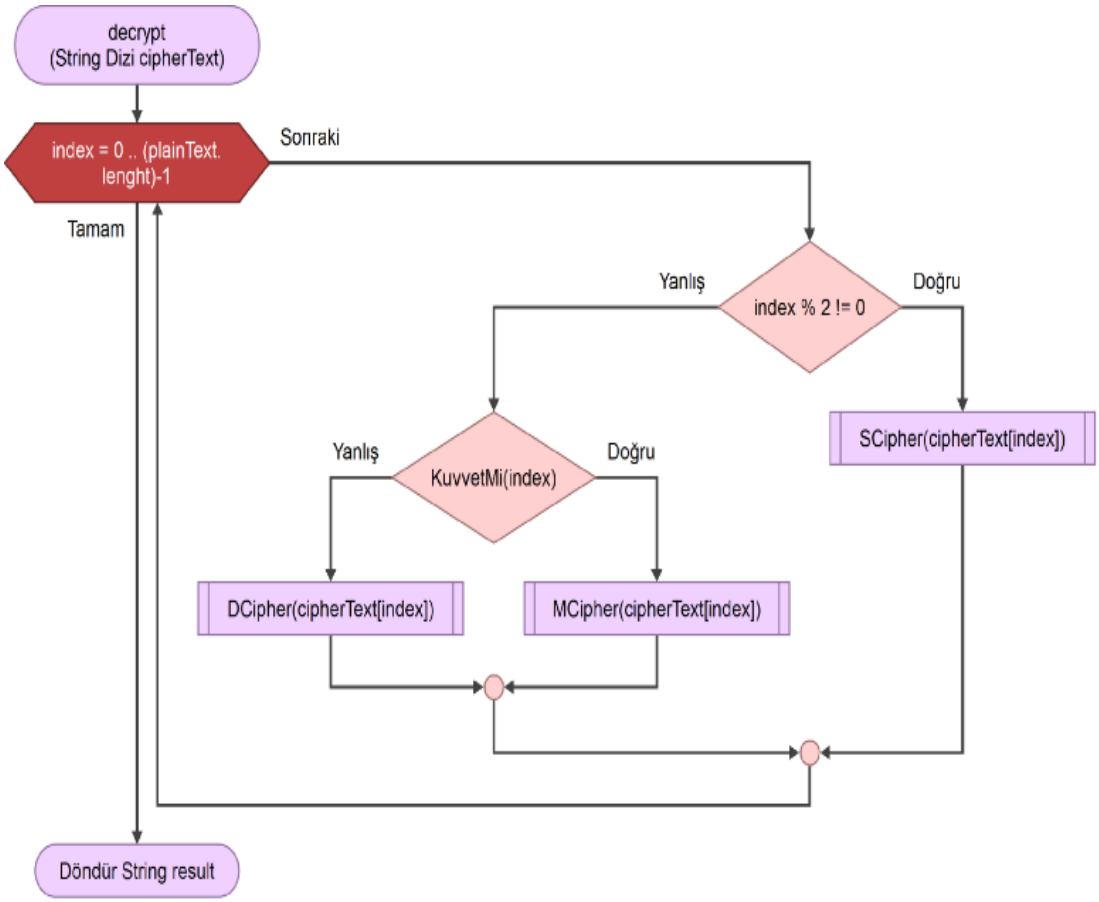
- Client tarafında algoritmalarımızı kullanarak şifreleme işlemini gerçeklenmesi
- Şifrelenmiş metnin http (değişebilir) protokolü üzerinden sunucuya gönderilmesi
- Sunucu üzerinde şifreli gelen metnin, deşifrelenerek plaintext'in elde edilmesi
- Elde edilen plaintext'in karşılığında sunucudan client'a geri gönderilecek metnin, tekrardan aynı algoritma kullanılarak şifrelenmesi
- Client tarafında şifrelenmiş metnin çözülerek iletişimin tamamlanması

2.2. Mimari Tasarım

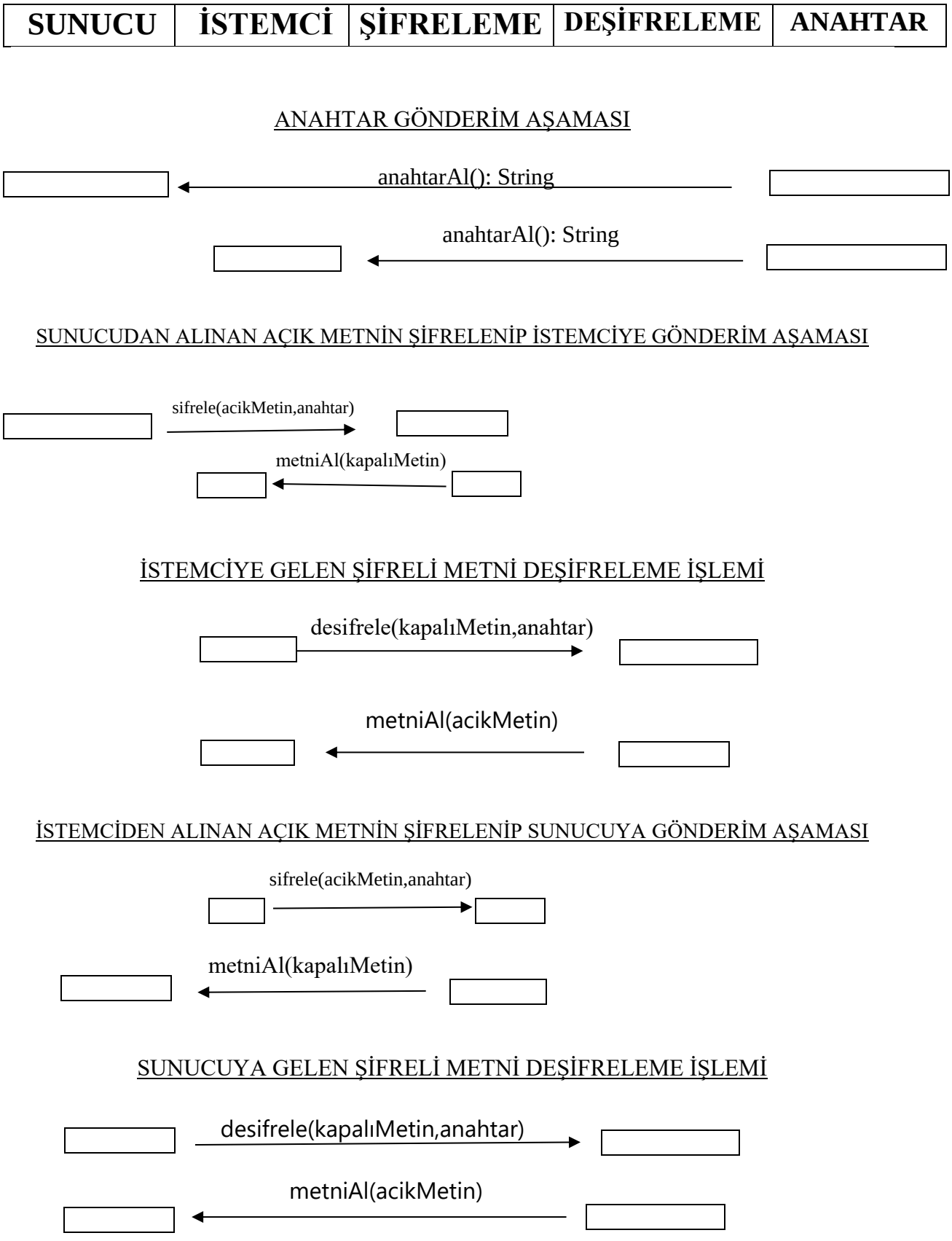



encrypt
(String pText, Tamsayı pLength, String key, Tamsayı keyLength)
curLength = keyLength - 1
index2 = 0 .. pLength
Sonraki
Tamam
current = pText[index2] + key[index] - key[curLength-index]
number[index2] = current
curLength = curLength - randomSayilar[index2% keyLength]
Yanlış
curLength < 0
Doğru
cLength = curLength + keyLength - 1
Yanlış
index > curLength
Doğru
index = index + 1
index = 0
index2 = 0 .. pLength
Sonraki
Tamam
binary = convert(number [index2],2)
binary.length < 8
Doğru
Yanlış
binary = "0" + binary
result = result + binary
Döndür String result
decrypt
(String cText, Tamsayı cLength, String key, Tamsayı keyLength)
curLength = keyLength - 1
index2 = 0 .. cLength
Sonraki
Tamam
current = number[index2] + key[curLength-index] - key [index]
number2[index2] = current
curLength = curLength - randomSayilar[index2% keyLength]
Yanlış
curLength < 0
Doğru
curLength = curLength + keyLength - 1
Yanlış
index > curLength
Doğru
index = index + 1
index = 0
index2 = 0 .. cLength
Sonraki
Tamam
result = result + (char) number2[index2]
Döndür String result


encrypt
(String Dizi plainText)
index = 0 .. (plainText.
lenght)-1
Sonraki
Tamam
Yanlış
index % 2 != 0
Doğru
SCipher(plainText[index])
Yanlış
KuvvetMi(index)
Doğru
DCipher(plainText[index])
MCipher(plainText[index])
Döndür String result


decrypt
(String Dizi cipherText)
index = 0 .. (plainText.
lenght)-1
Sonraki
Tamam
Yanlış
index % 2 != 0
Doğru
SCipher(cipherText[index])
Yanlış
KuvvetMi(index)
Doğru
DCipher(cipherText[index])
MCipher(cipherText[index])
Döndür String result

2.3. Şifreleme Algoritmalarının UML Sıralı Diyagramı (Sequence Diagram)

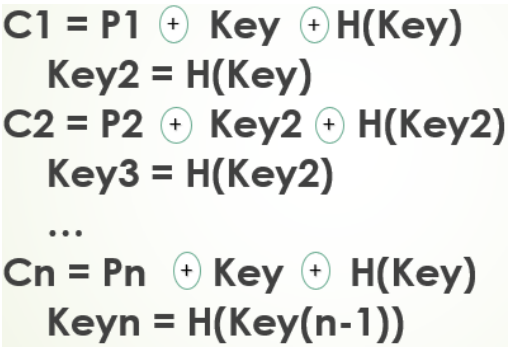


2.4. Yapılan Çalışmalar

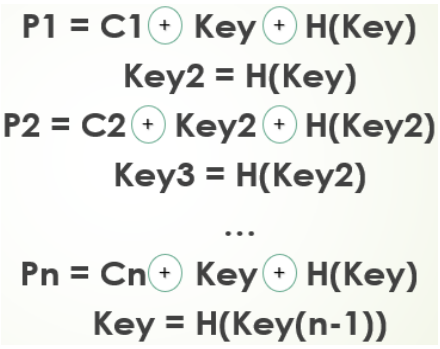
2.4.1. Simetrik Şifreleme Tasarımları

2.4.1.1. Simetrik Şifreleme Algoritması – 1

Simetrik şifreleme tasarımı için ilk önce kullanılacak yapıların belirlenmesi gerekmektedir. Bu süreçte şifrelenecek açık metin (plaintext), SHA256 özet fonksiyonu ile şifrelenir. SHA256 ile şifrelemek için açık metin 256 bitlik parçalara bölünür. Açık metin parçaları sırasıyla anahtar ve hash(key) ile XOR'lanır. Her şifrelenmesinden sonra anahtar hashlenmiş anahtar ile güncellenmektedir.



Blok Şifreleme Kısmı



Blok Deşifreleme Kısmı

Deşifreleme işleminde de şifrelenmiş metin 256 bitlik parçalara bölünür. Şifrelenmiş metin parçası anahtar ve hashlenmiş key ile XOR'lanır. Bu işlemle açık metin parçası elde edilir. Her metin parçasının şifrelenmesinden sonra anahtar hashlenmiş anahtar ile değiştirilir.

```
private static byte[] XOR(byte[] a,byte[] b)
{
      if(a.length!=b.length)
      {
         System.err.println("ERROR -> Array lengthes are different.");
         return null;
      }
      byte[] c = new byte[a.length];
      for(int i=0; i<c.length; i++)
         c[i]=(byte) (a[i] ^ b[i]);
      return c;
}
```

Açık metni şifrelerken kullanılan anahtarla birlikte yapılan XOR işlemi için bir metot tanımlanmaktadır. Burada alınacak 2 tane dizinin boyutunun eşit olması gerekir.

Şifreleme yapmak için kullanılacak metotta (encrypt), mesajı ve anahtarı byte array tipinde alır. Daha sonra 32 byte'lık parçalara bölünür. Toplam blok sayısını bulmak için, gelen mesaj uzunluğunda boş bir dizi açılır. Bu dizinin ilk 35 değeri direkt cipher'a aktarılır. Bu kısmın şifrelenmemesinin sebebi; dosyanın türünü değiştirmek istenmemesidir.

```
String file_path = "C:\\Users\\Hknaksoyy\\Desktop\\a.jpg";
File input_file = new File(file_path);
byte[] plainText = Files.readAllBytes(input_file.toPath());
String key = "asdfgytasdfgytrfghtuytorfghtuyto";
byte[] cipher = encrypt(plainText,key.getBytes());
Path path = Paths.get("C:\\Users\\Hknaksoyy\\Desktop\\b.jpg");
Files.write(path, cipher);
```

Bu kısımda ilk olarak şifrelenecek dosyanın konumu yazılır. Okunacak dosya bitlere çevrilip diziye aktarılır. Belirlenecek gizli anahtar 32 karakter uzunluğunda bulunmalıdır. Daha sonra şifrelenmiş mesaj cipher adlı diziye koyulup, istenilen konuma koyulabilmektedir.

```
String file_path = "C:\\Users\\Hknaksoyy\\Desktop\\b.jpg";
File cipher_file = new File(file_path);
byte[] cipher = Files.readAllBytes(cipher_file.toPath
String key = "asdfgytasdfgytrfghtuytorfghtuyto";
byte[]output = decrypt(cipher,key.getBytes());
Path path2 = Paths.get("C:\\Users\\Hknaksoyy\\Desktop\\c.jpg");
Files.write(path2,output);
```

Daha sonra şifrelenmiş mesajın konumundan şifrelenmiş mesajı cipher isimli bir dizinin içine aktarılır. Bu kısımda yapılacak deşifrelemek işleminde kullanılacak anahtarın, şifrelenirken kullanılan anahtarla aynı olması gerekir. Deşifreleme işlemi tamamlandık sonra çözülmüş mesajı yazdırmak istenilen konuma gönderilebilmektedir.

Açık Metin

Şifrelenmiş Metin

Deşifrelenmiş Metin

2.4.1.2. Simetrik Şifreleme Algoritması - 2

Şifreleme algoritmasının en başında kullanıcıdan şifrelenmesini istediği input'u alabilmek için bir input değişkeni tanımlandı. Daha sonra Encrypt ve Decrypt fonksiyonlarında kullanılmak üzere sayılar ve sayılar2 adında 2 tane int liste tanımladı. Üretilen random key içinde key adında bir değişen ve Decrypt fonsiyonunda kullanılmak üzere bir int dizi tanımlanmıştır.

```
string input = "";
List<int> sayilar = new List<int>();
List<int> sayilar2 = new List<int>();
string key = "";
int[] random_sayilar = new int[32];
```

Random key üretmek için kurulan algoritmada current değişkenine 65 ile 122 arasında değerler atanmaktadır. Bu değerlerin ASCII karşılıklarını kullanarak harf üretilebilmektedir. 65'in ASCII karşılığı A iken 122'nin ASCII karşılığı z'dir. Yani bu algoritma ile 26 tane farklı (Türkçe ve özel karakterler dahil değil) büyük harf 26 tanede küçük harf tanımlanabilmektedir.

```
for (int i = 0; i < 32; i++)
{
      current = rand.Next(65,123);
      while (current > 90 && current < 97)
                current = rand.Next(65, 123);
      key = key + (char)current;
}
```

Burada dikkat edilmesi gereken nokta ise ASCII tablosunda 91 ile 96 arasındaki değerler özel karakterler olduğu için, 91 ile 96 arasında bir değer üretildiğinde bu değer aralığındaki değerler gelmeyene kadar random sayı üretimine devam edilir. Üretilen değerler ASCII değerlerine karşılık düşen karakterlere dönüştürülerek key değişkenine atanır ve bu döngü 32 kez dönerek 32 karakterli random bir key üretilir.

```
for (int i = 0; i < key.Length; i++)
{
      random_sayilar[i] = key[i] % key.Length;
}
```

Buradaki algoritmada ise key uzunluğu kadar for döngümüz dönmektedir ve her dönüşünde key de ki harflerin ASCII değerlerinin mod 32'sini almaktadır. Örneğin *A harfi* için 65(A'nın ASCII karşılığı)%32 = 1 olarak random_sayilar dizimize atanmaktadır. Diğer bir örnek ise *a harfi* için inceleyecek olursak 97(a'nın ASCII karşılığı)%32 = 1 olarak random_sayilar diziye atanmaktadır. Bu diziyi daha sonra Decrypt fonksiyonda şifreli mesajı(cipher text) çözmek için kullanılmaktadır.

Alttaki kısımda şifrelemede kullanılan current_length ve j değişkenleri tanımlandı. Current_length değişkeninin ilk değeri keyin son indexini gösterecek şekilde, key.Length-1 (31) değerini alır. J değişkeni ise keyin ilk indexini gösterecek şekilde 0 değerine setlenmiştir.

```
int current_length=key.Length-1;
int j=0;
for (int i = 0; i < input.Length; i++)
{
   if (j > current_length)
         j = 0;
int current = int.Parse((input[i]+key[j] - key[current_length - j]).ToString());
sayilar.Add(current);

current_length = (current_length - random_sayilar[i % random_sayilar.Length]);
if (current_length <= 0)
   current_length = key.Length - 1 + current_length;
j++;
}
```

Döngü kullanıcıdan alınan text'in uzunluğu kadar dönmektedir. Döngünün her dönüşünde kullanıcıdan alınan text'in karakterleri sırasıyla şifrelenmektedir. Şifrelemede örnek ile açıklayacak olursak girilen text'in 'bilgisayar' olduğu düşünüldüğünde, current değişkenine (b harfinin ascii karşılığı) + (key'in ilk harfinin ascii karşılığı) – (key'in son harfinin ascii karşılığı) atanmaktadır. Çıkan sonuç sayilar listesine eklenir ve bu sayilar listesi *Decrypt* fonksiyonumuzda kullanılacaktır. Bilgisayar kelimesinin uzunluğu kadar (10) döngümüz diğer harflerinin şifrelenmesi içinde tekrarlanacaktır.

Current_length değişkenini her döngüde anahtarın harfine göre bir sayı atanmaktadır. Örneğin anahtarın o anki harfinin 'O' olduğu düşünülürse, bu harfin ASCII karşılığı 79 % 32 = 15, current_length = 31-15 = 16 olarak güncellenir. Her döngüde key değerine göre current_length değişkeni güncellenebilir, eğer current_length değişkeni 0'dan küçük olursa current_length=(key.Lenght-1) + current_lenght değeri ile güncellenir.

```
for (int i = 0; i < sayilar.Count; i++)
{
     string binary = Convert.ToString(sayilar[i], 2);
     while(binary.Length < 8)
     {
         binary = "0" + binary;
     }
     Output = Output + binary;
}
```

Encrypt fonksiyonunun son kısmında ise şifreleme algoritmasının yardımıyla elde edilen değerlerin binary karşılığını elde edilir. Eğer çevrilen binary değerin uzunluğu 8 değilse, bu uzunluğu 8'e tamamlayana kadar binary değerin başına 0 eklenir ve son olarak bu binary değeri output değişkenine atanmaktadır.

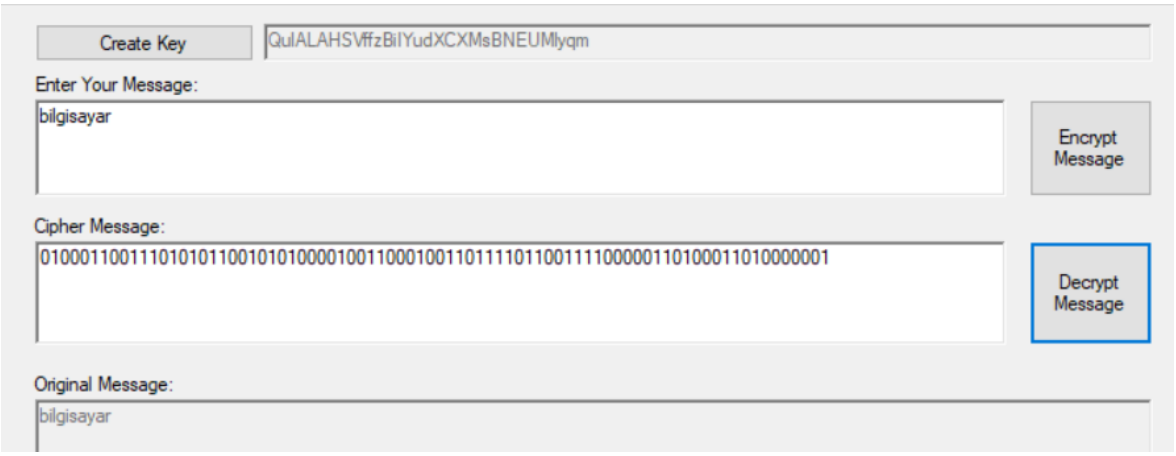
```
int j = 0;
int current_length = key.Length - 1;
for (int i = 0; i < input.Length; i++){
int current=(sayilar[i]+int.Parse((key[current_length-j] - key[j]).ToString()));
sayilar2.Add(current);
```

Encrypt fonksiyonunda da kullanılan j ve current_lenght değişkenleri Decrypt fonksiyonunda da tanımlanıp kullanıldı. Current_length değişkeninin ilk değeri anahtarın son indexini gösterecek şekilde key.Length-1 (31) değerine setlenmiştir. Döngümüz kullanıcıdan alınan text'in uzunluğu kadar dönmektedir. Döngünün her dönüşünde kullanıcıdan alınan text'in karakterleri sırasıyla deşifrelenmektedir.

```
for (int i = 0; i < sayilar2.Count; i++)
       Output = Output + (char)sayilar2[i];
```

Döngünün içerisinde current değişkenine şifreli değerleri tutulan sayilar listesini ve şifrelerken kullanılan keyleri ters şekilde kullanarak, şifreli değeri çözüp sayilar2 listesine eklenmektedir. Çözülmüş ve sayilar2 listesine eklenmiş değerler başta input olarak girilen text deki harflerin ASCII karşılığıdır. En son olarak deşifrelediğimiz değerleri karaktere çevirip ekrana yazdırmaktayız.

Uygulamanın ekran görüntüsü



2.4.1.3. Simetrik Şifreleme Algoritması - 3

Bu yöntemde ilk olarak her türkçe harf karakterine karşılık bir şifreli karakter kullanılmaktadır. Şifrelemedeki key(anahtar) bu karakterlerin tutulduğu tablolar olmaktadır. Bu tablolar her bir açık metnin her karakterinin teklik, çiftlik ve 2'nin kuvveti durumlarına göre oluşturulmuştur. Örneğin "A" karakteri için çift tablosunda "." karakteri belirlenmiştir. Tek tablosu için ise "A" karakterine karşılık "m" karakteri belirlenmiştir.

```
Dictionary<string, string> dictionary = new Dictionary<string, string>();
Dictionary<string, string> dictionary1 = new Dictionary<string, string>();
Dictionary<string, string> dictionary2 = new Dictionary<string, string>();
ArrayList aList = new ArrayList();
```

Yukarıdaki kod kısmında tabloların tanımlamalarını ve tiplerini belirlendi ayrıca girilen karakterlerin tutulduğu dizi listesi oluşturulmuştur.

```
private void dictionaryCift()
{
    dictionary.Add("A", ".");
    dictionary.Add("B", "z");
    dictionary.Add("C", "@");
    dictionary.Add("Ç", "y");
       .             .
       .             .
}
```

Tablolara (tek, çift, ikinin kuvveti) değer atama işlemleri yapıldı. Dictionary anahtar kelimesi ile her karaktere karşılık bir şifreli karakter eşleme işlemi yapılmıştır.

```
for (int i = 1; i <= aList.Count; i++)
{
    int index = i;
    if (index % 2 == 0 && ((index &    (index - 1)) != 0))
              cift(aList[i - 1].ToString());
    else if (index % 2 != 0)
              tek(aList[i - 1].ToString());
    else if (index != 0 && ((index & (index - 1)) == 0))
              two_us(aList[i - 1].ToString());
    aList[index - 1] = " ";
}
```

Bu kısımda girilen düz metnin boyutunun sayısı kadar bu işlemler tekrarlanmaktadır. (Encrypt kısmıdır). Girilen düz metnin her bir karakterinin indis'ine karşılık düşen tablo değerlerine göre şifreli metni oluşturan kısmıdır. (Decrypt kısmı) Şifrenin çözüldüğü kısımda aynı mantıkla çalışmaktadır. Şifreli metnin her bir karakterine karşılık gelen açık metni oluşturur.

Algoritmada için ilk yapılması gereken açık metinin seçimidir. Açık metinde dikkat edilmesi gereken nokta, harflerin konumlarıdır. Harflerin konumlarına göre teklik, çiftlik, ikinin kuvveti şeklinde oluşturulan tablolara göre şifreleme yapılır.

Bir karakterin konumu hem çift hem ikinin kuvveti ise ikinin kuvvetine öncelik verilir. Örneğin "1." Konuma denk gelen karakterin tek tablosuna göre şifrelemesi yapılır. "2." Konuma denk gelen karakterin ikinin üssü tablosuna göre şifrelemesi oluşturulmuştur. "6." Konuma denk gelen karakterin ise çift tablosuna göre şifrelemesi oluşturulmuştur.

Tasarlanan şifreli metnin aynı tablolar yardımıyla alıcı tarafta her bir şifreli karaktere karşılık düşen açık metin karakteriyle çözümlenmesi yapılmaktadır. Yöntemde şifre oluşturulmaya çalışıldığında bir harfin farklı konumlara göre şifreli karakter karşılığı değişecektir. Bu yöntemin böyle kullanılması, bruteforce ataklarına karşı dayanıklılığını söz konusudur. Ayrıca farklı kombinasyonlara göre belirlenen tabloların sayısı arttırılarak da şifrelemenin sağlamlığı arttırılabilir.

Bu şifrelemenin oluşturulmasında günümüzde kullanılan simetrik şifreleme tekniklerinden faydalanılmıştır. Bu teknikleri kullanırken hem simetrik şifrelemenin yapısının anlaşılabilmesi hem de olan şifreleme tekniklerinden farklı bir şifreleme tekniğinin olması amacı ile oluşturulmuştur. Bu şifreleme tekniği veri güvenliği sağlaması amacı ile oluşturulacak olan güvenlik protokolü yapısında diğer şifreleme teknikleriyle birlikte uyumlu bir şekilde kullanılabilir.

Açık Metnin Şifrelenmesi

Şifreli Metnin Deşifrelenmesi

2.4.2. Kaba Kuvvet Saldırısı (Brute Force Attack)

Bilgisayar bilimlerinde bir metnin içerisinde başka bir metnin aranması için kullanılan en ilkel ve dolayısıyla en düşük performanslı arama algoritmasıdır. Brute Force, genellikle bir parolayı (veya anahtarı) ele geçirmek için yapılan bir çeşit dijital ve kriptografi saldırısıdır. Brute Force tekniğinde elde herhangi bir bilgi bulunmuyor olmasına karşın belli şifreler denenerek doğru şifreye ulaşmaya çalışılır.

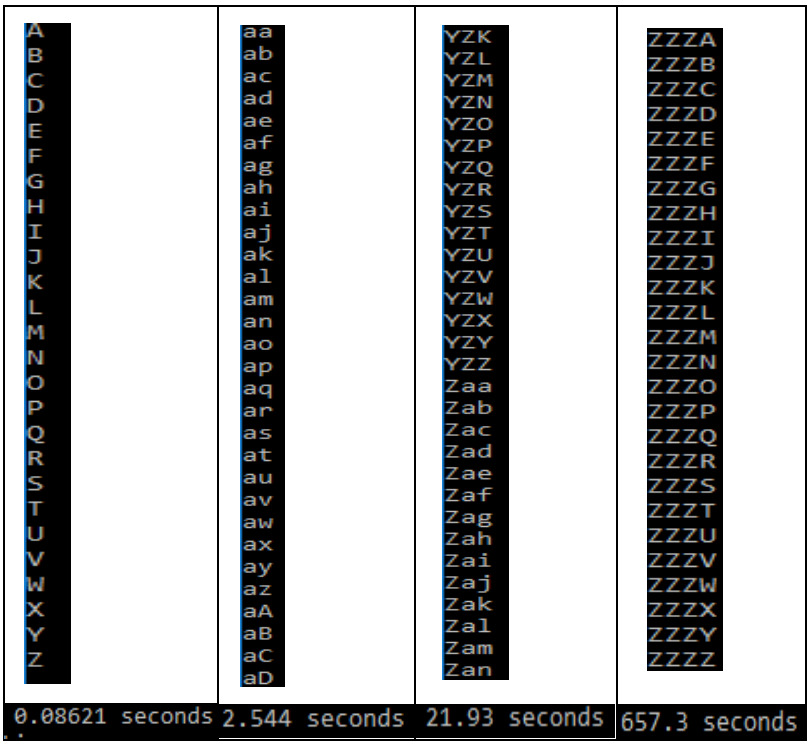
Bu saldırı yönteminde genellikle insanların sıklıkla kullandığı 123456789 ve 987654321 gibi basit şifreler başta olmak üzere birçok şifreden oluşan bir liste hazırlanır. Daha sonra meydana getirilen yardımcı bir yazılım yardımıyla veya el yordamıyla bu şifreler parolası bulunmak istenen hesaba giriş yapabilmek için tekrar tekrar denenir. Yazılım, doğru şifre bulunduğu anda bir sinyal vererek işlemi durdurur.

Brute Force garanti işe yarayan bir yöntem olmadığı gibi aynı zamanda yoğun zaman isteyen bir saldırı türüdür. Çünkü eldeki liste ne kadar büyük olursa bu bir avantaj olacak olmasına karşın denenecek şifreler de o kadar olacak demektir. Bu şartlar altında Brute Force ile başarıya ulaşmak saatler, aylar ve hatta yılları bile alabilmektedir.

```
static const char words[] =
"abcdefghijklmnopqrstuvwxyz"
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
//"0123456789";
```

```
static const int wordSize = sizeof(words) - 1;
```

```
for ( i=0; i<wordSize; i+=1 )
{       str[index] = words[i];
        if ( index==maxSize-1) printf("%s\n", str);
        else bruteImplament(str, index + 1, maxSize);
}
```

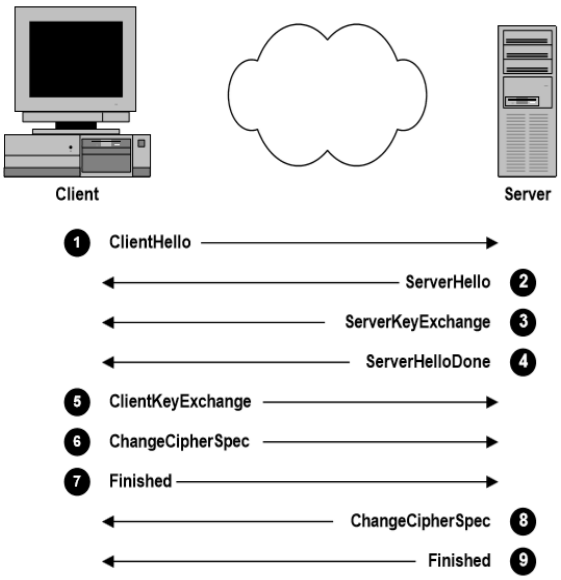


2.4.3. SSL Protokolü

2.4.3.1. Handshake Protokol (El sıkışma Protokolü)

Bir tarayıcı SSL ile korunan bir sunucuya bağlanmak istediğinde, sunucu ve tarayıcı arasında "SSL Handshake" denilen bir süreç başlatılmış olur. Bu süreç kullanıcılardan soyutlanmış anlık bir süreçtir. Bu süreçte üç adet anahtar kullanılmaktadır ve bunlar; public, private ve simetrik oturum anahtarıdır. Asimetrik algoritmanın işlem gücünün maliyetli oluşu, sadece simetrik oturum anahtarının paylaşımında kullanılmasına sebep olmuştur. Bu aşamadan sonra güvenlik simetrik oturum anahtarı ile sağlanmaktadır.

- Tarayıcı SSL ile güvenliği sağlanan bir web sunucusuna (https) bağlanır ve sunucudan kendini tanıtmasını ister.
- Sunucu; sahip olduğu SSL sertifikanın kopyasını ve açık anahtarını gönderir.
- Browser gelen sertifika için zamanının geçmediğini ve geçerli bir sertifika olduğu kontrollerini yapar. Tarayıcı sertifikanın bağlandığı siteye ait olduğunu teyit ederse, simetrik oturum anahtarını sunucunun public anahtarı ile şifreleyip gönderir.
- Sunucu şifreyi çözerek, simetrik oturum anahtarını elde ettiğine dair acknowledge bilgisini simetrik oturum anahtarı ile şifreleyerek tarayıcıya gönderir.
- Bu aşamadan sonra sunucu ve istemci iletilecek mesajı şifrelenmiş olarak gönderebilir.



SSL handshake protokolü, sertifika transferi ve oturum anahtarlarının tanıtılması için, istemci-hizmet biriminin her birinin hangi şifre takımlarını karşılıklı kimlik tanımlamada kullanacağını belirler.

SSL TCP protokolüne ihtiyaç duyar. Dolayısıyla SSL iletişiminden bahsedebilmek için öncelikle TCP 3-way handshake ile TCP bağlantının kurulması gereklidir. Paket dosyasındaki ilk 3 paket 3-way handshake mekanizmasına aittir ve TCP bağlantı kurulduktan sonraki aşama SSL Handshake'tir.

Şifre takımı; DES, DSA, KEA, MD5, SHA-1, RC2, RC4, RSA, RSA SKIPJACK, Triple DES ve Key Exchange algoritmalarını tanır.

Handshake protokolü her mesaj tür, uzunluk ve içerik bilgilerini içerir. Dört aşamadan oluşmaktadır. Birinci aşama güvenlik yetenekleri belirlenmesi (client hello ve server hello), ikinci aşama sunucu kimlik doğrulama ve anahtar değişimi (Sertifika, RSA, Diffie Hellman), üçüncü aşama istemci kimlik doğrulama ve anahtar değişimi (anahtar değişimi, sertifikayı doğrulama), son aşamada güvenli bağlantı oluşturmayı tamamlama (change cipher spec)'dır.

2.4.3.1.1. Handshake Protokol Aşamaları

Client Hello Aşaması

İstemci, SSL iletişimi kurma isteğini ClientHello mesajı göndererek belirtebilmektedir. Bu mesajın içine desteklediği SSL protokol versiyonu, şifreleme mekanizmaları (Cipher Suites olarak gösterilen alan, key exchange + cipher + hash algoritmaları bütünü), destekleyebildiği sıkıştırma yöntemi, SessionID ve RandomNumber denen ve 32 byte'lık bir değer (bu değer şifreleme anahtarı üretiminde kullanılır, ilk 4 byte'ı tarih ve saat bilgisini taşır) yerleştirir.

Server Hello Aşaması

Server Hello, Certificateve Server Hello Done mesajları paket dosyasındadır. Sunucu Client Hello mesajını aldığında Server Hello mesajıyla yanıt verir. İstemcinin Client Hello mesajından destekleyebildiği protokolleri,mekanizmaları öğrenen sunucu, SSL iletişiminde kullanılacak şifreleme mekanizmasını, SSL versiyonu, SessionID numarasını, sıkıştırma metodunu belirler ve yine şifreleme(simetrik şifreleme) anahtarının oluşturulmasında kullanılmak üzere ürettiği randomNumber değerini de bu mesaj içine yerleştirir.

Certificate Aşaması

Sunucu public sertifikasını (Public Key) bu mesajla gönderebilmektedir. Public sertifika CA (Certificate Authority) private sertifikası tarafından imzalandığından ve CA public sertifikasıyla bu imza doğrulanabildiğinden, sunucu sertifikasının güvenirliği kontrol edilebilmektedir. Kontrol edilemezse sertifikanın güvenilir olmadığına dair kullanıcı uyarılır.

Server Hello Done Aşaması

Bu mesaj anlaşmanın ilk fazının bittiğini belirtir ve istemci handshake'i tamamlamak üzere ClientKeyExchange, ChangeCipherSpec ve Finished mesajlarını gönderebilir.

ClientKeyExchange Aşaması

Bu aşamada istemci daha önce değiş-tokuş edilen randomNumber'lar ile birlikte kullanılarak simetrik key (anahtar) üretiminde kullanılacak 46 byte uzunluğunda "pre-master secret" da denen gelişigüzel bir numara üretir ve bu numarayı, MAC (Message Authentication Code) değeriyle birlikte sunucunun daha önce "Certificate" mesajıyla kendisine gönderdiği public sertifikasını kullanarak şifrelenebilmektedir. Bu mesajın yapısı ve içeriği, uzunluğu da kullanılan key-exchange algoritmasına göre değişebilmektedir.

"Certificate" mesajında gönderilen public sertifikayla şifrelenen bu değeri deşifre edebilecek olan da sadece bu public sertifikanın private sertifikasına sahip olan uçtur (Private Key). Şifreleme işleminde anahtar bu sertifika değil; sertifika aslında şifreleme işleminde kullanılacak anahtarı üretmede kullanılan "ön anahtar"dır. Asıl anahtar istemci ve sunucu arasında karşılıklı veya tek taraflı gönderilmemektedir. Anahtarı, daha önce iletilen bilgiler (randomNumber) ışığında istemci ve sunucu (pre-master secret ı da kullanarak) üretmektedir ve sonuç olarak ayrı ayrı üretilen bu anahtar aslında aynı anahtarı(simetrik) ifade etmektedir.

ChangeCipherSpec Aşaması

İstemci bu mesajla sunucuya bundan sonraki iletişimde üzerinde mutabık olunan protokoller, algoritmalar, anahtarla SSL protokolünü kullanma isteğini bildirir.

Finished Aşaması

ChangeCipherSpec mesajının gönderilmesiyle, istemci ve sunucu kendilerinden gelecek mesajların simetrik anahtarla şifreleneceği bilgisini içeren bir mesaj gönderir. Finished mesajı şifrelenen ilk mesajdır ve bundan sonraki iletişim de şifrelenir. Şifrelenen veri ise daha önceki SSL handshake mesajlarının hashi ve gönderen tarafın istemci veya sunucu olduğunu belirten bir başka değerdir.

TCP 3-way handshake'ler TCP bağlantı, SSL Handshake'le de güvenli iletişim kanalı kurulduktan sonraki iletişim, gönderilen Request ve Response'lar aynıdır,hiç bir farkı yoktur.

2.4.3.2. Alert Protokol (Uyarı Protokolü)

SSL kayıt katmanı tarafından desteklenen içerik türlerinden biri uyarı türüdür. Uyarı mesajları, mesajın ciddiyetini ve uyarının açıklamasını iletir. Ölümcül seviyeli uyarı mesajları, bağlantının derhal sonlandırılmasına neden olur.

Bu durumda, oturuma karşılık gelen diğer bağlantılar devam edebilir, ancak başarısız olan oturumun yeni bağlantılar kurmak için kullanılmasını engelleyen oturum tanımlayıcısının geçersiz kılınması gerekir. Diğer mesajlarda olduğu gibi, geçerli mesajda belirtildiği gibi uyarı mesajları şifrelenir ve sıkıştırılır.

2.4.3.2.1. Kapanış Uyarıları (Closure Alerts)

İstemci ve sunucu, bir kesme saldırısını önlemek için bağlantının sona erdiği bilgisini paylaşmalıdır. Her iki taraf da kapanış mesajı değişimini başlatabilir.

close_notify: Bu mesaj alıcıya gönderenin bu bağlantıda daha fazla mesaj göndermeyeceğini bildirir. Herhangi bir bağlantı düzgün kapatılmadan sonlandırılırsa, uyarı uyarıya eşit seviyedeki mesajları bildirmeden oturum önemsizleşir. Her iki taraf da close_notify uyarısı göndererek kapanış başlatabilir. Bir kapatma uyarısından sonra alınan veriler göz ardı edilir.

Her bir taraf, bağlantının yazma tarafını kapatmadan önce bir close_notify uyarısı göndermelidir. Karşı tarafın kendi tarafından bir close_notify uyarısı ile yanıt vermesi ve bekleyen tüm yazıları atmadan hemen bağlantıyı kapatması gerekir. Bağlantının okunan tarafını kapatmadan önce kapanış başlatıcısının yanıt veren close_notify uyarısını beklemesine gerek yoktur. Not: Bir bağlantının kapatılmasının taşımayı yok etmeden önce bekleyen verileri güvenilir bir şekilde sağladığı kabul edilir.

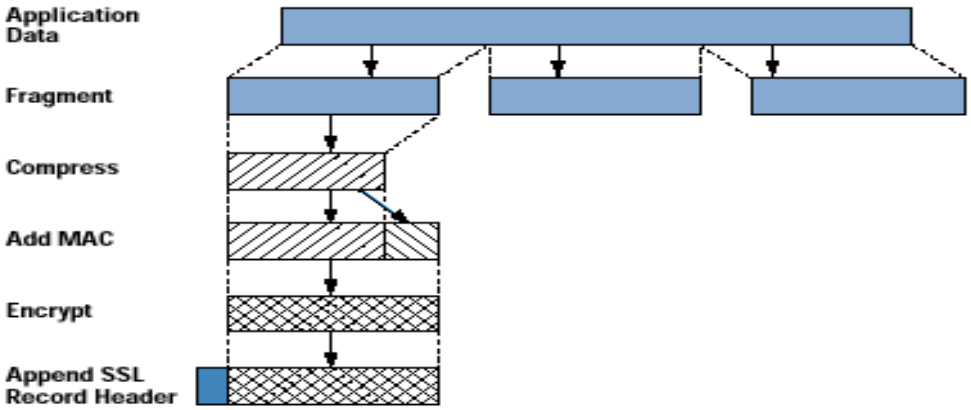
2.4.3.2.2. Hata Uyarıları (Error Alerts)

SSL anlaşması protokolünde hata işleme çok basittir. Bir hata tespit edildiğinde, tespit eden taraf diğer tarafa bir mesaj gönderir. Ölümcül bir uyarı mesajı alındığında veya alındığında, her iki taraf da bağlantıyı derhal kapatır. Sunucular ve istemciler, başarısız bir bağlantıyla ilişkili tüm oturum tanımlayıcılarını, anahtarlarını ve sırlarını unutmak zorundadır.

2.4.3.3 Record Protocol (Kayıt Protokolü)

Kayıt protokolü bağlantılar için gizlilik ve mesaj bütünlüğü sağlamaktadır. Gizliliği, uygulama verisini şifreleyerek, mesaj bütünlüğünü ise mesaj doğrulama kodu (MAC) kullanarak gerçekleştirir.

Kayıt Protokolü iletilmek üzere bir uygulama mesajını alır, bu mesaj boyutu belirli olan bloklara (2^{14} bytes) bölünür. Sonra bloklara isteğe bağlı olarak sıkıştırma (compression) uygulanır. Simetrik anahtar ile mesaj doğrulama kodu (MAC) hesaplanır ve sıkıştırılmış veriye eklenir. MAC'li ve sıkıştırılmış veri bloğu simetrik (AES, DES, Fortezza) olarak şifrelenir ve SSL Record Header bloğa eklenerek SSL Kayıt İşlemi (SSL Record Operation) tamamlanır.



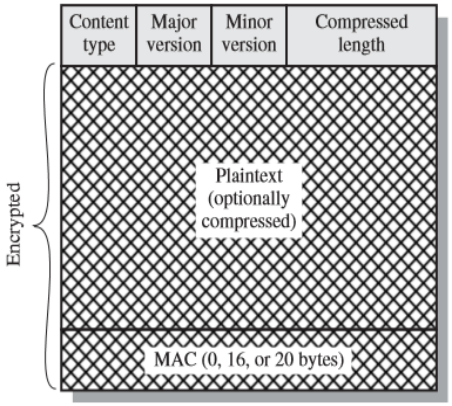
SSL Kayıt Protokolü işleminin son adımı,bir header(başlık) hazırlamaktır ve bu header aşağıdaki alanlardan oluşmaktadır.

- Content Type (İçerik Türü)(8 bit): Ekteki parçayı işlemek için kullanılan üst katman protokolü. Tanımlı olan içerik türleri change cipher spec, alert, handshake ve application data'dır.

- Major Version (Ana versiyon) (8 bit): Kullanılan SSL'in ana sürümünü gösterir. Örneğin SSLv3 için değer 3'tür.

- Minnor Version (Küçük versiyon)(8 bit): Kullanılan minnor(küçük) versiyonu gösterir. Örneğin SSLv3 için değer 0'dır.

- Compressed Lenght (Sıkıştırılmış uzunluk) (16 bit): Düz metnin bayt cinsinden uzunluğu eğer sıkıştırma kullanılıyorsa sıkıştırılmış parçanın uzunluğudur.



SSL Record Protokol Formatı

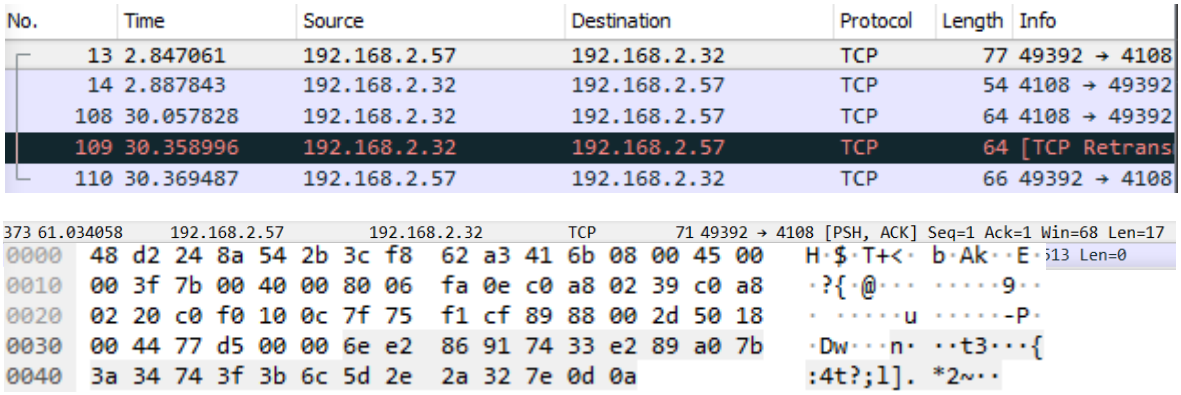
2.4.4. WireShark Testi

Wireshark network trafiğinin, bir grafik arayüz üzerinden izlenmesini sağlayan, pek çok zaman hayat kurtarancı öneme sahip bir programdır. Uygulamanın kurulu olduğu bilgisayar üzerinden anlık network trafiği izlenebileceği gibi, Wireshark daha önce kaydedilmiş dosyaların incelenmesi amacı ile de kullanılabilir.

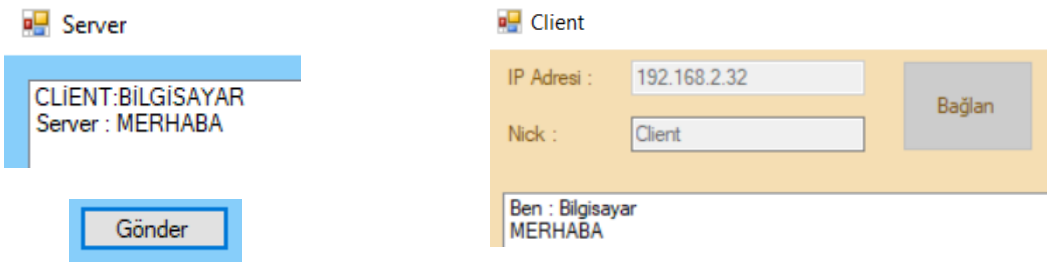
2.4.4.1. Wireshark'ın Özellikleri

- Önceki ismi Ethereal'dir ve GPL lisansıyla dağıtılır.
- Unix/Linux ve Windows işletim sistemlerinde kullanılabilir.
- Yerel ağdaki paketleri tutar ve ayrıntılı olarak protokol bilgileri de dahil olmak üzere bizlere görüntüler. Yakalamış olduğu paketleri kaydedebilir.
- Oluşturacağınız kriterlere göre paket arama ve filtreleme yapabilirsiniz.
- Filtreleme sayesinde paketleri renklere ayırır ve kategorileştirir.
- İstatistikleri bizim yapacağımız ayarlar ile bizlere sunar.
- Protokoller için şifre çözme desteği vardır.

Aynı ağ üzerinde bağlı olan 2 bilgisayar üzerindeki iletişim



Proje tasarımında gerçeklemeye çalışılan şifreleme algoritmalarının testlerini bir sunucu-istemci uygulaması üzerinden testi yapıldı. Bu test üzerinde ilk önce server tarafında açılan port ile dinleme işlemi başladı. Bu kısımda soket programlamada thread yapısının kullanımı söz konusudur. Daha sonra istemci tarafından belirlenen ip numarası ile bağlantı sağlanmış oldu.



3.1. Şifreleme Algoritmalarının Kütüphaneleştirilmesi

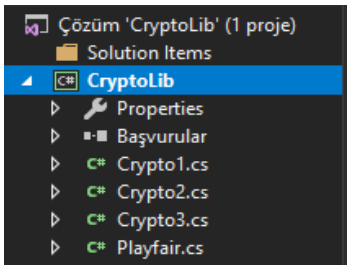
Visual C# geliştirme aracını kullanarak tasarladığımız sınıfları nasıl DLL şeklinde derlenebileceği ve nasıl kullanılması gerektiği öğrenilmelidir.

Günümüzde artan sektörel ihtiyaçlardan dolayı yazacağımız kod satırı sayısı günden güne artmaktadır. Temelde aynı şeylerin yapılmasına rağmen yapılan programlardan beklenmektedir. Bazı problemler tek bir bilgisayardan halledilemez hale geldiği için, dağıtık yapıdaki sistemlerdeki bilgisayarlarda çalışan programlar günümüzde ihtiyacı en çok hissedilen programlar olmuştur. İnternet üzerinde çalışan programlar dağıtık programlamaya örnektir.

Yazılım alanındaki ihtiyaçlar arttıkça kod yazan programcının da işi zorlaşmaktadır. Programcı daha fazla kod yazma durumunu ortadan kaldırmak için birçok kez tekrar edebilen yapıları kütüphane ortamına dahil ederek okunurluğu olumlu yönde etkilemektedir. Belirtilen bu nedenlerden ötürü yazacağımız kodları azaltmak için bir takım teknolojiler geliştirilmiştir. Amaç, önceden yazılmış kodları bir şekilde kaynak kodunu gizleyerek hem başkasının kullanımına açmak hem de tekrar kullanmasıdır.

Bu durumda bir takım engellerin çıkması mümkündür. Bunlardan en önemlisi şudur: Farklı yapılara sahip hatta farklı veri türlerine sahip olan dillerde yazılmış olan kaynak kodları başka bir dilde nasıl kullanılabileceğidir. Bu ilk başta zor görülsede, eğer sistematik bir standartlaşmaya gidilirse ve her programlama dili bu standarda uyarsa sorunun çözülebilir.

Bu yüzden COM ve CORBA gibi teknolojiler geliştirilmiştir. Bu teknolojiler herhangi bir dil ile yazılmış olan kodların binary (mikroişlemcinin anlayacağı seviye) düzeyde birbirlerini anlayacak standartları içermektedir. COM programlamada ise çeşitli diller ile yazılmış kodların bir takım işlemler sonucu kodların COM standartlarına uygun hale getirilmesidir. Uygun hale getirilmiş kodlar, her paltformda farklı dosya formatlarında saklanır.



VS Kütüphane Örneği

En çok duyduğumuz formatlardan biri de DLL lerdir. Bir DLL, tekrar kullanılabilen program parçalarından oluşmaktadır. DLL denilen program parçacıkları tek başlarına bir iş yapmamasına rağmen, yazılabilen çalıştırılabilir (exe gibi) programlarda onları kullanarak işlemler kolaylaştırılabilir. Herhangi bir dilde COM programlama yapabilmek için o dile ait derleyicinin COM yada benzeri standarları destekliyor olması gerekir. Günümüzde en çok kullanılan iki standart COM ve CORBA standartlarıdır.

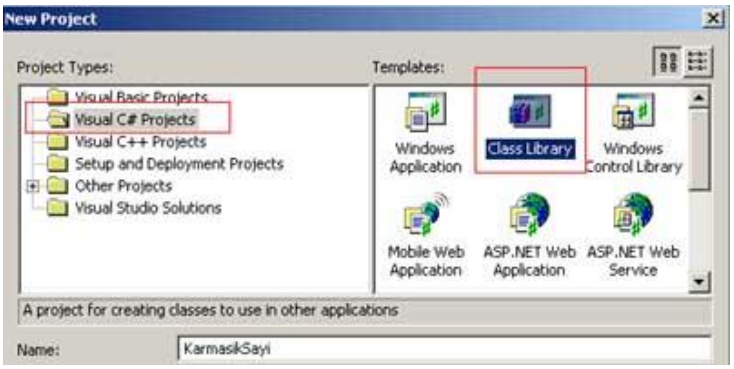
Burada .NET ortamında kullanılan herhangi bir dilde kullanmak üzere bir dll tasarımı görülmektedir. DLL'i oluşturmak için Visual Studio.NET ortamını tercih edilebilir. Yazacağımız dll in içerdiği standart, .NET ortamının çalışma zamanı bilgilerini içeren ve yöneten CLR(Common Language Runtime) olduğu için dll olarak hazılanabilen kütüphaneyi ancak CLR'yi destekleyen bir dilde kullanmamız mümkün olacaktır. Yani .NET için component geliştirilme durumunu belirtir. CLR'yi desteklemeyen bir dilde kullanabilmesi için kütüphaneyi COM standarlarına uygun hale getirmek gerekecektir.

.NET ortamındaki CLR birimini daha iyi kavranması gerekmektedir. VB.NET'te Integer türünden tam sayıları belirten temel bir veri türü vardır. C#'ta bunun karşılığı ise int türüdür. Aslında çalışma zamanında bu iki veri türüde tek bir veri türüne denk gelmektedir. Bu veri türüde System isimalanında bulunan Int32(System.Int32) veri türüdür. Yani C#'ta kullandığımız int veri türü ile VB.NET te kullandığımız veri türü aslında çalışma zamanında tek bir veri türü olarak algılanır. Bu eşleştirme işini sağlayan ise .NET'in CLR denilen birimidir.

Görüldüğü gibi COM programlamanın getirdiği en büyük özelliklerden biri olan tür uyumlaştırması CLR sayesinde mümkün hale gelmektedir. O halde .NET için component geliştirilmesi hiç umulmadığı kadar kolay durumdadır.

3.1.1. Kütüphane Sınıfını Tasarlamak

Visual Studio.NET'i çalıştırdıktan sonra açılabilecek proje tipi aşağıdaki şekilden de görüldüğü gibi "Class Library" olacaktır. Bu ekrana ulaşmak için File->New->Project menüsünü kullanılabilir. Proje tipi olarak Visual C# template olarak ise Class Library seçtikten sonra projeye uygun isim verilip tasarım işlemi tamamlanabilir.



Kütüphane Projesi Açma Adımı

Temel girdi/çıktı işlemleri için System isim alanına referans vermek gerekir. Sınıfımıza diğer uygulamalarda kullanmak için bir isim alanı eklenebilir. Complex isimli veri türümüzün iki tane private veri elemanı olacak. Bunlar "re" ve "im" dir ve complex sayının reel ve sanal kısımlarını temsil edecektir. Bazı üye fonksiyonları sınıfın kurucu(constructor) üye fonksiyonları olduğu gibi bazıları ise override (aşırı yüklenmiş) edilmiş fonksiyonlarıdır.

Görüldüğü gibi DLL oluşturmak için yazılan kaynak kodun main işlevi olmayan bir uygulamadan farkı yoktur. Burda en önemli olan nokta sınıfın bir isimalanı içine alarak başka modüllerden kullanımını sağlamaktır.

Sınıfı yazdıktan sonra Build -> Build Solution menüsünü seçerek DLL dosyası tasarlanabilir. Eğer komut satırı derleyici kullanılıyorsa dll kütüphanesi oluşturmak için komut satırına "csc /target:library crypto.cs" ifadesi yazılabilir. Bu işlem ile derleyiciye çalıştırılabilir bir dosya yerine dll şeklinde bir kütüphane oluşturulacağı belirtilir. Eğer komut satırı derleyicisi kullanılırsa komut satırı kullanılarak dll dosyası tasarlanabilir.

Build işlemi gerçekleştikten sonra projenin bulunduğu dizin içinde bulunan Bin->Debug klasörünün içinde crypto.dll isimli bir dosya oluşabilir. Bu dosyayı kullanarak dll içinde bulunan Complex sınıfına istenilen .NET dili ile ulaşabilmek mümkün hale gelir.

3.2. C#'da Proje oluşturma ve Test İşlemleri

İlk olarak, bir C# uygulaması projesi tasarımında proje türü için ihtiyaca göre şablon dosyaları eklenmelidir. Bu işlem için **yeni bir proje oluşturma** penceresine girilerek veya **windows** platform listesinden proje açılabilir. Tüm dil ve platform filtreleri uyguladıktan sonra **konsol uygulaması (.NET Core)** şablonu seçilir.

İçinde **yeni projenizi yapılandırın** penceresinde yazın veya içinde **proje adı** kutusu ile proje tasarımı gerçeklenebilir. Uygulama içerisinde **F5**, programın hata ayıklama modunda çalıştırabilmesini sağlar. Ancak, konsol penceresinde kapatılmadan önce yalnızca bir süre görünürlüğü söz konusudur. Bu davranışı nedeniyle gerçekleşir Main yöntemi, tek bir deyim yürütüldükten sonra ve uygulama sona şekilde sonlandırır.

3.2.1. Simetrik Şifreleme Algoritması - 1 İmplementasyonu

Şifreleme metotlarını bir kütüphane içerisine dahil etmek için, uygulamadan bağımsız şekilde çalışması gerekmektedir. Proje içerisinde bulunan şifreleme algoritmalarını belirlenen ortak programlama dili üzerinde çalışmalıdır.

Simetrik Şifreleme Algoritması -1'de Kullanılan Fonksiyonlar

```
public void Encrypt(string file_path, string cipher_path, string key)
public byte[] EncryptToByteArray(byte[] mainByteArray, string key)
private byte[] Encryption(byte[] mess, byte[] key)
private byte[] EncryptBlock(byte[] mess, byte[] key)
public void Decrypt(string cipher_path, string output_path, string key)
public byte[] DecryptToByteArray(byte[] cipherByteArray, string key)
private byte[] Decryption(byte[] cipher, byte[] key)
private byte[] DecryptBlock(byte[] cipher, byte[] key)
```

Belirtilen algoritmanın implementasyon testleri için belirli parametrelerle gönderimin yapılması gereklidir. Açılan örnek "ConsoleApp1" uygulaması üzerinden tasarlanılan kütüphane metotlarını kullanarak testleri tamamlandı. Programın testleri proje üzerinden konsol çıktısı ile kontrol edildi

```
//Crypto1 sifreleyicim = new Crypto1();
String file_path = "C:\\Users\\Hknaksoyy\\Desktop\\plainText.jpg";
//String cipher_path = "C:\\Users\\Hknaksoyy\\Desktop\\deneme.jpg";
String key = "asdfgytasdfgytrfghtuytorfghtuyto";
//sifreleyicim.Encrypt(file_path, cipher_path, key);
String decrypt_path = "C:\\Users\\Hknaksoyy\\Desktop\\decodedText.jpg";

byte[] plainText = File.ReadAllBytes(file_path);
Crypto1 sifreliMetin = new Crypto1();
byte[] cipherText = sifreliMetin.EncryptToByteArray(plainText, key);
byte[] acikMetinbyte = sifreliMetin.DecryptToByteArray(cipherText, key);
File.WriteAllBytes(decrypt_path, acikMetinbyte);
```

.

3.2.2. Simetrik Şifreleme Algoritması - 2 İmplementasyonu

Karakterlerin ASCII değerlerini kullanılarak yapılan şifreleme yönteminde birden çok parametrenin sunucu-istemci arasında bilinmesi gerekmektedir. Bunlar; şifreleme yaparken tasarlanılan List <int>, int [ ] ve anahtar değeridir. Bunların veriyle bağlantılı olması iletişimde güvenliği arttırabilmektedir fakat hızı kötü yönde etkilemektedir.

Bu nedenle kullanılacak kütüphane içerisinde şifreleme yaparken üretilen list <int> değerleri, karşı tarafa gönderilen binary değerlerle üretilmesi için güncelleme yapıldı.

```
for (int i = 0; i < input.Length / 8; i++)
{
     if (j > current_length) j = 0;
     string gelenDeger = input.Substring(i * 8, 8);
     int sonuc = 0;
     for (int artma = 0; artma < gelenDeger.Length; artma++)
     {
          try
          {
               if (Int32.Parse(gelenDeger[artma].ToString()) == 1)
                  sonuc += (int)Math.Pow(2, gelenDeger.Length - 1 - artma);
               else if (Int32.Parse(gelenDeger[artma].ToString()) > 1)
                  throw new Exception("Invalid!");
          }
          catch
          {
               throw new Exception("Invalid!");
          }
}
```

Simetrik Şifreleme Algoritması - 2'de Kullanılan Fonksiyonlar

```
public String Encrypt(String input)
public String Decrypt(String input)
public String DecryptKey(String input, String key)
public String DecryptKeyArrayList(String input, String key, int [] array,
List<int> sayilar)
```

Belirtilen algoritmanın implementasyon testleri için belirli parametrelerle gönderimin yapılması gereklidir. Açılan örnek "ConsoleApp1" uygulaması üzerinden tasarlanılan kütüphane metotlarını kullanarak testleri tamamlandı.

```
Crypto2 sifreleyicim2 = new Crypto2();
String cipher = sifreleyicim2.Encrypt("DENEME");
Console.WriteLine(cipher);

Crypto2 desifreleyicim2 = new Crypto2();
String plain = desifreleyicim2.Decrypt(cipher);
Console.WriteLine(plain);
```

3.2.3. Simetrik Şifreleme Algoritması - 3 İmplementasyonu

Algoritmada kullanılan tablolarla şifreleme ve deşifreleme işlemi yapılırken güvenliği arttırmak için Playfair algoritması dahil edildi. Bu algoritma kütüphane içerisine eklenirken; karşı tarafı yollanacak olan veri gönderilmeden önce, belirlenen matrisi kullanarak şifreleme yapmaktadır.

```
..
Playfair sifrele = new Playfair();
String playCipherText = sifrele.PlayFairSifre(cipherText);
return playCipherText;
..
Playfair sifrecoz = new Playfair();
String playPlainText = sifrecoz.PlayFairDesifre(ciphertext);
```

Kütüphane içerisine bulunan **Crypto3** sınıfı içerisinde tanımlanan **Playfair** sınıfı ile kullanım sağlanır. Genel olarak yapılan blok şifreleme algoritmasında parçalanan kelime grupları arasında bir bağlantı vardır. Bu bağlantı şifrelenecek verinin karakter uzunluğuna göre değişiklik göstermektedir. Playfair algoritması ile şifrelenen verinin, deşifrelenmesi sonucu ortaya çıkan verideki "**X**" karakterleri doldurma (**padding**) şeklinde tanımlandığı için bu karakterler çıkarıldıktan sonra açık metin elde edilebilmektedir.

Simetrik Şifreleme Algoritması – 3'te Kullanılan Fonksiyonlar

```
public String Encrypto(String plaintext)
public String Decrypto(String ciphertext)
private void DictionaryCift()
private void DictionaryTek()
private void Dictionary2us()
private String Encrypto_Odd(string karakter)
private String Encrypto_Even(string karakter)
private String Encrypto_Pow(string karakter)
private String Decrypto_Odd(string karakter)
private String Decrypto_Even(string karakter)
private String Decrypto_Pow(string karakter)
```

Belirtilen algoritmanın implementasyon testleri için belirli parametrelerle gönderimin yapılması gereklidir. Açılan örnek "**ConsoleApp1**" uygulaması üzerinden tasarlanılan kütüphane metotlarını kullanarak testleri tamamlandı.

```
Playfair sifreleyicimPlay = new Playfair();
Console.WriteLine(sifreleyicimPlay.PlayFairSifre("AAAAA"));

Playfair desifreleyicimPlay = new Playfair();
Console.WriteLine(desifreleyicimPlay.PlayFairDesifre("UDUDUD"));

Crypto3 sifreleyicim3 = new Crypto3();
String sonuc = sifreleyicim3.Encrypto("DENEME");
Console.WriteLine(sonuc);

Crypto3 desifreleyicim3 = new Crypto3();
Console.WriteLine(desifreleyicim3.Decrypto("WHAHSA"));
```

3.2.3.1. Playfair (blok şifreleme) Algoritmasının İmplementasyonu

Playfair bir simetrik algoritmadır. Genel olarak blok şifreleme özelliği kullanılan yapı; şifrelenecek metni parçalayarak tanımlanan tablonun kullanımıyla şifrelenme işlemi yapılır. Belirtilen matris kullanılan karakter sayısı ile bağlantılı bir şekilde tasarlanıp, belirli kural çerçevesinde metin şifrelenmektedir.

Bu kurallar;

- Art arda gelen aynı karakterlerin arasına "X" eklenerek şifrelenmesi
- Eğer şifrelenecek metin eklenebilecek "X" karakteriyle birlikte tek sayıda karakter içeriyorsa, padding (doldurma) yapılarak karakter sayısı çifte tamamlanır.
- Belirtilen işlemden sonra; İkili gruplara parçalanan harf grupları aynı satırda ise her harf için aynı satırda kendinden sonra gelen karakter (sağındaki eleman) seçilir.
- İkili gruptaki elemanlar aynı sütunda ise her harf için aynı sütunda bulunan bir sonraki karakter seçilir. Belirtilen bu koşul dışında ise her harf için bulunduğu satır ve sütun kesişim noktalarındaki karakter baz alınarak şifreleme işlemi yapılabilmektedir.

```
char[,] matris = { {'A','B','C','D','E'},
                   {'F','G','H','I','J'},
                   {'K','L','M','N','O'},
                   {'P','Q','R','S','T'},
                   {'U','V','W','X','Y'} };
```

Şifreleme Örneği:

Açık Metin : **THETABLETENNIS**

İlk adım → **TH – ET – AB – LE – TE – NN – IS**
İkinci adım → **TH – ET – AB – LE – TE – NX – NI – S**
Üçüncü adım → **TH – ET – AB – LE – TE – NX – NI – SX**
Şifreleme adımı → **JR – JY – BC – BO – YJ – SD – SN – XD**

Kapalı Metin : **JRYHBCBOYJSDSNXD**

İlk adım → **JR – JY – BC – BO – YJ – SD – SN – XD**
Deşifreleme adımı → **TH – ET – AB – LE – TE – NX – NI – SX**

Açık Metin : **THETABLETENXNISX**

Playfair'da şifrelenecek metin içerisinde padding (doldurma) için kullanılan karakterin bulunmaması gerekir. Bunun nedeni; deşifreleme anında çıkarılan "X" karakteri ile açık metin bozulmadan geri dönebilmesi sağlanabilmektedir.

Anahtar uzayı; toplamda matriste belirtilen eleman kadar n! kadar anahtar seçimi mümkündür. Kayıplı şifrelemenin yaşanmaması için matris boyutu kadar anahtar seçilip, tablo üzerinde kullanılması gerekmektedir.

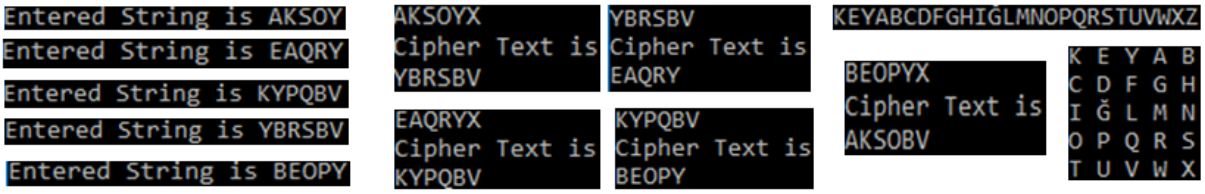
3.2.3.2. Playfair Algoritmasının Zayıflık Testi

Playfair algoritmasında sadece şifreleme yapılarak açık metin elde edilebilmektedir. Burada gizli olarak iletilmesi gereken veriyi karşıya gönderiminde yapılan şifreleme adımından sonra, iletilen veri üzerine tekrardan şifreleme işlemi uygulanabilmektedir.

Şifreli metni elde etmek istediğimizde; geriye dönüşün sağlıklı bir şekilde yapılabilmesi bir koşulu sağlanması gerekir. Bu koşul; veriyi şifrelerken kullanılan matrisin çift sayı boyutunda bulunmasıdır. Bunun sebebi; şifrelerken kullanılan algoritmada art arda gelen kelimeler eğer aynı satır ve aynı sütunda değilse, satır ve sütun kesişim noktalarındaki karakter baz alınarak şifreleme yapılmaktadır.

Bu durumda boyutu tek seçilen matriste geriye dönüş için yapılan şifreleme tekrarı, aynı sütun ve aynı satırda bulunan kelimeler için başarı sağlamaz. Belirtilen koşul sağlandığı taktirde yani yapılan şifreleme sayısı boyut kadar tekrarlandığında matris dairesel olduğu için geriye dönüş sağlanacaktır.

Verinin Deşifreleme Örneği



3.2.3.3. Simetrik Şifreleme Algoritması – 3'e Playfair Yapısının Eklenmesi

Algoritmanın ilk aşamasında kullanılan **Tek**, **Çift** ve **Üs** tablolarının iletişim sırasında her iki tarafa da iletilmesi bir sorundur. Bunu güvenli bir şekilde sağlayabilmemiz için kullanılabilecek Asimetrik şifreleme yapısı veya Kerbelos gibi bir yapının eklenmesi gerekir. Bu iletişimin maliyetini oldukça arttırabilmektedir.

Blok şifrelemesinde kullanılan playfair algoritmasında; verinin güvenli şekilde iletilmesi için belirlenen matris, bir anahtarı baz alarak tasarlanır. Bu şekilde şifrelenecek veri için anahtarın paylaşım sorunu tekrardan görülebilir. Bu nedenle şifrelenecek veri, statik belirlenen matrisin iletişim yapacak kişiler arasında bilindiği kabul edilir.

3.3. Soket Programlama

Soketler iki yönlü haberleşme aygıtlarıdır. Aynı makinedeki farklı süreçler arasında ya da uzak makinelerdeki süreçlerin haberleşmesinde kullanılabilir. Uzak makinelerdeki süreçler arasında da haberleşmeyi olanaklı kılan tek yöntemdir.

Soket oluşturulacağı zaman, üç parametrenin belirtilmesi gerekir: haberleşme stili, namespace ve protokol. Haberleşme stili soketin iletilen veriye nasıl davranacağını belirler. Aynı zamanda taraflar arasındaki paket iletiminin nasıl gerçekleştirileceğini belirler. Namespace, soket adreslerinin nasıl yazılacağını belirler.

Soket adresi soket bağlantısının bir ucunu işaretler. Örneğin lokal namespace için soket adresleri dosya isimleri iken, internet namespace için soket adresi ağa bağlı olan uzak makinenin IP adresi ve port numarasıdır. Son parametre olan protokol verinin nasıl iletileceğini belirler. Soket programlamada yer alan temel metodlar; socket, close, connect, bind, listen ve accept yapısıdır.

Soketleri oluşturma ve yok etme: socket ve close fonksiyonları soketleri oluşturur ve yok eder. Bir soket oluştururken üç parametre bildirilir, namespace, haberleşme stili, ve protokol. Soket ’in kullanımı sona erdiğinde ise close komutu ile yok edilir.

Bağlantının çağırılması: İki soket arasında bağlantının oluşturulabilmesi için client tarafında connect fonksiyonunun kullanılması gerekir. Bu fonksiyon hangi server soketin adres bilgisini belirtir. Client bağlantının ilklendirmesini gerçekleştirirken, server bağlantıları kabul etmek için beklemededir.

Lokal Soketler Aynı bilgisayardaki süreçleri bağlayan soketler, PF_LOCAL ve PF_UNIX sabitleri ile beraber lokal namespace’i kullanırlar. Bunlar lokal soketler yada UNIX domen soketleri olarak adlandırılırlar. Dosya isimleri ile belirtilen soket adresleri bağlantılar oluşturulacağı zaman kullanılırlar. Soketin ismi sockaddr_in yapısında belirtilir. Sun_family alanı AF_LOCAL olarak tanımlanır. Bu sabit değer soketin lokalliğini gösterir.

Herhangi bir dosya ismi soket oluşturmada kullanılabilir. Ancak sürecin dizine yazma hakkının olması gerekir. Bir sokete bağlanılabilmesi için, sürecin dosya için okuma hakkına sahip olması gerekir. Farklı bilgisayarlardaki süreçler aynı dosya sistemini paylaşsalar dahi, yalnızca aynı makinedeki süreçler lokal namespace soketleri yardımı ile aralarında haberleşebilir. Lokal namespace için kullanılabilecek protokol 0 değeri ile gösterilir.

Sunucu programı lokal namespace soketi oluşturur ve bağlantılar için dinlemeye başlar. Bir bağlantı talebi aldığında, bağlantıdan gelen mesajı okur ve bağlantı kapanıncaya kadar yazar. Eğer gelen mesajlardan biri “quit” dir, sunucu programı soketi kaldırır ve bağlantıyı sonlandırır. Socket-server programı soket dosyasının yerini parametre olarak komut satırından almaktadır. TCP soket haberleşme kullanarak basit bir istemci-sunucu uygulaması gerçekleştirilecektir.

3.3.1. C# 'da Multi Client Server Programlama

Multi client server; sunucudan (server) istediği zaman bilgi isteyebilen ve bu bilgileri kullanıp üzerinde işlem yapan istemci (client) sayısının birden çok olma durumunu belirtir. Sunucu programları bazı engellemelere rağmen birçok istemci ile bağlantı sağlayabilir.

```
serversock.Bind(ipep);          //ipep sunucunun bitiş noktasını belirtir.
serversock.Listen(10);
Socket client = serversock.Accept();
```

Yukarıda belirtilen kodun uygulanmasından sonra, sunucu bağlanmak için bir istemci beklemektedir. Bağlantı kurulduktan sonra bu istemci soketi kapanana kadar diğer istemciler sunucuya erişememektedir. Kısacası bu yapı bir istemci ile çalışmak anlamına gelir. İstemciler de bir seferde sadece bir kere bağlanabilmektedir. Bu yüzden istemciler sunucu kullanacakları zaman bu bir sorun haline gelebilmektedir.

Günümüzde genellikle daha çok tercih edilen sunucu, birçok istemciye aynı anda bağlantı sağlayan sunucu tipidir. Bunun için 2 yaklaşım kullanılabilir. Bunlar;

- Poll( ) / Select( ) Metodu
- Çok Parçacıklı (Multithreaded) Server

3.3.1.1 Poll( ) / Select( ) Metodu

Pool( )Metodu: .Net soket kütüphanesi, Unix soket kütüphanesinde bulunan ve engellenemeyen soket yöntemlerini içerir. Bu sayede programcılar Windows ortamında Unix ağ programlarına kolayca giriş sağlarlar. Poll( ) / Select( ) metodunun formatı aşağıdaki gibidir;

bool Poll (int microseconds, SelectMode mode);

Durum sonucu olarak eğer eylem tamamlanırsa **true**, fakat eylem gerçekleşmezse **false** döndürülür. **microseconds** Poll( ) metodunun belirtilen olaylar için bekleyeceği ve soketi izleyeceği süreyi belirtmektedir. **SelectMode** ise izlemek için kullanılacak eylem türünü belirtmektedir. SelectMode metoduna bağlı bazı fonksiyonların sonucunun true olma koşulları aşağıda verilmiştir.

SelectMode.SelectRead'in değeri aşağıdaki şartlarda Poll( ) metodunun true döndürmesine sebep olur;

- Eğer Accept( ) metodunun çağrısı başarılı olursa,
- Eğer soket üzerinde veri varsa ve
- Eğer bağlantı kapatıldıysa

SelectMode.SelectWrite'ın değerinin true döndürmesine sebep olacak koşullar ise aşağıda belirtilmiştir;

- Eğer bir Connect( ) metodu çağrısı başarılı olursa
- Ya da veri soket üzerinde gönderilebilirse

SelectMode.SelectError'ın değerinin true döndürme koşulları;

- Bir Connect( ) metodu başarısız olduysa,
- Ya da elimizde "out-of-band data" mevcut ise ve soketin OutOfBandInline özelliği henüz ayarlanmamış ise

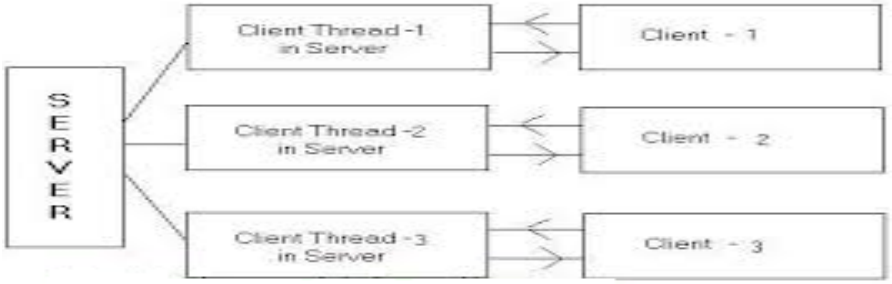
Select( ) Metodu: Select metodu bazı fonksiyonları engellemek için soketlerden bir ya da bir kaçını ortadan kaldırması amacıyla kullanılabilmektedir. Soketler okuma ve yazma için uygun hale geldikçe Select metodu hangilerinin kullanıma hazır olduğuna ve hangilerinin kullanılırsa engellenebileceğine karar vermektedir. Metodun formatı şöyledir;

Socket.Select (IList checkRead, IList checkWrite, IList checkError, int microseconds)

Ilist nesnesi ayrı ayrı, indeks değeri ile ulaşılabilecek nesne koleksiyonunu temsil eder. Eğer sunucu soketi gerçek veriyi alırsa, standart send() metoduyla kaynak istemciye yansıtır.

3.3.1.2. Çok Parçacıklı (Multithreaded) Server

Select( ) veya Poll( ) metodunu kullanmanın sakıncası kodla sarılı hale gelen birçok istemciyi idare etmektir. Programcılık mantığı açısından, farklı aralıklardan veri gönderen ve alan birçok istemciyi bağdaştırmak oldukça zordur. Ayrıca bu yaklaşımların etkisiyle zaman içinde paylaşılan engelleme yöntemleri ve istemcilerin sırasını beklemesi gerekir. Her istemci sunucuyla konuşabilmesi için özel bir kanala sahip olma durumu Multithread yapısıyla gerçeklenebilmektedir.



Çoklu İstemci Server Örneği

Multithreaded Server'da kilit konu, ana programda ana sunucu Soket nesnesi tasarlamaktır. Her istemci sunucuya bağlandığında, ana program bağlantıyı sağlamak için ayrı bir iş parçacığı (multithread) tasarlamaktadır. Bu konudaki ilave bilgiler için ThreadPool class'ının incelenmesi fayda sağlayacaktır.

4.1. Simetrik Şifrelemede Kullanılan Anahtarın Paylaşımı

Simetrik anahtar şifrelemesinde (AES, DES, DES3, vb) veri gizli bir anahtar tarafından şifrelenerek saklanır veya bir başkasına iletilir. Şifrelenmiş veri gizli anahtar kullanılarak deşifre edilir ve kullanılır. Bu sisteme simetrik denmesinin sebebi; verinin aynı anahtar kullanılarak hem şifrelenmesi hem de deşifre edilebilmesidir.

Veriyi güvenli saklama için etkili bir şekilde kullanılabilen simetrik şifrelemenin en önemli dezavantajı; veri iletimi senaryolarında gizli anahtarın paylaşılma problemidir. Gizli anahtarın veri iletiminin iki tarafına dağıtılma problemi, anahtarın kendisinin de güvenlik sebebiyle sık sık değiştirilmesi gerektiği için çok daha büyük bir problem haline gelmiştir.

Asimetrik anahtar şifrelemesinin çözdüğü en önemli problem tahmin edebileceği gibi simetrik anahtar şifrelemesinin çözemediği anahtar dağıtımı problemdir. Simetrik şifrelemenin aksine iki farklı anahtar açık(public)/kapalı(private) anahtar(key) kullanılır.

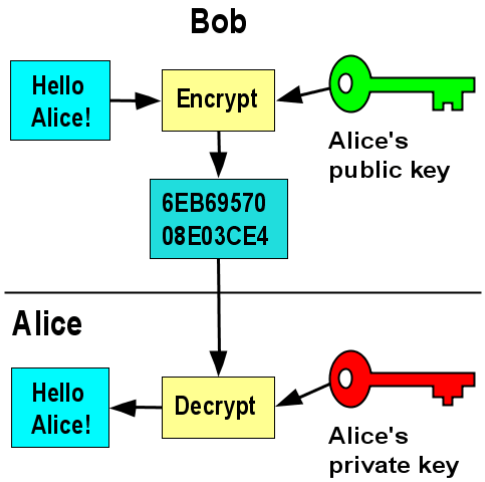
Açık anahtar ile şifrelenen veri kapalı anahtar ile, kapalı anahtar ile şifrelenen veri ise açık anahtar ile deşifre edilebilir. Adlarından da anlaşılabileceği gibi gizli anahtar kişiye özel gizli bir anahtar, açık anahtar ise herkese açık bir anahtardır. İletişime geçilmek istenen kişinin açık anahtarla şifrelenen veri sadece ilgili kişinin gizli anahtarıyla deşifre edilebileceği için güvenli bir şekilde ilgili kişiye iletilebilir. Bu yöntemlerden en bilinenleri RSA , Diffie-Helman, El-Gamal dır.

4.2. RSA Nedir?

Bir açık anahtarlı şifreleme yöntemi olan RSA, 1977 yılında Ron Rives, Adi Shamir ve Leonard Aldeman tarafından bulunmuştur. Güvenliği tam sayıları çarpanlarına ayırmanın algoritmik zorluğuna dayanan bir yöntemdir.

4.2.1. RSA'nın Özellikleri

- Asimetrik bir şifreleme algoritmasıdır. Simetrik şifrelemedeki gibi tek anahtar kullanılmasının yerine; biri gizli (Private Key) diğeri açık (Public Key) olmak üzere iki anahtar kullanılır.
- Güvenilirlik derecesi, şifrelemede kullanılan asal sayıların büyüklüğü ile orantılıdır.Özellikle çok kullanıcısı olan sistemlerde güvenli veri paylaşımına ve sayısal imza ile kimlik doğrulaması (authentication) yapılmasına olanak sağlamaktadır.
- Sistemin güvenilirliğinin yanı sıra hızının da yüksek olması için, kullanılacak anahtarın sayısal büyüklüğü önemlidir. Yeterli güvenilirlik derecesine ulaşmak için gerekli büyüklük Eliptik Eğri Şifreleme (ECC) Algoritması kullanılarak belirlenmektedir.



4.2.2. RSA'nın Avantajları

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

4.2.3. RSA'nın Dezavantajları

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

4.2.4. RSA Algoritmasının İncelenmesi

- Yeterince büyük iki adet asal sayı seçilir. Bu sayılar; p ve q varsayılsın.

- Daha sonra n = p*q hesaplanır. Buradaki n sayısı iki asal sayının çarpımıdır ve hem şifreleme hem de deşifreleme için taban (modulus) olarak kabul edilir.

- Totient fonksiyonu hesaplanır. Burada çarpanların ikisi de asal sayı olduğu için $\varphi(n) = (p-1) * (q-1)$ olarak bulunur.

- Hesaplanan totient fonksiyonu, değeri ($\varphi(n)$) ile aralarında asal olan bir e sayısı alınır.

- Belirlenen bu değer $1 < e < \varphi(n)$ olmalıdır. Bu seçilen e sayısı açık(public) anahtar(key) olarak belirlenir.

- Daha sonra d sayısı hesaplanır. Bu sayı için şu denklik geçerli olmalıdır. $d * e \equiv 1 \bmod (\varphi(n))$. Bu d değeri private anahtar olarak saklanır. Bu sayının hesaplanması sırasında uzatılmış öklit (extended euclid) algoritmasından faydalanılır.

RSA'nın implementasyonu yapılırken ilk olarak p ve q gibi iki sayı üretildi ve bunlar miller rabin asallık testinden geçirildi. Testten sonra gcd fonksiyonu kullanarak $\varphi(n)$ ile aralarında asal olan e public değeri üretildi. İlk olarak üretilen p ve q asal sayıları çarpılarak şifreleme ve deşifrelemede mod tabanı olarak kullanılacak olan n tasarlandı.

Totient fonksiyonu gereği oluşturulan p ve q sayılarının bir eksiği alınarak ((p-1)*(q-1)) çarpım sonucu f değeri oluşturulur. Bu adımdan sonra şifreleme işlemine geçilmiştir. Şifreleme işlemi m^e mod n şeklinde gerçekleştirilir. Alıcı tarafta gelen şifreyi çözmek için e public değeri ve f değeri ile $d*e \equiv 1 \bmod (\varphi(n))$ işlemini uygulayarak d private değerini üretir. Oluşturulan p * q = n değerini de şifreyi gönderen taraftan alarak c^d mod n ile deşifreleme işlemini gerçekleştirir.

4.2.5. Miller Rabin Testi

n - 1 = 2^s r (Formül 1)

n rastsal sayısının M&R asallık testi için ilk önce n-1'in ikiye bölünme sayısı olan s değeri ile Formül 1'deki eşitliği sağlayan r değeri hesaplanır. Geriye kalan aşamaların adım adım uygulanması aşağıda verilmiştir.

- n'den küçük olacak bir rastsal a sayısı bulunur.
- j = 0 olarak ayarlanıp z = a^r mod n hesaplanır.
- Eğer (z = 1) veya (z = n – 1) ise n asallık testini geçer ve asal olabilir.
- Eğer (j > 0) ve (z = 1) ise n asal değildir.
- j = j + 1 olarak ayarlanır. Eğer (j < s) ve (z ≠ n-1) ise (z = z^ 2 mod n) olarak ayarlanır ve 4. Adıma geri dönülür. Eğer (z = n – 1) ise n asallık testini geçer ve asal olabilir.
- Eğer (j = s) ve (z ≠ n - 1) ise o zaman n sayısı asal değildir.

4.2.5.1. M&R Testinde Asal Taban Almanın Avantajları

M&R Testi güçlü asallık testi olarak bilinir. Böyle olmasına rağmen bu testi yanıltan sayılar vardır. M&R Testini yanıltan sayılara güçlü yalancı tanık denir. Bazı bileşik sayılar, çok az sayıda güçlü yalancı tanığa sahiptir.

Örnek olarak bileşik 105 (3 x 5 x 7) sayısının yalancı tanıkları 1 ve 104'tür. Buradan varılan genelleme şudur: bir n sayısı 2 veya daha fazla ilk tek asal sayının çarpımından oluşuyorsa, bu sayının yalancı tanıkları sadece 1 ve n-1 olmaktadır. Güçlü yalancı tanıklar yüzünden M&R testinde yanlış sonuçlara varılmaması için taban olarak ilk asalların (2, 3, 5, 7 gibi) alınması önerilmektedir. Birçok bileşik sayı için güçlü yalancı tanıkların adedi, Q(n) fonksiyonu için maksimum Q(n) / 4 olmaktadır.

Yalancı-tanıklık (non-witness) Kavramı Bir bileşik n sayısı için, W(n) kümesi n'in asal olmadığını kanıtlayabilecek sayılardan yani tanık sayılardan oluşsun. Tümleyen küme L(n), L(n) = Zn – W(n) elemanlarından oluşacaktır ve bu kümenin elemanları yalancı-tanık olarak adlandırılır. Eğer testlerde parametre olarak yalancı-tanıklar kullanılırsa yanlış sonuçlara ulaşılması mümkündür.

Bu testler bileşik sayının asal olduğunu bildireceklerdir. Bu tür yanlışlıklarla karşılaşmamak için bu tür testleri (yeteri kadar büyük bir t sayısı için) t kere tekrarlanması hata olasılığını daha da düşürecektir . Miller&Rabin testi için yalancı-tanıkların sayısının çok az olduğu Higgins'ın yaptığı araştırmalarda ortaya çıkmıştır.

4.2.6. Genişletilmiş Öklit Algoritması

Genişletilmiş Öklid algoritması, cebirsel alan uzantılarında ve özellikle asal olmayan sıralamadaki sınırlı çarpımlarda çarpımsal ters hesaplamayı sağlar. Öklid algoritmasının kriptografide yaygın olarak kullanıldığı anlaşılmaktadır. Özellikle, modüler çoğaltılmış ters hesaplama, RSA genel anahtar şifreleme yönteminde önemli bir adımdır.

Bu yöntemin amacı belirli bir tabana göre verilen sayının tersini bulmaktır. Yani basitçe anlatırsak de= 1 mod f denklemini bilinen bir e ve f sayıları ile çözüme ulaştırır. Başka bir ifade ile sayının bir modda hangi sayıyla çarpılınca 1 sonucunu verdiğini bulmaktadır.

Buna göre algoritmada tersi alınacak olan sayı e ve taban değeri olan f biliniyor olacak ancak d sayısı aranacaktır. Bu sayı aşağıdaki şekilde de ifade edilebilir: Yukarıda e sayısının tersi olarak ifade edilen d sayısı için aşağıdaki algoritma adımları izlenebilir:

- (X1, X2, X3) <- (1 ,0 ,f) sayıları konulur.
- (Y1, Y2, Y3) <- (0, 1, e) sayıları konulur.
- Şayet Y3 değeri 0 ise tersi yoktu hükmüne varılıp algoritma sona erdirilir.
- Şayet Y3 değeri 1 ise aranan ters değer Y2 değeridir ve eY2 ≡ 1 mod f ya da *e^-1 = Y2 mod f* sonucuna ulaşarak algoritma sonlandırılır.
- Q değeri hesaplanır : Q= |_ X3 / Y3 _|
- (T1, T2, T3) <- (X1 – QY1, X2 – QY2, X3 – QY3) değerleri hesaplanarak yerleştirilir.
- (X1, X2, X3) <- (Y1, Y2, Y3)
- (Y1, Y2, Y3) <- (X1, X2, X3)
- bu adımdan 2. adıma geri dönülür.

Belirtilmiş olan yapıda uzatılmış öklit metodunda çıkan sonuç yani verilmiş olan e ve f değerleri için bulunan d (algoritmadaki Y2) değeri sonuçta aşağıdaki yorumlara izin verir:

ed ≡ 1 mod f

df ≡ 1 mod e

dolayısıyla hem e hem de f tabanına göre sayının tersi bulunmuş olur.

4.3. Hash Fonksiyonu (Mesaj Özeti) Nedir?

Hash fonksiyonu, değişken uzunluklu veri kümelerini, sabit uzunluklu veri kümelerine haritalayan algoritmadır. Bu sabit uzunluklu veri kümelerine "**Message Digest"** veya "**Checksum"** denilmektedir.

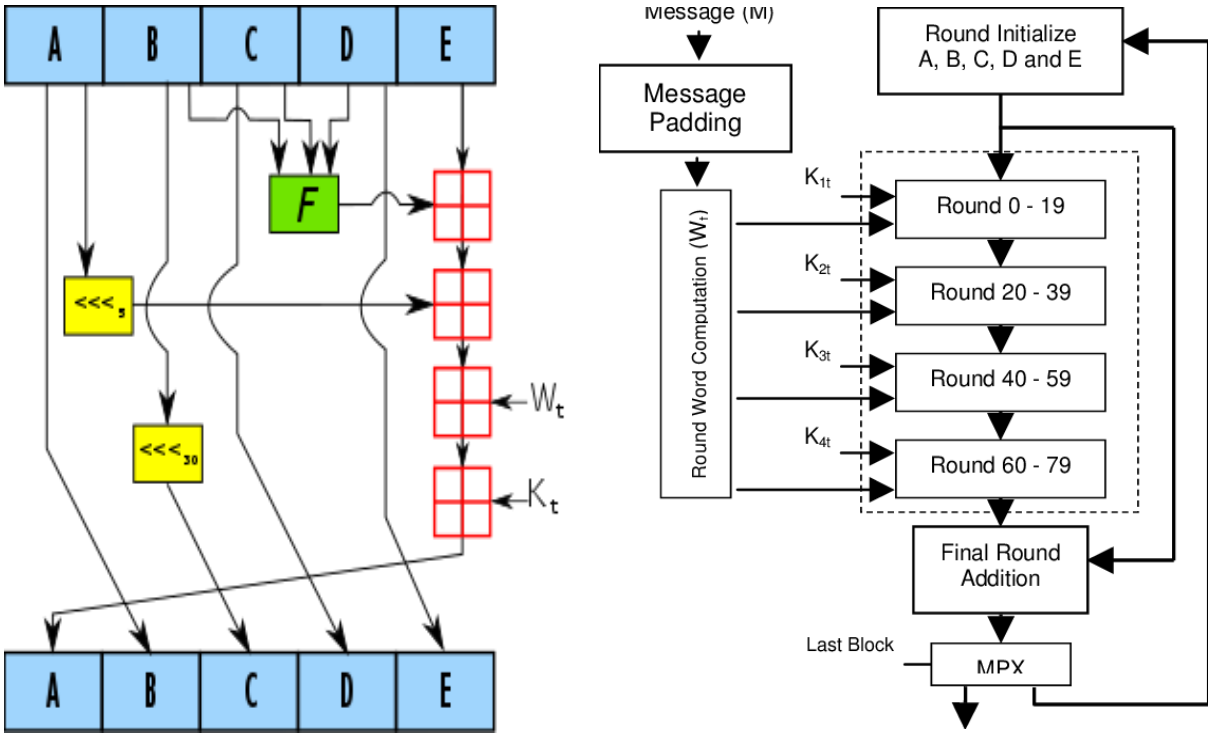
Literatürde tek yönlü algoritmalar (**one way algorithms**) şeklinde bilinmektedir. Tek yönlü algoritma; algoritmanın ürettiğin sonuçtan asıl metinin tekrar elde edilememesidir. Hash fonksiyonunu normal şifreleme algoritmalarından ayıran kısım, geri dönüşün mümkün olmamasıdır. Tüm hashing algoritmaları aldıkları veriyi bir takım matematiksel işlemlerden geçirerek geri hesaplanması ve döndürülmesi imkansız çıktılar üretirler. Şifreleme algoritmaları ise çift yönlüdür, yani üretilen sonuçtan asıl metin(plain text) i geri elde edilebilir.

İki farklı girdi aynı çıktıyı oluşturuyorsa buna "**collision**" denir. Collision bulunduğu anda hash fonksiyonu kırılmıştır ve kriptografik olarak güçlü değildir. SHA-1 için 2017'de 6610 yıl işlemci zamanı kullanılarak bir collision tespit edilmiştir, MD5'ta ise 2005'ten beri collision'lar vardır. Tespit edilen collusion'lardan dolayı MD5 ve SHA-1 Hash fonksiyonları günümüzde kriptografik olarak yeterince güçlü kabul edilmemektedir.

4.3.1. SHA-1 Hash Fonksiyonu

SHA1 algoritması TLS, SSL, PGP, SSH gibi en yaygın uygulama ve protokollerde yer almaktadır. SHA-1, mesaj özeti olarak da bilinen 160-bit hash (özet) değeri üretir.

SHA-1 Hash Fonksiyonunun Block Diyagramları



4.3.1.1. SHA1 Özeti Alma

Özet değerini bulmak için "**mavi**" kelimesinin SHA1 özeti incelenecek olursa;

- Kelimedeki karakterler ASCII formata çevrilir. → m=109 a=97 v=118 i=105
- Daha sonra binary formatına çevrilir;
  → m=109=01101101 a=97=01100001 v=118=01110110 i=105=01101001
- Binary'ye çevrilen karakterler birleştirilir; 01101101011000010111011001101001
- Birleştirilen karakterlerin sonuna 1 eklenir: 01101101011000010111011001101001**1**
- Mod 512'de 448 karakter uzunluğunda olacak şekilde yukardaki belirtilen değerin sonuna 415 adet 0 eklenmesi gerekmektedir.
- Mesajın sonuna orijinal mesajın uzunluk bilgisi 64 bit olacak şekilde eklenir: (orijinal mesajımızın uzunluğu 32 bit)

```
01101101011000010111011001101001100000000000000000000000000000000000000-
00000000000000000000000000000000000000000000000000000000000000000000000-
00000000000000000000000000000000000000000000000000000000000000000000000-
00000000000000000000000000000000000000000000000000000000000000000000000-
00000000000000000000000000000000000000000000000000000000000000000000000-
00000000000000000000000000000000000000000000000000000000000000000000000-
00000000000000000000000000000000000000000000000000000000000000000000000-
00000000000000000000000000000000000000000000000000000000000000100000
```

- Daha sonra mesajın uzunluğu 512 bitten uzun olması durumunda, değer 512 bit'lik parçalara bölünmesi gerekmektedir. 512 bitlik mesaj 32 bit'lik 16 parçaya bölünür:

0: 01101101011000010111011001101001

1: 10000000000000000000000000000000

2: 00000000000000000000000000000000

13: 00000000000000000000000000000000

14: 00000000000000000000000000000000

15: 00000000000000000000000000100000

……..

- Bu adımdan sonra 16 parçaya bölünmüş mesaj 80 parça oluncaya kadar çoğaltılır.

```
for i from 16 to 79  w[i] = (w[i-3] xor w[i-8] xor w[i-14] xor w[i-16]) leftrotate 1
```

- İlk for döngüsü i=16 için 13,8,2 ve 0 numaralı mesaj parçaları seçilir. Elde edilecek yeni mesaj parçaları bu döngü içerisinde önceki mesaj parçalarının xor'lanması ile bulunur. Sonuçta çıkan mesaj 1 bit sola ötelenerek yeni bir mesaj oluşturulmuş olur.

13.parça **XOR** 8. Parça **XOR** 2. Parça **XOR** 0. Parça:01101101011000010111011001101001

1 bit sola öteleme: 11011010110000101110110011010010 (Elde edilen sonuç 16.mesajdır.

16.mesaj: 11011010110000101110110011010010

- Daha sonra 17 ile 79 arasındaki mesajlarda aynı adımlar ile hesaplanmaktadır.
- SHA1 algoritmasında 5 adet başlangıç değişkeni kullanılır. Bunlar;

```
1. değer →  h0 = 01100111010001010010001100000001
2. değer →  h1 = 11101111110011011010101110001001
3. değer →  h2 = 10011000101110101101110011111110
4. değer →  h3 = 00010000001100100101010001110110
5. değer →  h4 = 11000011110100101110000111110000
```

- 5 adet başlangıç değişkeninden, elde edilen 80 mesajın çeşitli işlemlerden geçirilmesiyle yeni değerler elde edilir. Bulunan bu yeni değerler birleştirildiğinde özet değeri bulunmuş olur. Başlangıç değişkenleri h0=A, h1=B, h2=C, h3=D, h4=E olacak şekilde harfle ifadelendirilir.

- İlk 20 mesaj için fonksiyon 1, ikinci 20 için fonksiyon 2, üçüncü 20 için fonksiyon 3, dördüncü 20 için fonksiyon 4 kullanılır. Her fonksiyonda farklı işlemler yapılmaktadır.
- Birinci fonksiyonda f = (B **and** C) **or** ((**not** B) **and** D) işlemi yapılır.

```
B:11101111110011011010101110001001
C:10011000101110101101110011111110
B AND C  →  10001000100010001000100010001000
!B AND D → 00010000001100100101010001110110
F : (!B AND D) OR (B AND C) → 10011000101110101101110011111110
```

- Her fonksiyonda bir de k değeri kullanılır.

  1.Fonksiyondaki K değeri → 01011010100000100111100110011001'dir.

- F değeri bulunduktan sonra; A değeri 5 bit sola ötelenir. Bu işlemde F, E, K ve ilk mesajla toplanır. [(A lrotation 5) + F + E + K + Word 0]

```
K → 10110101000001001111001100110001
A lrotation 5 → 11101000101001000110000000101100
F → 10011000101110101101110011111110
(A lrotation 5) + F →  11000000101011111001111010010101
E → 11000011110100101110000111110000
(A lrotation 5) + F + E → 10010001010011001000011111100011010
(A lrot 5) + F + E + K  → 11010011111101101001001100010110011
```

```
Word 0 → 01101101011000010111011001101001 ( 16 parçaya ayrılan ilk mesajdır. )
(A lrot 5) + F + E + K + Word 0 → 11110000110100010110000011110001110 0
```

Bulunan değere temp adı verilmektedir. **Temp**: 11110000110100010110000011110001110 0

Yukarıdaki işlemlerin kod şeklinde gösterimi:

```
for i from 0 to 79
if 0 ≤ i ≤ 19 then    f = (b and c) or ((not b) and d)     k = 0x5A827999
else if 20 ≤ i ≤ 39            f = b xor c xor d          k = 0x6ED9EBA1
else if 40≤ i ≤ 59  f = (b and c) or (b and d) or (c and d)  k = 0x8F1BBCDC
else   if 60   ≤   i   ≤   79              f   =   b xor c xor d
k=0xCA62C1D6
temp = (a leftrotate 5) + f + e + k + w[i]        e = d        d = c
c = b leftrotate 30        b = a        a = temp
```

- Toplama işleminde 32 bitlik değerler toplandığında elde kalırsa 32 bit kalacak şekilde soldaki fazla bitler atılır. Ardından değişkenler şu şekilde yeniden belirlenir.

  E=D D=C C=B Left Rotate 30 B=A A=temp

Kısaltılmış temp:	00011010001011000001111000111 00
E = D	00010000001100100101010001110110
D = C	10011000101110101101110011111110
C = B Left Rotate 30	01111011111100110110101011100010
B = A	01100111010001010010001100000001
A = Temp	00001101000101100000111100011100

- 80 mesaj döngüye girip çıktıktan sonra A, B, C, D, E değerleri her döngüde değişerek 80. mesajda son halini alır. Son adımda yukarıdaki değerler başlangıçla toplanmaktadır.
  → h0 = h0 + A h1 = h1 + B h2 = h2 + C h3 = h3 + D h4 = h4 + E

  Kısaltılmış değerleri belirtirsek;

```
H0: 11110000101000000000011110011100
H1: 10010111010001110011010100011000
H2: 10111110101110001111001000000001
H3: 00101000110101110001000010100100
H4: 01001001100101110000110100011001
```

- Sonuç olarak "mavi" kelimesinin SHA1 özet değeri aşağıdaki gibi belirtilebilir:

```
11110000101000000000011110011100100101110100011100110101000110001011111010111000 11
11001000000001001010001101011100010000101001000100100110010111000011010001100 1
Ya da : f0a0079c97473518beb8f20128d710a449970d19
```

5. SONUÇLAR

Proje kapsamında tasarladığımız “Simetrik Şifreleme” algoritmalarını bir platformda toparladık. Bu işlemi yaparken birçok parametreye dikkat ederek algoritmalarda iyileştirmeler yaptık. Tasarımların akış diyagramlarını “flowgorithm” uygulaması üzerinden görselleştirdik. Burada gerçeklenen algoritmaların kodlama kısımlarını tamamladıktan sonra üretilen anahtarın uzunluğu ile bağlantılı olarak BruteForce test işlemini gözlemledik.

Günümüzde güvenli haberleşme için kullanılan SSL protokolünü baz alarak, tasarlanılan simetrik şifreleme algoritmalarıyla bir güvenlik protokolünün implementasyonunu sağlamayı hedefledik. Bu süreçte ilk olarak simetrik şifreleme algoritma yapılarını oluşturup, "diffusion" ve "confusion" değerleri ile algoritmaların performansını inceledik.

Simetrik şifreleme yapılarının tasarım sürecinde; daha önce kullanılmış olan birçok şifreleme algoritmalarını inceleme fırsatı yakaladık. Bunların başında gelen "Playfair" algoritması, belirli bir matrisi kullanarak şifreleme işlemi yapmaktadır. Bu algoritmada bulduğumuz bir zafiyetle şifreli metni, sadece kullanılan matrisin boyutu bilinerek deşifreleme işlemi yapılabileceğini gözlemledik.

Projenin ilk kısmında verinin şifreleme ve deşifreleme işleminde kullanılan gizli anahtarın her iki tarafta da bulunduğu kabul edilerek implementasyonunu sağladık. İlerleyen bölümde ise anahtarın paylaşımını güvenli bir algoritması ile (örnek: RSA) sağlayarak, güvenlik protokolünü tamamlamayı hedefledik.

Projenin ilerleyen kısmında ise SSL protokolünün özelliklerini barındıran bir yapının implementasyonu için protokol içinde bulunan alt katmanları detaylı bir şekilde inceledik. Bu süreçte hocamızın tavsiyesiyle “SSL & TLS Essentials - Securing the Web” kitabı üzerinden Handshake-Alert-Record protokolleriyle ilgili araştırmalarımızı tamamladık. Algoritma tasarımından sonra SSL protokolünde mevcut olan; Şifrelemedeki anahtarların paylaşımında Handshake, Alert ve Record protokollerini içeren bir implementasyonu sağladık.

Algoritmaların uygulamalara implementasyonundan sonra, bir istemci-sunucu yapısı ile şifreleme algoritmalarını test ettik. Bu süreçte ilk olarak açık metni şifreleyip gönderimi sağladık. Gönderilen şifreli metni “Wireshark” üzerinden test ettik. Daha sonra şifreli metne yaptığımız deşifreleme işlemiyle açık metni elde ettik.

6. ÖNERİLER

Güvenli veri iletişimi için tasarlanılan simetrik şifreleme algoritmalarını bir soket programlama yapısı kullanılarak iletim sağlandı. Bu iletim sırasında bir sunucu ve bir istemci üzerinden testleri gerçekleştirildi.

İletişimin çoklu kullanıcı üzerinden gerçekleşmesi sağlanabilmesi için birden çok port açarak yayın (broadcast) iletim sağlanabilir. İletişime katılan her istemci için açılan port, ayrı bir anahtar paylaşımını gerektirmektedir.

Farklı platformlarda yazılmış olan programların aynı çatı altında toplanması, kullanılan metotların benzerliğine bağlı olarak kodlanma süresi değişebilir. Bu süreçte sıklıkla debug işlemine başvurulabilir.

Şifreleme metotlarının bulunduğu kütüphane, açık anahtarın paylaşımından sonra kullanılan platform üzerinde optimum şekilde çalışabilmesi gerekir. Bu yapı ilk önce bir bilgisayar üzerinden test edildi. Test sırasında belirtilen veri yollarının başta statik olarak belirlenmesi, farklı bilgisayarlar üzerinden çalışırken sorun çıkarttı. Bu nedenle şifreleme ve deşifreleme yapmadan önce kullanıcılardan veri yollarının bilgileri alınarak sorun çözüldü.

İletişim için kullanılan soket programlama yapısında, gerekli metotları incelerken yapılması gereken ilk işlem; sıkça kullanılması gereken github kaynaklarına bakılmasıdır. Belirtilen kaynaklarda birçok farklı kullanımı görmek mümkündür.

7. Kaynaklar

- SSL and TLS Essentials – Securing the Web (Stephen Thomas)
- Şifreleme (Doktora dersi) – (cs.stfx.ca/~ltyang/csci-467)
- http://sutod.selcuk.edu.tr/sutod/article/viewFile/120/601
- https://crypto.stanford.edu/~nagendra/papers/dtls.pdf
- https://www.cise.ufl.edu/~nemo/crypto/slides/ch16.pdf
- https://tools.ietf.org/html/rfc5246#page-29
- http://blog.btrisk.com/2014/04/ssl-nedir-2bolum.html
- https://www.iacr.org/archive/fse2006/40470265.pdf
  (A Study of the MD5 Attack: Insihts and Improvements)
- https://doc.lagout.org/security/Crypto/Implementing../.pdf
  (Implementing Using Cryptography and PK)
- https://get-itlabs.com/category/security/
- https://www.exploit-db.com/docs/[.]turkish/40448-wireshark-important-tips.pdf
- "C# Network Programming," by Richard Blum, Sybex.
- https://github.com/Aksoyy/Design_Graduation_Project
- http://www.cse.unt.edu/~tarau/teaching/PP/NumberTheoretical/RSA%20(cryptosystem).pdf
- http://www.kalemacademy.com/Cms_Data/Sites/KalemAcademy/Files/KalemAcademyRepository/sayilar/Sayi4_05RSASifrelemesi.pdf
- https://en.wikipedia.org/wiki/SHA-1
- https://kerteriz.net/hash-almak-nedir-encryption-ile-arasindaki-farklar-nelerdir/
- https://en.wikipedia.org/wiki/Collision_attack
- https://tr.wikipedia.org/wiki/Hash_fonksiyonu

STANDARTLAR ve KISITLAR FORMU

Projenin hazırlanmasında uyulan standart ve kısıtlarla ilgili olarak, aşağıdaki soruları cevaplayınız.

1. Projenizin tasarım boyutu nedir? (Yeni bir proje midir? Var olan bir projenin tekrarı mıdır? Bir projenin parçası mıdır? Sizin tasarımınız proje toplamının yüzde olarak ne kadarını oluşturmaktadır?)

Projenin ilk kısmında simetrik şifreleme algoritması tasarımı gerçeklenmiştir. İlerleyen bölümde bu algoritmaları kullanarak SSL protokololüne benzer bir yapı tasarlanmıştır.

2. Projenizde bir mühendislik problemini kendiniz formüle edip, çözdünüz mü? Açıklayınız.

Tasarlanılan simetrik şifreleme algoritmalarını formüle edip, örnek bir sunucu ve istemci uygulaması üzerinden testi gerçeklenmiştir.

3. Önceki derslerde edindiğiniz hangi bilgi ve becerileri kullandınız?

Projemizde nesne yönelimli programlama yapılarını sıklıkla kullanıldı. Bu yapılarla birlikte; sunucu ve istemci arasında kurulan soket programlama yapısı için "Bilgisayar Ağ Programlama" dersinden yararlanılmıştır.

4. Kullandığınız veya dikkate aldığınız mühendislik standartları nelerdir? (Proje konunuzla ilgili olarak kullandığınız ve kullanılması gereken standartları burada kod ve isimleri ile sıralayınız).

Şifrelemelerin implementasyonu ortak bir programlama dilinde gerçeklenmiştir. Daha sonra testleri için "Wireshark" programından faydalanılmıştır.

5. Kullandığınız veya dikkate aldığınız gerçekçi kısıtlar nelerdir? Lütfen boşlukları uygun yanıtlarla doldurunuz.

a) Ekonomi

Projenin ekonomik açıdan bir maliyeti bulunmamaktadır.

b) Çevre sorunları:

Projenin çevreye olumsuz bir etkisi bulunmamaktadır.

c) Sürdürülebilirlik:

Tasarlanacak şifreleme algoritma sayılarının artması, ilerleyen zamanda kullanımı olumlu yönde etkileyecektir.

d) Üretilebilirlik:

Projenin ilerleyen kısmında tasarlanılan algoritmaların verimliliği ile kullanıcıların etkinliği artmaktadır.

e) Etik:

Proje etik açıdan bir problem teşkil etmemektedir.

f) Sağlık:

Proje sağlık açısından bir sorun teşkil etmemektedir.

g) Güvenlik:

Projenin amacı “Veri güvenliği” olduğu için, bu konuda yapılan istemci-sunucu testlerinde verinin şifrelendikten sonra gönderilmesi, araya girebilecek kötü niyetli kişilerin açık metne erişimini engellemektedir.

h) Sosyal ve politik sorunlar:

Projenin herhangi bir sosyal ve politik bir sorunu bulunmamaktadır.