**T.C**

**FIRAT ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

# ARM İŞLEMCİLER İLE TEK KULLANIMLIK ŞİFRE UYGULAMASININ GERÇEKLEŞTİRİLMESİ

**YÜKSEK LİSANS TEZİ**

**Ömer TÜRK**

**(091129101)**

**Anabilim Dalı: Bilgisayar Mühendisliği**

**Programı: Yazılım**

**Tez Danışmanı: Yrd. Doç. Dr. A. Bedri ÖZER**

**Tezin Enstitüye Verildiği Tarih: 22.04.2011**

**NİSAN–2011**

**T.C**

**FIRAT ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

# ARM İŞLEMCİLER İLE TEK KULLANIMLIK ŞİFRE UYGULAMASININ GERÇEKLEŞTİRİLMESİ

**YÜKSEK LİSANS TEZİ**

**Ömer TÜRK**

**(091129101)**

**Anabilim Dalı: Bilgisayar Mühendisliği**

**Programı: Yazılım**

**Tezin Enstitüye Verildiği Tarih : 22.04.2011**

**Tezin Savunulduğu Tarih : 24.03.2011**

**Tez Danışmanı:** **Yrd. Doç. Dr. A. Bedri ÖZER (F.Ü)**

**Diğer Jüri Üyeleri:** **Doç. Dr. Mehmet KAYA (F.Ü)**

**Yrd. Doç. Dr. Mustafa TÜRK (F.Ü)**

**NİSAN–2011**

## ÖNSÖZ

Tez çalışmam boyunca yardımlarını esirgemeyen ve beni yönlendiren çok değerli hocam Sayın Yrd. Doç. Dr. Ahmet Bedri ÖZER ’e, bana manevi destekte bulunan aile bireylerine sevgi ve saygılarımı sunarım.

**Ömer TÜRK**

**ELAZIĞ - 2011**

## İÇİNDEKİLER

**Sayfa No**

**Sayfa No**

## ŞEKİLLER LİSTESİ

**Sayfa No**

**Sayfa No**

## TABLOLAR LİSTESİ

**Sayfa No**

**KISALTMALAR**

**AES** : Advanced Encryption Standart

**ARM** : Acorn RISC Machine

**ARPANET** : Advanced Research Projects Agency Network

**CHA** : Chaotic

**DARPA** : The Defense Advanced Research Projects Agency

**DES** : Data Encryption Standart

**DDN** : Digital Divide Network

**FIPS** : Federal Information Processing Standards

**HOTP** : HMAC One Time Password

**HMAC** : Hash Message Authentication Code

**IANA** : Internet Assigned Numbers Authority

**IESG** : Engineering Steering Group

**IETF** : Internet Engineering Task Force

**ISOC** : Internet Society

**MAC** : Message Authentication Code

**MD5** : Message Digest Algorithm 5

**NIST** : National Institute of Standards and Technology

**OTP** :One Time Password

**RFC** : Request for Comments

**SHA-1** : Secure Hash Algorithm-1

## ÖZET

Tek kullanımlık şifre uygulaması, kimlik doğrulamanın önemli olduğu sistemlerde statik şifrelerden çok daha güvenlidir. Kullanıcının kullandığı şifreler doğrulama onayından geçtikten sonra bir daha kullanılamaz. Kullanıcının her giriş isteğinde yeni üretilen ve eskisinden farklı bir şifre ile kimlik doğrulanma sağlanır. Bu şifrelere atılabilir şifreler de denir. Şifreler üretilirken meydana gelebilecek saldırılar göz önünde bulundurulmalıdır. Bu yüzden şifre üreten sistemlerin güvenli bir özetleme fonksiyonu kullanmaları gerekmektedir.

Günümüzde, benzersiz bir OTP (One Time Password) oluşturmak amacıyla, HMAC (Hash Message Authentication Code) tabanlı üretilen HOTP (HMAC One Time Password) 'lar kullanılmaktadır. HMAC tabanlı HOTP algoritmasında, istemci/sunucu tarafından gizli ortak bir anahtar paylaşılır ve başlama sayacı senkronize başlatılır. Gizli ortak anahtar ve sayacın değeri, özetleme fonksiyonuna gönderilir. Özetleme fonksiyonu ile gelen mesaj özetlenir. HMAC sonucu olarak dönen özetleme fonksiyonunun çıkışı kesilip HOTP değerleri elde edilir. Bu çalışmada MD5 ve SHA–1 klasik özetleme fonksiyonlarının yanı sıra kaotik harita ile oluşturulmuş özetleme fonksiyonu kullanılarak, saldırılara karşı daha güvenilir HOTP 'lar elde edilmeye çalışılmıştır.

HOTP doğrulama mekanizmasında etkileşimde bulunan iki taraf bulunur. Bu tezde, sunucu üzerinde çalışabilecek bir HOTP doğrulama ve Arm9 ailesinden Mini–2440 ürünü üzerinde çalışan bir HOTP üreteç yazılımı geliştirilmiştir. Bu iki yazılım birbirleriyle senkron halde tek kullanımlık şifre üretme ve bu şifreyi doğrulama işlemlerini yapmaktadırlar.



**Anahtar Kelimeler:** ARM, HMAC, Kaotik Özetleme Fonksiyonu, Tek Kullanımlık Şifre, Atılabilir Şifre.

## SUMMARY

The application of one time passwords (OTP) is much safer than static passwords for the systems which give special importance to authentication. The passwords employed by the user cannot be used again after the authentication approval. Authentication is ensured by providing a new password, which is different from the earlier ones, for each time the user wants to access to the system. These passwords are also called as "disposable passwords". While the passwords are being generated, it should be taken into consideration that they may be attacked. Therefore, password-generating systems are required to use a secure hash function.

At present, HOTPs (HMAC One Time Password), which are produced as HMAC (Hash Message Authentication Code) based, are used in order to create a unique OTP (One-Time-Password) [1]. In the HMAC-based HOTP algorithm, a secret common key is shared by client/server and starting counter is launched synchronically. The secret common key and the value of counter are sent to hash function. The message that comes with hash function is summarized. The output of hash function, which returns as HMAC result, is truncated and the HOTP values are acquired. In this study, besides the MD5 and SHA-1 classic hash functions, by using the hash function created by chaotic map, more secure HOTPs against the attacks have been tried to be obtained.

In the HOTP authentication mechanism, there are two sides in interaction. In this thesis, one HOTP authentication which can work on server and one HOTP generator software which can work on the Mini–2440 product from the family of ARM9 has been developed. These softwares, working in synchronization, produce one-time-use password and verify that password.



**Keywords:** ARM, HMAC, Chaotic Hash Function, One Time Password, Disposable Password.

# 1. GİRİŞ

Günümüzde, internet uygulamalarının neredeyse tümü, kendi veri kaynaklarına erişim için şifreli kimlik doğrulama yöntemi kullanmaktadır; ancak iki taraflı kimlik doğrulama kullanımı ve üretimi artan tehdit ve saldırı seviyelerine rağmen kullanıcı erişimi için halen zayıf kimlik doğrulama yöntemlerine dayanmaktadır. Artan bu tehdit ve saldırılar, statik şifre kullanımının saldırılar karşısında yetersiz kalmasına neden olmuştur. Bu saldırılar sonucu uygulamalar için veri şifreleme ve kimlik doğrulama büyük bir önem kazanmıştır.

Kullanıcıların ve uygulamaların veri erişimini güvenli bir şekilde sağlamaları için, kullanıcı dostu ve her veri erişiminde değişen şifreli kimlik doğrulama yöntemleri ön plana çıkmıştır.

Genel olarak geliştirilen kimlik doğrulama yöntemleri: şifre yardımıyla kimlik doğrulama, akıllı kart yardımıyla kimlik doğrulama ve parmak izi kullanarak kimlik doğrulama olarak sınıflandırılabilir. Ayrıca şifre kullanan bir kimlik doğrulama sisteminde sunucu tarafında herhangi bir şifre veya doğrulama tablosu saklanmamalı ve şifreler, sistem yöneticisi dahil olmak üzere hiç kimse tarafından görülmemeli, düz metinler halinde ağda iletilmemeli, kullanıcılar için ideal uzunlukta olmalıdır. Ayrıca kullanıcının yanlış şifre girdiği, tespit edilebilme gibi özelliklere sahip olmalıdır [1–2].

Çoğu ağ uygulamaları; kullanıcıları, bir kullanıcı adı ve şifre sistemiyle doğrularlar. Tek kullanımlık şifre sistemleri ise, tekrarlanan şifreyi geçersiz kılarak, kullanılmış şifre kullanımını ortadan kaldırır [3]. Günümüzde, Tek kullanımlı şifre uygulamaları için HMAC tabanlı üretilen HOTP ’lar kullanılmaktadır [4].

Bu çalışmada, “Tek Kullanımlık Şifre” uygulamasının tasarım aşamaları, özetleme fonksiyonları, kaotik çadır haritalama ve Arm İşlemci ile çalışması incelenecektir.

## 1.1. Tezin Amacı

Tek kullanımlık şifre, doğruluğun en önemli olduğu alanlardan biri olan kimlik doğrulama mekanizması için en yüksek güvenceyi sağlayan mekanizmalardan biridir.

Bu tezde tek kullanımlık şifre uygulaması yüksek hız ve güvenlik sebebi ile HOTP üreteç yazılımı ARM işlemci ile gerçekleştirilmiştir. Ayrıca klasik özetleme fonksiyonları yerine kaotik harita ile oluşturulmuş özetleme fonksiyonu kullanılarak güvenlik seviyesi artırılmaya çalışılmıştır.

## 1.2. Tezin Yapısı

Tezin yapısı aşağıdaki gibi oluşturulmuştur.

Birinci Bölüm: Bu bölümde güvenlik standartlarını yayımlayan kurumlar hakkında ve geliştirilen yazılımın istemci tarafını oluşturan Arm işlemciler hakkında genel bilgiler verilmiştir.

İkinci Bölüm: Bu bölümde genel kriptoloji ile ilgili bilgiler verilmiş ve mesajlaşan tarafların karşılaşabilecekleri saldırılara değinilmiştir.

Üçüncü Bölüm: Bu bölümde herhangi bir uzunluktaki mesajı işleyip sabit mesaj özeti üreten ve tersini hesaplamak neredeyse imkansız olan özetleme fonksiyonlarına değinilmiştir.

Dördüncü Bölüm: Bu bölümde mesajlaşan tarafların gönderdikleri mesajlara üçüncü kişilerin etkilerini önlemek amacı ile mesajlara eklenen ileti kimlik doğrulama kodları anlatılmaya çalışılmıştır.

Beşinci Bölüm: Bu bölümde Kaos tabanlı özetleme fonksiyonlarına değinilmiş ve tez kapsamında kullanılan çadır(tent) harita hakkında bilgiler verilmiştir.

Altıncı Bölüm: Bu bölümde bir HOTP değeri elde etme adımları anlatılmış ve geliştirilen yazılımın çalışması anlatılmıştır.

Yedinci Bölüm: Bu bölümde Kaotik harita ile elde edilen HOTP değerlerinin güvenliği incelenmiştir.

Sonuç Bölümü: Bu çalışmada elde edilen sonuçların değerlendirmesi yapılmıştır.

## 1.3. İnternet Mühendisliği Görev Gücü (IETF)

İnternet Mühendisliği Görev Gücü (Internet Engineering Task Force) ARPANET ve Amerikan Savunma Bilgi Şebekesi (DDN) üzerinde çalışan DARPA ’ ya teknik destek sağlamak için 1986 yılında oluşturulmuştur. Internet Mühendisliği Görev Gücü (IETF), internet protokollerini geliştiren ve standartlaştıran, resmi statüsü olmayan bir gruptur. IETF ’nin çalışmaları ve ürettiği dokümanlar internet üzerinden herkese açıktır. Çalışma gruplarına ve toplantılarına katılım için herhangi bir kısıtlama bulunmamaktadır. Toplantılar genellikle internet üzerinden tartışma grupları aracılığıyla sanal olarak yapılmaktadır.

IETF, ağ tasarımcıları, işletmenler, internet mimarisinin evrimi ve internetin düzgün çalışmasıyla ilgilenen araştırmacıların oluşturduğu uluslararasına açık bir topluluktur.

IETF ’nin amacı internetin daha iyi, kolay bir şekilde kullanımını sağlamak ve bunu yaymaktır. Ayrıca kişilerin teknik ve mühendislik çalışmalarını internet üzerinden daha kolay ve daha hızlı bir şekilde kullanılmasını sağlamak da amaçları arasındadır.

IETF ’nin asıl teknik çalışması kendi çalışma gruplarıyla yapılır. Bu çalışma grupları konu konu çeşitli alanlara ayrılmıştır. Güvenlik grupları, yönlendirme grupları ve ulaşım grupları bunlardan sadece birkaçıdır.

IETF ’ye gruplar aracılığı ile gönderilen tüm dokümanların her biri “IETF Katkısı” olarak adlandırılır ve gerekli gruplar tarafından incelenir.

IANA (The Internet Assigned Numbers Authority), her biri birbirinden farklı internet protokol değerlerini (IP) atayan merkezi koordinatördür. IANA ’da ISOC tarafından patentleşmiş ve internet protokol parametrelerini tanımlamak, atamak ve bunları koordine etmekle sorumludur.

IETF oturumu dışında, mail yoluyla ya da daha farklı şekillerde yapılan ifadeler ve çalışmalar IETF ’nin bir ürünü olarak kabul edilmemektedir. IETF ’ye katılan her katılımcı IETF kurallarını kabul etmiş sayılır. IETF ’deki herhangi bir katılımcı, yapılabilecek ya da mevcut olabilecek belge, video ve ses kayıtlarını kamuya açabilir.

### 1.4. Ulusal Standartlar ve Teknoloji Enstitüsü (NIST)

ABD Ticaret Bakanlığının bir parçası olan Ulusal Standartlar ve Teknoloji Enstitüsü (NIST), ABD hükümeti ve devlet daireleri tarafından kullanmak üzere standartları yayınlar ve rehberlik sağlar. Bu standartlar ve kurallar Federal Bilgi İşleme Standartları ’na (FIPS) uygun şekilde verilir. NIST, güvenlik ve birlikte işlerlik gibi federal hükümetin gereksinimlerini zorlayıcı durumlarda FIPS ’i geliştirir.

NIST, Federal Kayıttaki kamu görüşü ve yorumu için önerilen FIPS ’i yayınlar. Önerilen FIPS, Federal Kayıtta ilan olduğuyla aynı zamanda da NIST web sitesinde ilan edilir. Eğer uygulanabilir ise, önerilen FIPS ile ilgili metin ve ilgili şartnameler NIST sitesinde yayınlanır.

İncelenmek ve yorumların ibraz edilmesi için 90 günlük süre sağlanır. Yorumların NIST ’e teslim edilmesi gereken süre, Federal kayıtta ve diğer bildirilerde belirtilmiştir.

Federal kayıt ilanlarına ve diğer ilanlara karşılık gelen yorumlar, önerilen FIPS için gerekli değişikliklerin olup olmadığını belirlemek için NIST tarafından gözden geçirilir.

Alınan eleştirileri analiz eden ve gerekli değişikliklerin yapılıp yapılmadığını ya da tavsiye edilen değişikliklerin neden yapılmadığını açıklayan ayrıntılı bir gerekçe belgesi hazırlanır.

NIST önerilen FIPS 'i, ayrıntılı gerekçe belgesini ve standartların Federal Hükümet kullanımı için zorunlu ve bağlayıcı olup olmadığını gösteren belgeyi onaylanması için Ticaret Sekreterliğine teslim eder.

FIPS 'in onaylandığını ilan eden bir bildiri Ticaret Sekreterliği tarafından Federal Kayıtta ve NIST 'in Web sitesinde yayınlanır. NIST standartları, ABD hükümeti kullanımı için geliştirilmiş olmalarına rağmen, bunların çoğu endüstride yaygın olarak kullanılmaktadır. AES ve DES örnekleridir.

### 1.5. Arm Hakkında Genel Bilgi

Acorn Computers Ltd. tarafından geliştirme projesi olarak ARM dizaynı 1983 yılında başlamıştır. Roger Wilson ve Steve Furber liderliğindeki ekip, gelişmiş bir MOS Technology 6502 'yi geliştirmişlerdir. Acorn, 6502 'yi temel aldığı için programa benzer chip, şirket için önemli bir avantaj sağlamıştır [5].

ARM mimarisi (Acorn RISC Machine) pek çok gömülü tasarımda kullanılan 32-bit RISC işlemci mimarisidir. Güç tasarruf özelliklerinden dolayı, ARM işlemciler mobil elektronik gibi düşük güç tüketiminin kritik bir parametre olduğu pazarda en fazla tercih edilen CPU 'dur.

Günümüzde ARM işlemci ailesi yeryüzündeki tüm 32-bit gömülü işlemcilerin %75 'ini oluşturmaktadır. ARM işlemciler taşınabilir cihazlardan (PDA, cep telefonu, medya oynatıcılar, avuç içi oyun üniteleri ve hesap makineleri) bilgisayar parçalarına kadar (disk sürücüler, masaüstü) tüketici elektroniğinin her alanında yoğun olarak kullanılmaktadır.

### 1.6. FriendlyArm Mini–2440

FriendlyArm Mini–2440 cihazı, S3C2440 Arm9 mikroişlemciye dayalı tek kartlı bir bilgisayardır. Kart ölçüsü 10 cm x 10 cm 'dir. Bu ölçü Arm sistemlerinin sayısız ürünlere entegre edilmesi açısından idealdir.

Mini–2440, 64 MByte SDRAM, 64–128 MByte Nand flash, 2 Mbyte Nor Flash belleğe sahiptir.


**Şekil 1.1 FriendlyARM Mini-2440**

Mini–2440 cihaz kartının çalışması için 5V enerjiye ihtiyacı vardır. Gerekli diğer enerji voltajı ise kart üzerinden sağlanır. Linux 2.6, Android ve Windows CE işletim sistemlerini destekler.

# 2. KRİPTOLOJİ

## 2.1. Kriptoloji

Bilinmezlik kavramı insanoğlunun sürekli ilgisini çekmiştir. Bu ilgi merak duygusunu söz konusu etmen üzerinde yoğunlaştırmıştır. İnsanoğlu kendilerine ait bilgilerin, başka kişilerin eline geçip, muhtemel olumsuz sonuçlar doğurmasının önüne geçmek için çeşitli şifreleme yöntemleri kullanmıştır.

Kriptoloji kelime olarak, eski Yunanca 'da gizli dünya anlamına gelen "kryptos" ve neden-sonuç anlamına gelen "logos" kelimelerinden gelmektedir.

Kriptoloji, birbiri ile haberleşen tarafların bilgi alışverişini güvenli olarak yapmasını sağlayan; temeli, matematiksel fonksiyonlara, problemlere dayanan tekniklerin ve uygulamaların tümüdür.

Kriptoloji bir şifreleme bilimidir. Kriptoloji iki temel alt dala ayrılmaktadır:

1. Kriptografi
2. Kriptoanaliz

Kriptografi, okunabilir bir belgenin şifrelenmesi ve belgenin şifresinin çözülerek tekrar okunabilecek hale dönüştürülmesi için kullanılan yöntemlerdir.

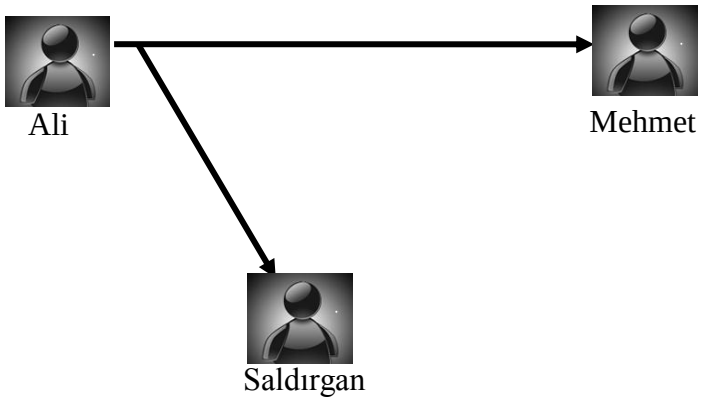
Kriptografinin en büyük amacı pasif ve aktif saldırılar karşısında belgeyi korumaktır.

Kriptoanaliz, bir şifreleme algoritmasını ve şifrelenmiş bir mesajı inceleyerek zayıf ve kuvvetli yönlerini ortaya koymaya çalışır [6].

## 2.2. Güvenlik Saldırıları

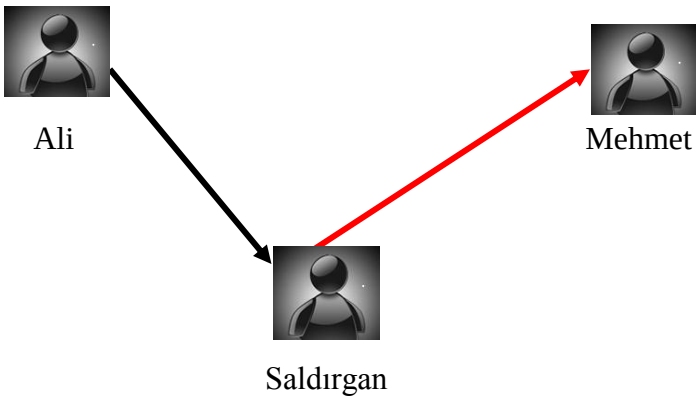
İletişim halinde bulunan kullanıcılar, çeşitli iletişim kanalları kullanarak birbirleriyle haberleşirler. İletişim kanallarında haberleşen bu kullanıcılar çeşitli saldırılara maruz kalabilirler. Bu saldırılar temel olarak aşağıdaki gibi sıralanabilir [6–7]:

- ✓ **Gizlilik İhlali:** Saldırgan gönderici ile alıcı arasındaki mesaj trafiğini dinleyerek mesajları elde edebilir. Elde ettiği bu mesajları okuyarak iletişimin gizliliği ihlal eder.



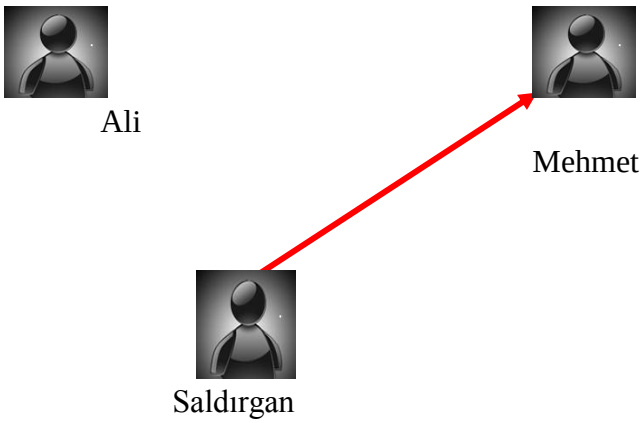

**Şekil 2.1 Gizlilik İhlali**

✓ **Bütünlük İhlali:** Saldırgan haberleşen taraflara müdahale ederek, göndericinin mesajına müdahale edebilir. Aldığı mesajı istediği gibi değiştirerek alıcıya gönderebilir.



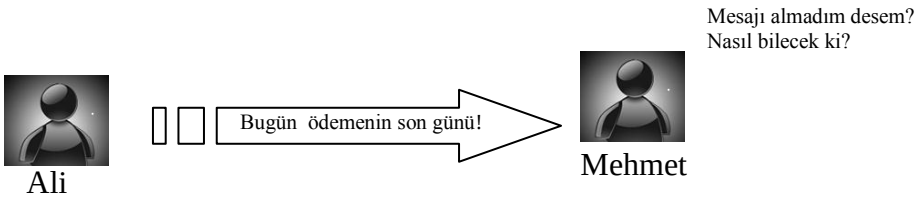

**Şekil 2.2 Bütünlük İhlali**

✓ **Kimlik Doğrulama İhlali:** Saldırgan, alıcıya göndericinin kimliğini taklit ederek bir mesaj gönderebilir. Bu durumda eğer alıcı güvenilir bir kimlik doğrulaması yapmıyorsa yanlış mesajlarla kandırılabilir.



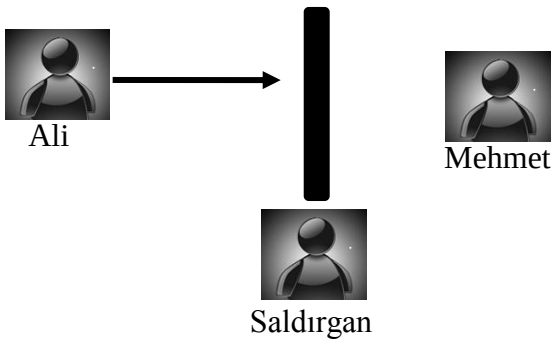

**Şekil 2.3 Kimlik Doğrulama İhlali**

- ✓ **İnkar Edememezlik İhlali:** Mesajı gönderen veya alan tarafın sonradan gerçekleştirdikleri eylemi inkar etmeleridir.




**Şekil 2.4 İnkar Edememezlik İhlali**

- ✓ **Süreklilik İhlali:** Saldırgan, haberleşmede kullanılan iki taraf arasındaki hattı veya iki taraf arasındaki haberleşme araçlarını kullanılamaz hale getirerek haberleşmenin sürekliliğini engellemeye çalışır.




**Şekil 2.5 Süreklilik İhlali**

Günümüzde, gerçekleştirilen güvenlik saldırıları, pasif ve aktif saldırılar şeklindedir. Pasif bir saldırı bir sistemden bilgiyi öğrenmeyi ya da kullanmayı dener ancak sistem kaynaklarını etkilemez. Aktif bir saldırı ise sistem kaynaklarını değiştirmeyi ya da onların işlemlerini etkilemeyi dener.

- ✓ **Pasif Saldırılar**

Pasif saldırılar, iletimlerin gizli dinlenmesi ya da görüntülenmesi doğasını taşırlar. Saldırganın amacı iletilen bilgiyi elde etmektir. Pasif saldırılar iki şekilde gerçekleştirilir: mesaj içeriklerini yayınlamak ve akış analizidir.

Pasif saldırılar, verinin herhangi bir değişimine yol açmamaları sebebiyle tespit edilmeleri çok zordur. Tipik olarak, görünüşte mesaj akışı normal bir şekilde gönderilir ve alınır. Ne gönderici ne de alıcı bir üçüncü tarafın mesajları okuduğu ya da akış şeklini gözlemlediği konusunda uyarılır. Bu saldırıları önlemek için şifreleme yöntemleri kullanılır [7].

- ✓ **Aktif Saldırılar**

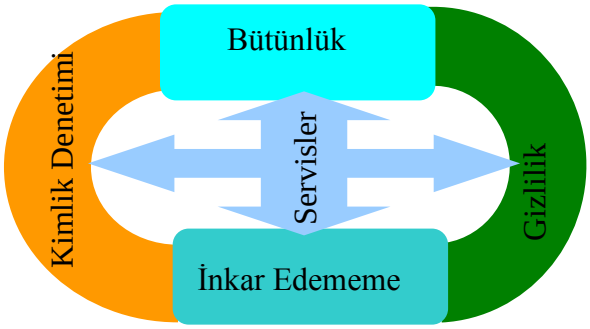
Aktif saldırılar, veri akışının belli bir değişimini ya da sahte bir akışının oluşturulmasını içerir ve dört farklı kategoriye bölünebilir: gerçeğin gizlenmesi, tekrarlama, mesajların değişimi ve hizmetin reddi.

Aktif saldırılar pasif saldırıların zıt özelliklerini taşımaktadırlar. Pasif saldırıların tespit edilebilmesi her ne kadar zor olsa da başarılarını önlemek amacıyla tedbirler mevcuttur. Diğer taraftan aktif saldırıları tamamıyla önlemek fiziksel, yazılım ve ağ korunmasızlıklarının var olan geniş çeşidi nedeniyle oldukça zordur. Bu saldırılara karşı amaç, yapılan saldırıları tespit etmek ve onlar tarafından neden olunan herhangi bir aksama ya da gecikmeyi düzeltmektir [7].

Haberleşen iki tarafın güvenlikle ilgili çeşitli beklentileri vardır. Bu beklentiler dört servis üzerinde yoğunlaştırılabilir [7]:

- ✓ **Gizlilik (Veri Güvenliği-Confidentiality)**: Bilgi, istenmeyen kişiler tarafından anlaşılamamalıdır. Veri güvenliği sağlamanın; fiziksel korumalardan matematik algoritmalara kadar birçok yöntemi bulunmaktadır.
- ✓ **Veri Bütünlüğü (Data Integrity)**: Bilginin içeriğinin iletimde değiştirilememesidir. Veri bütünlüğünü garanti etmek için yetkisiz kişilerin kullanımını ortaya çıkaracak yeteneğe sahip olunmalıdır. Veri kullanımı ekleme, silme ve değiştirme gibi işlemleri içerir.

- ✓ **Reddedilemezlik (İnkar Edememe- Non Repudiation)**: Bilgiyi gönderen veya isleyen kişinin yaptığı işi sonradan inkar edememesidir.
- ✓ **Kimlik Doğrulama (Kanıtlama-Authentication)**: Bir iletişim içine giren kişilerin kimliklerinin doğruluğundan emin olunmasıdır. Kanıtlama işlemi gönderilen bilginin doğruluğunu, bilginin; kaynağı, başlangıç tarihi, veri içeriği, gönderme zamanı gibi özellikleri dikkate alınarak yapılmalıdır.




**Şekil 2.6 Servisler**

Teknolojinin yoğun bir şekilde iş ve ticaret hayatında kullanılmasıyla birlikte, kullanıcıların kullanılan bu teknolojilere güvenlerini sağlamanın yolu etkin bir şifreleme yönteminin kullanılmasıdır.

## 2.3. Güvenli Şifreleme Yöntemleri

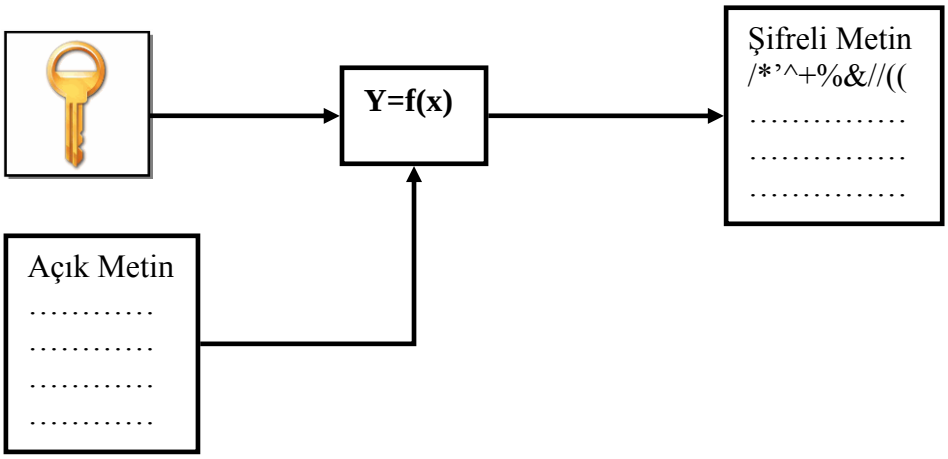
Şifreleme, gizlenmek istenen bilginin bir anahtarla veya özel bir yöntemle bir dizi değişiklikten sonra farklı bir şekle sokulması olarak tanımlanabilir. Şifreleme sonucu oluşan gizli veri, şifre çözme işlemine tabi tutularak ilk haline dönüştürülmelidir.

Şifreleme yönteminde aranan özellikler, aşağıdaki gibi listelenebilir [6]:

- ✓ Daha verimli bir şifreleme ve şifre çözme işlemi için şifrelenecek verinin güvenliği, seçilecek şifreleme algoritması ile doğru orantılı olmalıdır.
- ✓ Herhangi bir koşula bağlı anahtar seçimi ve şifreleme algoritması seçilmemeli, seçilen şifreleme algoritması her türlü bilgi için aynı adımları uygulamalıdır.
- ✓ Karışık sistemlerin gerçeklenmesi performans açısından tatmin edici olmayacağından, şifreleme ve şifre çözme işlemi olduğunca basit olmalıdır.
- ✓ Mesajın bütünlüğünü korumak için şifrelemede meydana gelen herhangi bir hata sonraki adımlara yansıtılmamalıdır.
- ✓ Kullanılacak şifreleme algoritmasıyla şifrelenen mesaj ile açık mesaj arasında herhangi bir ilişki kurulması zor olmalıdır.

- ✓ Açık mesaj, şifreli mesaja dönüştürülürken, kelime ve harf grupları şifreli mesaj içerisinde, kullanılacak algoritmanın dağıtma özelliği ile olabildiğince dağıtılmalıdır.

Güvenli şifreleme yöntemleri, kriptoanalize karşı güçlü olan algoritmalar kullanılarak gerçekleştirilir. Ayrıca bu güvenli yöntemler klasik şifreleme yöntemlerinin zayıf yönlerini ortadan kaldırır. Şifreleme ve şifre çözme yöntemleri elektronik ortamda ikili düzende saklanan veriler üzerinde uygulandığından, bu yöntemlerde "bit dizilerinden" oluşturulan anahtarlar kullanılır [6].




**Şekil 2.7 Şifreleme**

Anahtar uzunluğu bir şifreleme algoritmasının güvenliğini belirleyen en önemli faktördür [5]. Bir şifreleme algoritmasında kullanılan anahtarın uzunluğu, güvenlikle doğru orantılıdır. Kullanılan şifreleme algoritması için toplam anahtar sayısının çokluğu; şifrelemede, bu anahtarlardan herhangi biri kullanılacağı için anahtarın tahmin yoluyla elde etme olasılığı çok düşüktür.

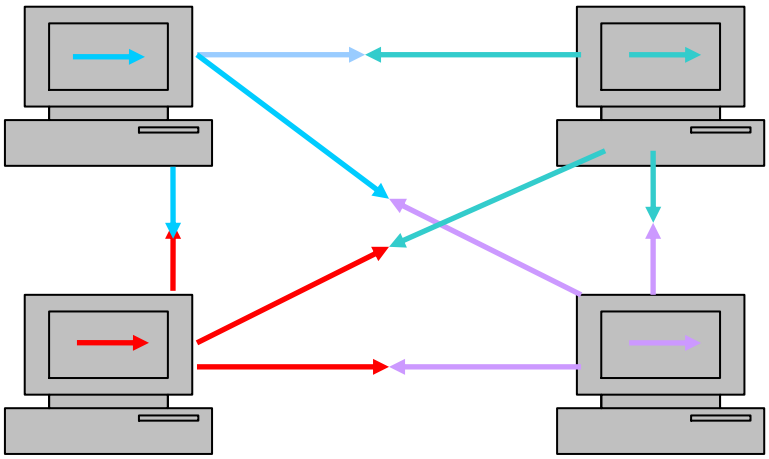
Güvenli şifreleme temel olarak iki çeşittir [6]:

1. **Simetrik Şifreleme**
2. **Asimetrik Şifreleme**

## 2.4. Simetrik Şifreleme Sistemleri

Simetrik şifreleme algoritmalarını kullanarak haberleşecek taraflar, şifreleme ve şifre çözme işlemi için gizli bir anahtar belirlerler. Bu taraflar, bu belirledikleri gizli anahtarın, başka tarafların eline geçmemesi için güvenli bir kanaldan birbirleri ile paylaşmaları gerekmektedir [6].

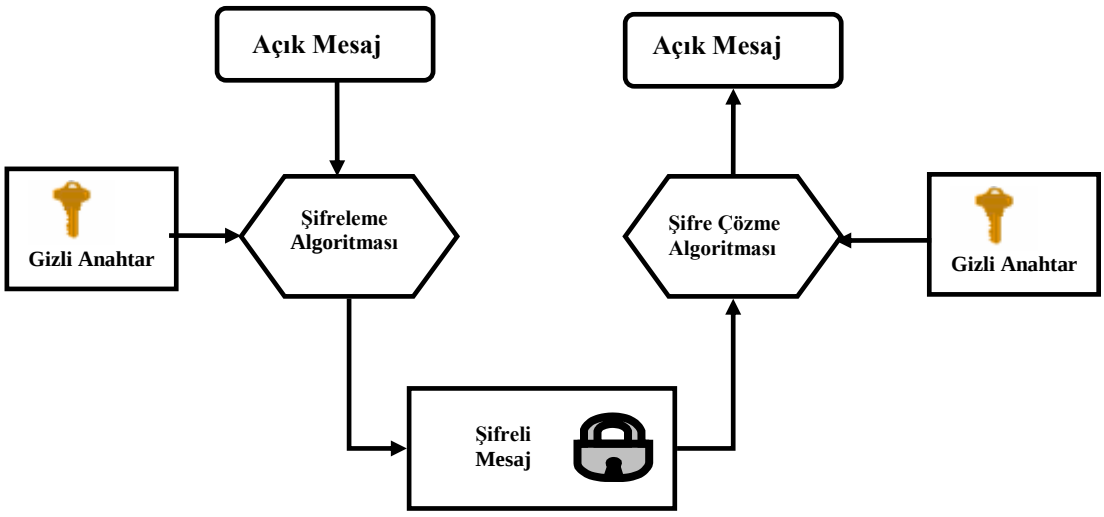
Bu şifrelemede “n” adet uç arasında şifreli iletişimin yapılabilmesi için iletişim öncesinde $\frac{nx(n-1)}{2}$ adet anahtarın iletilmesine gereksinim vardır.



**Şekil 2.8 Simetrik Şifreleme**

Simetrik şifreleme kullanan tarafların;

- ✓ Algoritmaları aynıdır.
- ✓ Algoritmaları birbirine uyumludur.
- ✓ Kullandıkları anahtarlar aynıdır.




**Şekil 2.9 Simetrik şifreleme Yöntemi**

Simetrik Şifreleme Algoritmasının Avantajları:

- ✓ Hızlıdır.
- ✓ Donanımla gerçeklenmesi kolaydır.
- ✓ "Gizlilik" hizmetini sağlarlar.

Simetrik Şifreleme Algoritmasının Dezavantajları:

- ✓ Anahtar dağıtımı zordur.
- ✓ Bu şifrelemede "Bütünlük" ve "Kimlik Doğrulama" hizmetlerini sağlamak zordur.

Simetrik şifrelemede amaç, bilginin sadece gizliliğini sağlamaktadır. Bu nedenle de günümüzde önem kazanan; sayısal imza, kimlik denetimi ve inkar edememe gibi özelliklerin gerekli olduğu alanlarda simetrik şifreleme oldukça yetersiz kalmaktadır [6].

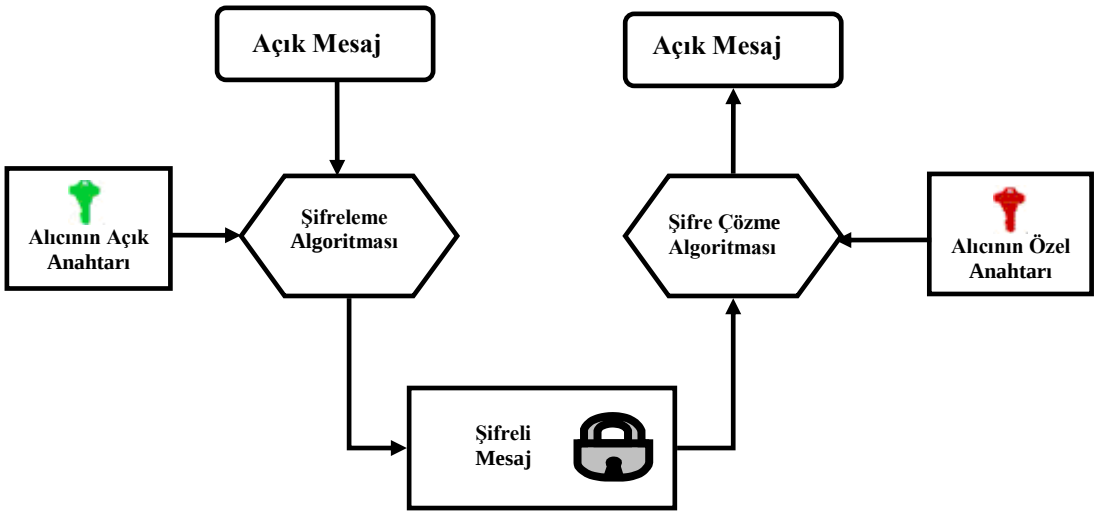
## 2.5. Asimetrik (Açık Anahtar) Şifreleme Sistemleri

Açık anahtar şifreleme algoritmalarında haberleşecek taraflar, şifreleme ve şifre çözme işlemi için "açık" ve "özel" anahtar adı verilen iki anahtar çiftini kullanırlar. Bu şifreleme algoritmasında kullanılan "açık" anahtar, başka kişilerle paylaşılabildiği için açık anahtar şifreleme sistemi olarak adlandırılır. Ancak "özel" anahtar adından da anlaşılacağı üzere gizli tutulmalıdır. Açık anahtar şifreleme yöntemini kullanan taraflar, haberleşmede aynı şifreleme algoritmasını kullanırlar [6].

Açık anahtar şifrelemede anahtar yönetimi çok önemli olduğundan, anahtar yönetiminde dikkat edilmesi gereken noktalar şu şekilde sıralanabilir [6]:

- ✓ Güvenli bir merkez tarafından dağıtılan açık anahtar değiştirilmemelidir.
- ✓ Anahtar üretimi esnek olmalıdır. Taraflar kendilerine ait anahtar çiftini üretebildiği gibi güvenilir bir merkez tarafından da üretilebilmedir.
- ✓ Haberleşen taraflara kullanılmayan anahtarların bir listesi iletilmeli ve anahtar iptalleri belli bir çerçevede gerçekleştirilmelidir.

Simetrik şifrelemede anahtar yönetimi açık anahtar şifrelemeye göre daha zordur. Açık anahtar şifrelemede, bir kullanıcıyla şifreli haberleşmek isteyen kişi karşı tarafın açık anahtarına ihtiyaç duyar. Açık anahtar şifrelemede, anahtar, kamuya açık olarak yayınlanmaktadır. Bu nedenle sisteme yeni giren bir kişi sadece bir anahtar çifti üretecektir [6].




**Şekil 2.10 Açık Anahtar (Asimetrik) Şifreleme Sistemi**

Açık anahtar şifrelemenin kuvvetli tarafları aşağıdaki gibi özetlenebilir:

- ✓ Anahtar yönetimi ölçeklenebilirdir.
- ✓ Kırılmaları oldukça zordur.
- ✓ Asimetrik şifreleme algoritmaları ile bütünlük, kimlik doğrulama ve inkar edememezlik güvenlik hizmetleri gereksinimleri gerçekleştirilebilmektedir.

Açık anahtar şifrelemenin zayıf yönleri ise aşağıdaki gibidir:

- ✓ Simetrik kriptografi algoritmalarına oranla 1500 kat daha yavaştırlar.
- ✓ Anahtar uzunluğu mobil cihaz gibi ortamlarda kullanmak için elverişli değildir.

Açık anahtar şifreleme algoritmasında, iletişimde bulunacak her kullanıcının bir anahtar çifti kullanması gerekir. Bundan dolayı bu algoritmadaki anahtar çifti sayısı "n" kullanıcı için "n" adettir. Bu algoritma ile haberleşecek 60 kullanıcı için 60 anahtar çiftine ihtiyaç vardır [6].

## 2.6. Simetrik ve Açık Anahtar Şifreleme Algoritmalarının Karşılaştırılması

- ✓ Simetrik şifrelemede, aynı algoritma ve anahtar ile şifreleme, şifre çözme işlemleri gerçekleştirilir. Açık anahtar şifrelemede ise aynı algoritma için anahtarlardan sadece biri kullanılır. Şifrelemede kullanılan anahtar şifre çözme işleminde kullanılmaz.

- ✓ Simetrik şifrelemede gönderici ve alıcı tarafları hem algoritmayı hem de anahtarı paylaşmalıdır. Açık anahtar şifrelemede ise gönderici ve alıcının ilişkili anahtarlardan birine sahip olması yeterlidir.
- ✓ Simetrik şifrelemede, şifrelemede ve şifre çözme işleminde kullanılan anahtar gizli tutulmalıdır. Açık anahtar şifrelemede, anahtarlardan biri gizli tutulmalıdır.
- ✓ Simetrik şifrelemede, şifreli metin örnekleriyle ve algoritmadan anahtar tespit edilememelidir. Açık anahtar şifrelemede ise şifreli metin örnekleri, algoritma ve anahtarlardan birini bilmekle diğer anahtar tespit edilememelidir.

## 2.7. RSA Algoritması

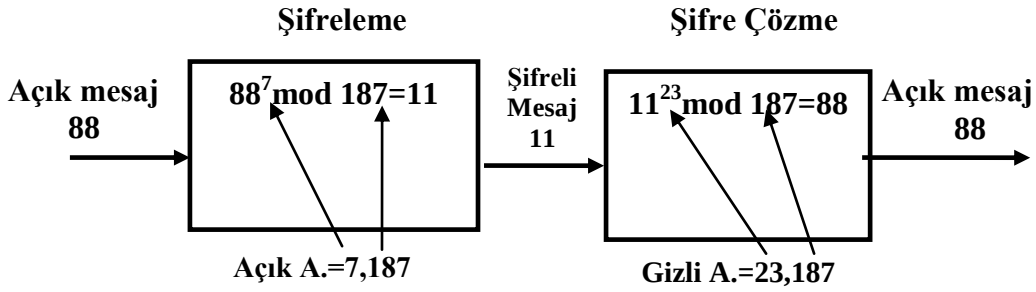
1978 yılında "Dijital imza elde etme metodu ve açık anahtarlı şifreleme sistemleri" adıyla yayınlanan makalede, R. Ronald, A. Shamir ve L. Adleman, adını kendi soyadlarının baş harflerinden alan RSA şifreleme sistemini açıkladılar. RSA şifreleme sisteminde haberleşen taraflar arasında bilinen veya belli bir yöntemle oluşturulan açık bir anahtar kullanılır. Ancak simetrik şifrelemede olduğu gibi, RSA şifreleme sisteminde kullanılan açık anahtara sahip olmak şifre çözme anahtarını ortaya çıkarmaz.

Hem gizlilik hem de dijital imza sağlamak amaçlı kullanılabilir. Bu sistemin güvenliği tamsayılarda çarpanlara ayırma probleminin kolaylıkla olmaması temeline dayanır [8]. RSA şifreleme sisteminde haberleşen tarafların şifreli mesajları göndermesi için, birbirlerinin açık anahtarlarına ihtiyaçları vardır. RSA şifreleme sisteminde, Şifre çözme işleminde gizli bir anahtar kullanılır. Anahtar oluşturma algoritması şu şekildedir.

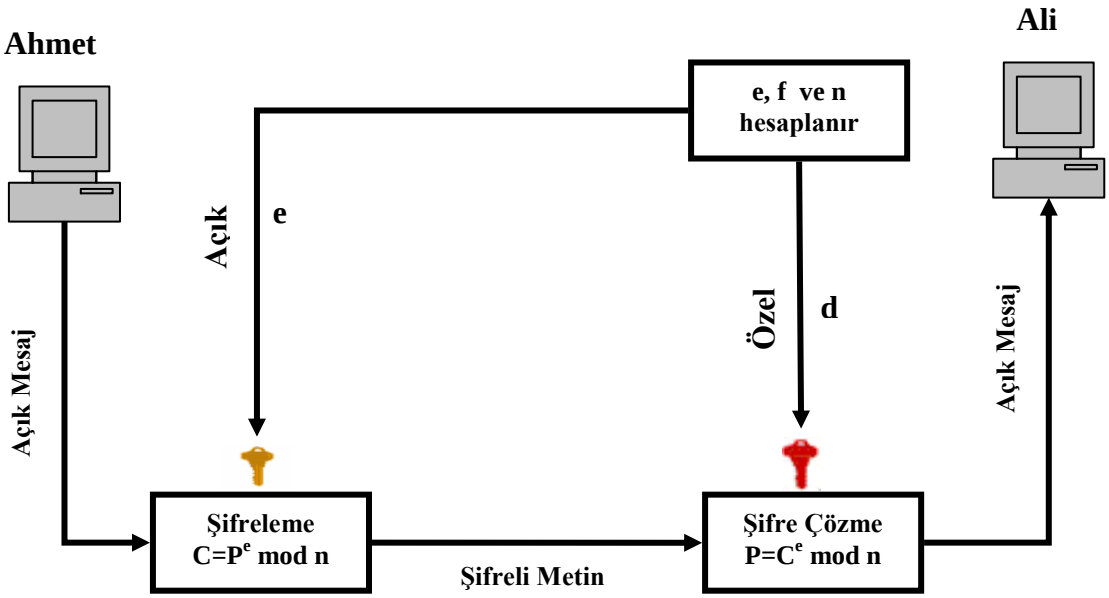
Haberleşen taraflardan her biri (A şahsı) anahtarını şu şekilde oluşturur [8–10]:

- ✓ İki tane farklı, rasgele ve yaklaşık aynı uzunlukta olan p ve q asal sayıları seçer.
- ✓ $n = pq$ ve $\varphi = (p-1)(q-1)$ değerleri hesaplanır.
- ✓ $1 < e < \varphi$ ve $\gcd(e, \varphi) = 1$ olacak şekilde rasgele bir e sayısı seçilir.
- ✓ Öklid algoritmasını kullanarak, $1 < d < \varphi$ ve $ed \equiv 1 \pmod{\varphi}$ koşulunu sağlayan d sayısı hesaplanır.
- ✓ A 'nın açık anahtarı (n, e); A 'nın gizli anahtarı ise "d" olur.

RSA şifreleme algoritmasında anahtar oluşumunda e ve d tamsayıları, şifreleme ve şifre çözme üssünü ve n ise mod sayısını göstermektedir.




**Şekil 2.11 RSA Şifreleme Örneği**




**Şekil 2.12 RSA Yapısı**

### 2.7.1. RSA 'nın Güvenliği

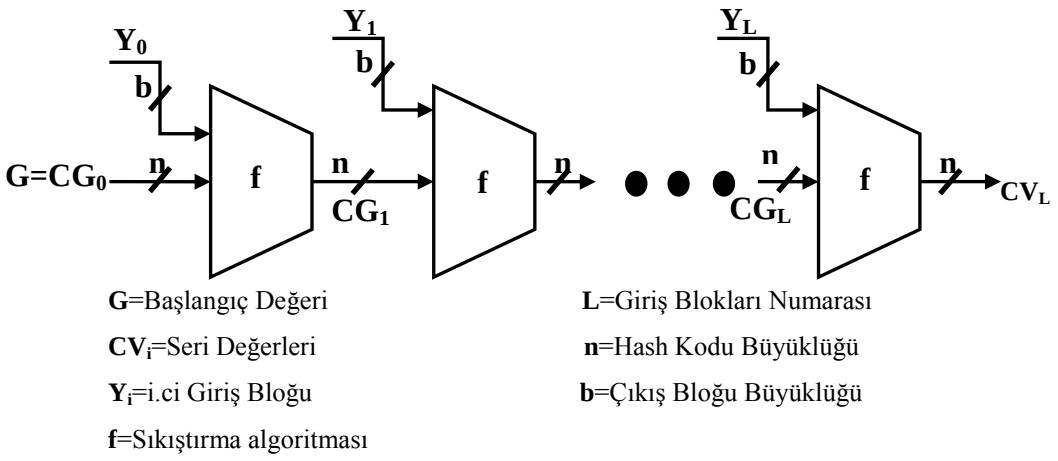
RSA algoritmasına saldırmak için kullanılabilecek üç metot aşağıdaki gibidir [10]:

- **Kaba Kuvvet (Brute-force):** Bütün özel anahtarların denenmesi ile gerçekleştirilir.
- **Matematik Ataklar:** Birkaç yöntem vardır, hepsinin amacı çarpımı oluşturan iki asal sayıyı bulmaktır.
- **Zaman Atakları:** Deşifreleme algoritmasının çalışması esnasında geçen zamana bağlıdır.

- **Seçilmiş Şifreli Metin Atağı:** Bu tip bir atak RSA ’nın özelliklerini kötüye kullanır. Kaba kuvvet saldırılarına karşı RSA’nın savunması diğer şifreleme sistemlerin çözümünden farklı değildir, çözüm geniş anahtar uzayı kullanmaktır.

## 3. TEK YÖNLÜ FONKSİYONLAR

Tek yönlü fonksiyonların asıl amacı, iletilecek dosyada, mesajda veya verinin bir bloğunda parmak izi bırakmaktır. Tek yönlü fonksiyonlarda, tek yönde çözümü hesaplamak çok kısa sürmesine rağmen aksi yönde çözümü hesaplamak oldukça zordur [7].




**Şekil 3.1 Tek Yönlü Fonksiyon Genel Yapısı**

Şekil 3.1 'de tek yönlü bir fonksiyonun genel yapısı verilmiştir. "Merkle" tarafından önerilen bu yapı, tekrarlanan tek yönlü fonksiyon olarak adlandırılır. Tek yönlü fonksiyon, bir giriş mesajı alır ve her biri "b" bit olan L sabit büyüklükteki bloklara ayrılır. Gerekirse son blok "b" bit olarak doldurulur. Son blokta tek yönlü fonksiyon, mesaj toplam uzunluk değerini içerir. Bu uzunluk bir saldırgan için elde edilebilecek uzunluktan daha kapsamlıdır. Saldırgan ya özetleri aynı değerde, aynı uzunlukta veya onların boy değerleri ile aynı değerdeki özet ile birlikte iki mesaj bulmalıdır [7].

Tek yönlü fonksiyonlar, tekrarlanan "f" sıkıştırma fonksiyonunu içeren iki giriş alır ve "n" bit çıkış üretir. Özet başında bir algoritma parçası olarak belirtilen değişken zincirlemenin başlangıç değeri vardır. Zincirleme değişkeninin son değeri özet değeridir, çoğu kez b>n olarak özetlenir. Tek yönlü fonksiyon aşağıdaki şekilde özetlenebilir:

**$CG_0$= ilk n-bit lik değer, $CG_i = f(CG_{i1}, Y_{i1})$ 1<= i <= L, , H(M) = $CG_L$**

Burada tek yönlü fonksiyonlarının girişi $Y_0$, $Y_1$,..., $Y_L$ bloğundan oluşan "M" mesajıdır. Son yıllarda, tek yönlü fonksiyonlar üzerinde kriptoanalitik ataklar geliştirilmiş ve bu ataklar sonucu önemli başarılar elde edilmiştir. Tek yönlü fonksiyonlarının kriptanalizi "f" fonksiyonunun içyapısı üzerine odaklanır ve bu f fonksiyonunun tek bir

uygulaması için çarpışmalar üretmek amacıyla verimli teknikler aramaya dayanır. Bir kez yapıldığında, saldırgan "G" 'nin sabit değerini dikkate almalıdır. "f" fonksiyonu üzerindeki saldırı içyapısındaki istismara bağlıdır. Tipik olarak, simetrik blok şifrelerde olduğu gibi, "f" fonksiyonu bir dizi işlem tamamlanarak meydana gelir [7].

### 3.1. Özetleme (Özet) Fonksiyonları

Özetleme fonksiyonları, iletilecek dosyada, mesajda veya verinin bir bloğunda parmak izi bırakarak alıcının aldığı mesajın iletim esnasında herhangi bir değişikliğe uğramadığının garantisi için kriptolojide en çok kullanılan fonksiyonlardır [11–13]. Bu özelliklerinden dolayı özetleme fonksiyonları günümüz kriptolojisinde önemli bir yere sahiptir.

Özetleme fonksiyonları "M" değişken uzunluktaki herhangi bir mesajı alarak sabit uzunlukta bir özet çıktısı üretir. Mesaj doğrulamada özet fonksiyonu olan "H" 'nin verimli bir şekilde kullanılabilmesi için, aşağıdaki özellikleri taşıması gerekir [7]:

1. H farklı büyüklükteki mesajlara uygulanabilir olmalıdır.
2. H farklı büyüklükteki her girdi için sabit bir çıkış üretmelidir.
3. H(x), her x girdisi için kolayca hesaplanabilir olmalıdır, böylece yazılımsal ve donanımsal uygulamalarda pratiklik kazandırır.
4. Herhangi bir h için, H(x)=h olacak şekilde bir x değerinin hesaplanabilmesi zor olmalıdır. Bu özet fonksiyonunun tek yön özelliğidir. Hattı dinleyen bir saldırgan elde ettiği özet değerini kullanarak fonksiyonu tersi yönde işletip özet fonksiyonunun güvenliğini ortadan kaldırabilir.
5. x 'in bilinen değerleri için, $y \neq x$ ve H(y)= H(x) eşitliğini sağlayan bir y değerinin hesaplanması güç olmalıdır. Bu durum zayıf çarpışma direnci olarak adlandırılmaktadır
6. H(x)= H(y) olacak şekilde bir (x,y) çiftinin hesaplanması güç olmalıdır. Bu durum bazen güçlü çarpışma direnci anlamına da gelmektedir.

İlk üç özellik, özet fonksiyonunun mesaj doğrulamasında kullanılan pratik uygulamasıdır.

Dördüncü özellik tek yönlü özelliktir. Bu özellik bir mesaj bloğu için bir özet çıktının üretilmesinin kolay olduğunu ifade eder. Eğer doğrulama tekniği gizli bir değer içeriyorsa

bu özellik önemlidir. Çünkü gizli değerin kendisi gönderilmemektedir. Özet fonksiyon tek yönlü özelliği yok ise bir saldırgan gizli değeri rahatlıkla ele geçirebilir.

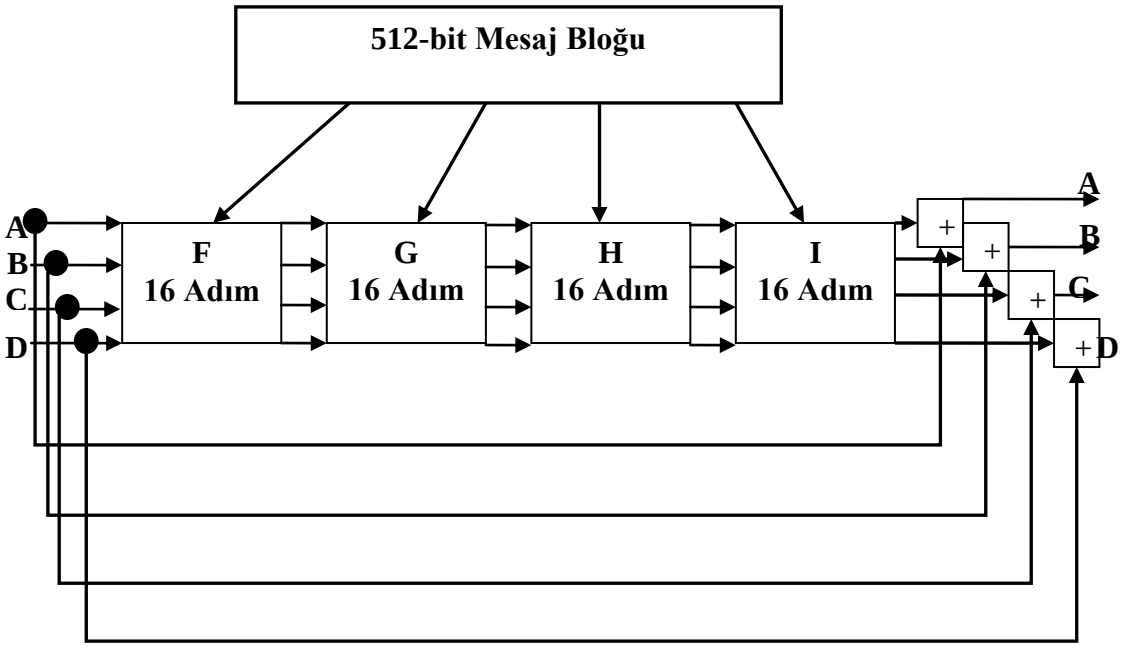
Beşinci özellik, verilen bir mesaj olarak aynı değerin özeti olan alternatifli bir mesajın bulunamayacağını garanti eder.

Altıncı özellik, bir özet fonksiyonunun, doğum günü saldırılarına karşı direncini göstermektedir.

En çok bilinen Özetleme algoritmaları MD5 ve SHA ailesidir. MD serisi özetleme algoritmaları, Ron Rivest tarafından geliştirilmiştir. 128 bit özetleme sağlamaktadır. Bu seride MD2 en yavaşı, MD4 en hızlı olanıdır. MD5, MD4 'e göre daha kapsamlı geliştirildiğinden hızı daha düşüktür ancak daha güvenilirdir.

MD5 Özetleme Fonksiyonunun Temel Adımları:

- ✓ MD5, veriyi 512-bit 'lik bloklara ayırır ve her bir blok üzerinde aynı işlem uygulanır.
- ✓ Üzerinde işlem yapılacak verinin 512 'nin katı olması gerekmektedir, fakat gerçek verimiz bu özelliği sağlamayabilir. Bu sorunu çözmek için ekleme (padding) işlemi uygulanır.
- ✓ Ekleme işleminde şu kural gözetilir: Verinin uzunluğu 512 nin en yakın katından 64 bit eksik olacak şekilde, verinin sonuna bir adet "1" ve geri kalanlar için ise "0" eklenir. Bu 64-bit 'lik fark verinin uzunluğunu belirtmekte kullanılır.
- ✓ Ekleme işleminden sonra, MD5 veriyi işlemeye başlar.




**Şekil 3.2 MD5 Ana Döngüsü**

MD5 döngüsünün başlangıcında 32-bitlik dört tane (A,B,C,D) değişken bulunur. Başlangıçta bu değişkenlerin değerleri sabittir ve her 512-bit 'lik bloğu işleme soktukça bu değişkenlerin değerleri değişir ve en sondaki bloğuda islendikten sonra elde edilen A,B,C ve D değişkenlerinin değerlerini yan yana dizildiğinde (A-B-C-D) 128-bit 'lik MD5 özet sonucu elde edilir.

Burada F-G-H-I adımları görünmektedir. Her adımın önceden tanımlı ve kendisine özgü birer fonksiyonu bulunmaktadır ve bu fonksiyonlar her adımda 16 kez çağırılarak elde edilen sonuç bir sonraki adıma iletilir. Her bir 512-bit lik blok için MD5 algoritması 4 adım *x* 16 işlem = 64 adet işlem yapmaktadır. Özetleme fonksiyonunda bu kadar fazla adımın amacı, simetriği engelleyip, dağıtma ve dağılma özelliğinin kullanılarak farklı girdiler için farklı sonuçlar üretilebilmesini sağlamaktır.

MD5 güvenilir olarak kabul edilmiş olmasına rağmen yapılan son araştırmalar MD5 'inde kırılabileceğini göstermiştir. Bu nedenle MD5 'e duyulan güvende kaybolmuştur.

Diğer özetleme algoritmalarından biri olan SHA–1 (Secure Hahshing Algorithm), NSA (National Security Agency) tarafından geliştirilmiştir. SHA–1, MD algoritmalarına göre daha uzun özet çıkışı üretebilmektedir. Bu çıkış özet uzunluğunda üretilen bir dizi için gerekli süre MD5 algoritmasından %25 daha yavaş olsa da SHA–1 algoritmasının kullanılması tavsiye edilmektedir. İlk sürüm SHA–0 olarak adlandırılmıştır.

SHA–0 ve SHA–1 160 bitlik özet değeri üretir. Daha sonra SHA ’nın 256, 384 ve 512 bit sürümleri çıkarılmıştır [14]. Ancak Bruce Schneier bloğunda, Sandung Üniversitesinden bir grup araştırmacı, SHA–1 algoritmasının kırıldığını belirtmiştir [15]. SHA–0 ve SHA–1 ’e yönelik daha önceki kırma girişimlerinden üretilen bu yeni metot, büyük bir kriptoanalitik sonuç olarak belirtilmiştir. En son güncel versiyonu ise SHA–2 (SHA–224, SHA–256, SHA– 384, SHA–512) olarak kullanılmaya başlanmıştır.

### 3.2. MD5 - SHA–1 Özetleme Fonksiyonlarının Karşılaştırılması

- MD5 ’in çıktısı 128-bit iken, SHA ’nin çıktısı 160-bit ’tir. Yani MD5 ’te 4 adet 32-bit lik değişken kullanılırken, SHA ’da 5 adet 32-bitlik değişken kullanılır.
- Her ikisi de 512-bit lik bloklar üzerinde işlem yaparlar.
- SHA ’da ekleme (padding) işlemi, MD5 ’taki ile ayni şekilde yapılır.
- SHA ’da her 512-bit lik blok için 4 adımda işlemler yapılır, fakat bir farkla: MD5 ’te her adımda önceden tanımlı fonksiyonların kullanımı 16 kez tekrarlanırken, bu sayı SHA ’da 20 dir.
- SHA girdi olarak maksimum $2^{64}$-1 uzunluğunda veriyi kabul eder. Bunu yanında MD5 için böyle bir kısıt yoktur.
- SHA ürettiği 160-bit lik sonuç ile kaba kuvvet saldırılarına karşı daha dayanıklıdır.

## 4. MESAJ DOĞRULAMA KODLARI (MAC)

Kriptografik şifrelemenin toplam kontrolü olarak bilinen mesaj doğrulama kodu (MAC), C fonksiyonundan aşağıda gösterildiği gibi türetilmektedir:

MAC = C(K, M)

MAC bilgisi mesajın hatasız olduğu tespit edildiğinde mesaja eklenir. Mesajı kabul eden taraf da doğrulama için gerekeli bilgileri girerek MAC engelini aşar ve mesajı görüntüleyebilir.

Bu teknik, iki iletişim tarafına, biri A diğeri B olan iki taraf, ortak gizli anahtar olarak K 'yı paylaşır. A, B 'ye mesaj gönderdiği zaman, MAC mesajın işlevi olarak hesaplanır ve anahtar: MAC = C(K, M) olarak bulunur;

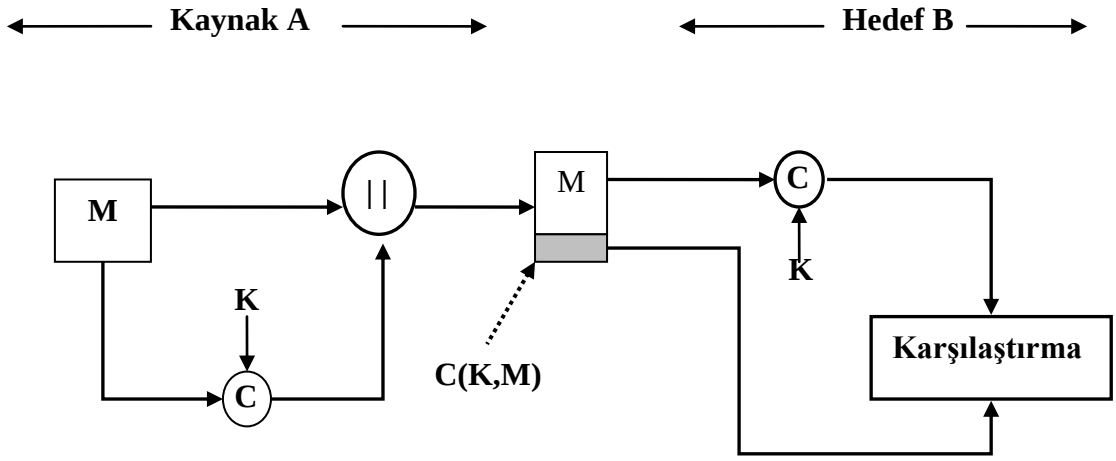
M= giriş mesajı, C= MAC işlevi, K= paylaşılan gizli anahtar, MAC= mesaj doğrulama kodu;

MAC eklenmiş mesaj, planlanmış alıcıya aktarılır. Alıcı, alınan mesaj üzerinde aynı gizli anahtarı kullanarak yeni MAC oluşturmak için aynı hesaplamayı gerçekleştirir. Alınan MAC, hesaplanan MAC ile karşılaştırılır (Şekil 4.2). Eğer alınan MAC hesaplanan MAC ile eşleşiyorsa o zaman alıcı mesajın değişmediğinden emin olur. Eğer saldırgan mesajı değiştirir fakat MAC 'ı değiştirmez ise o zaman alıcının MAC hesaplaması alınan MAC 'ten farklı olur. Çünkü saldırgan gizli anahtara sahip değildir. Saldırgan mesaj içindeki değişikliğin fark edilmemesi için MAC 'ı değiştirmez [7].

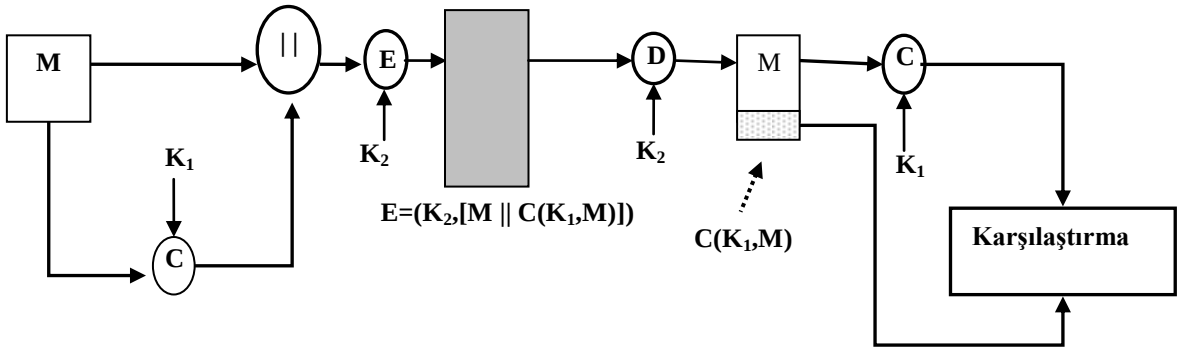
MAC 'ın işlevi şifrelemeye benzer. Genel olarak MAC 'ın birden fazla işlevi vardır. İşlevin etki alanı bazı gelişigüzel uzunluktaki mesajları kapsar, halbuki yayılma aralığı olabilecek tüm MAC ve olabilecek tüm anahtarları içerir. Ayrıca k-bit anahtarı, ikili olabilecek anahtarı göstermektedir ($2^k$).

100-bit mesaj ve 10-bit MAC kullanıldığını farz edelim. Bu durumda toplam $2^{100}$ farklı mesaj, fakat $2^{10}$ farklı MAC oluşmaktadır. Böylece, ortalama olarak tüm MAC değerleri için toplam $2^{100}/2^{10} = 2^{90}$ farklı mesaj üretilmiş olur. Eğer 5-bit anahtar kullanılırsa o zaman MAC değerlerine göre $2^5 = 32$ farklı eşleştirme mesaj seti oluşturulur.

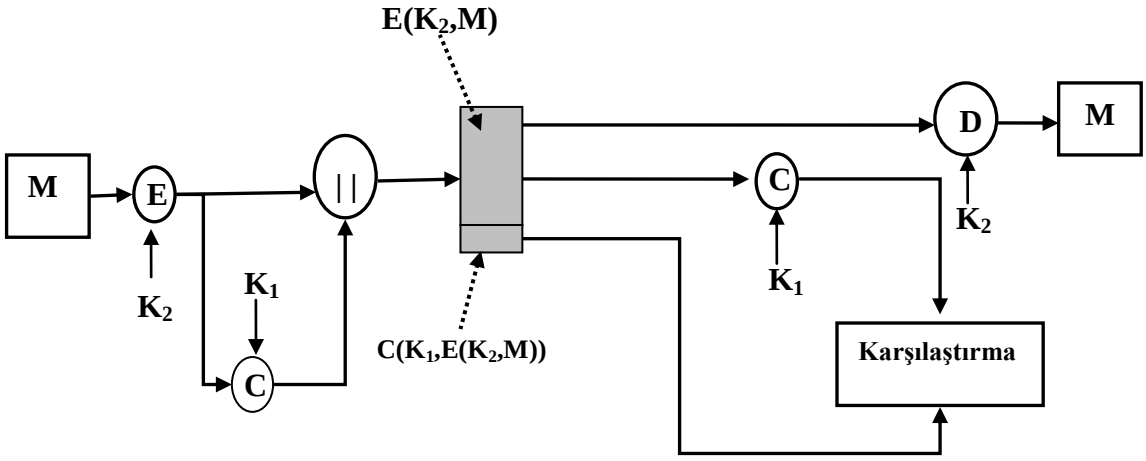
Çünkü doğrulama işlevlerindeki matematiksel özellikler şifrelemeden kırılma olarak daha hassastır [7].




**Şekil 4.1 Mesaj Doğrulama**




**Şekil 4.2 Mesaj Doğrulama ve Gizlilik: Doğrulama Şifresiz Metne Bağlanır.**




**Şekil 4.3 Mesaj Doğrulama ve Gizlilik: Doğrulama Şifreli Metne Bağlanır.**

Şekil 4.1 ’de, doğrulanma sağlanmıştır, ancak mesajın gizliliği yoktur, çünkü mesajlar açık bir şekilde aktarılmıştır. Gizlilik, Şekil 4.2 ve Şekil 4.3 ’teki gibi mesaj şifreleme oluşturulursa sağlanır. Her iki durumda da iki ayrı anahtar gereklidir. Her biri hem alıcı hem gönderici tarafından paylaşılır. Şekil 4.2 ’de MAC giriş yapılan mesaj ile hesaplanır ve mesaj ile birleştirilir, eklenmiş blok şifrelenir. Şekil 4.3 ’te ise mesaj ilk önce şifrelenir, MAC sonucu ortaya çıkan değerle hesaplanır ve şifreli metinler kullanılarak şifreli metin ile birleştirilir ve aktarılmış blok elde edilir. Genellikle şifresiz metnin doğrulanması tercih edilmekte ve Şekil 4.2 ’de gösterilen metod kullanılmaktadır. Çünkü simetrik şifreleme, ayrılmış mesaj kullanımı yerine kolayca kullanılır ve uygun doğrulamayı sağlar [7].

### 4.1. Mesaj Kimlik Doğrulaması İçin Anahtarlı Özetleme (HMAC)

Güvenilmez bir ortamda depolanan ya da iletilen bilginin bütünlüğünü kontrol edebilecek bir yöntem sağlamak, açık bilgisayar ve iletişim dünyasında başlıca bir gerekliliktir. Gizli anahtara dayalı böyle bir bütünlük kontrolü sağlayacak mekanizmalar genellikle “İleti Kimlik Doğrulama Kodları” (MAC) olarak adlandırılır. Tipik olarak, mesaj kimlik doğrulama kodları kendileri arasında geçen bilgileri onaylatmak için gizli bir anahtar paylaşan iki taraf arasında kullanılır [7].

HMAC, MD5 ve SHA–1 gibi özet fonksiyonu içinde, gizli bir anahtar ile birlikte kullanılabilir. HMAC ’ın kriptografik gücü özet fonksiyonunun özelliklerine bağlıdır.

HMAC, herhangi bir tekrarlanan kriptografik özet fonksiyonu ile birlikte kullanılabilir. MD5, SHA–1 ve Kaotik tabanlı özetleme fonksiyonları böyle özet fonksiyonlarına örnektir. HMAC, ayrıca mesaj kimlik doğrulama değerlerinin hesaplanması ve doğrulanması için gizli bir anahtar kullanır. Bunun arkasındaki temel hedefler:

- Değişiklik yapmadan, mevcut özet fonksiyonlarını kullanmak.
- Özet fonksiyonun orijinal performansını, önemli bir bozulma olmadan, korumak.
- Özet fonksiyonun altındaki makul varsayımlara dayalı kimlik doğrulama mekanizmasının gücünün kriptografik analizini iyi anlamak.
- Daha hızlı ya da daha güvenli özet fonksiyonlarının bulunması veya ihtiyacı durumunda belirlenen özet fonksiyonunun kolayca yer değiştirilebilmeleri için imkan sağlamak.

### 4.1.1. HMAC Adımları

HMAC, “H” ile gösterilen kriptografik bir özet fonksiyonu ve gizli bir anahtar olan “K” ’yı gerektirir. Burada “H” fonksiyonu, veri blokları üzerinde temel sıkıştırma fonksiyonunun yinelenerek özetlendiği kriptografik bir özet fonksiyon olarak kabul edilir. Burada “B”, blokların byte uzunluğu, “L” ile özet çıkışlarının byte uzunluğu temsil edilmektedir. Kimlik doğrulama anahtarı “K” özet fonksiyonunun blok uzunluğu olan “B” ’ye kadar herhangi bir uzunlukta olabilir. “B” byte ’larından daha uzun anahtarlar kullanan uygulamalar, ilk olarak “H” fonksiyonunu kullanan anahtarı özetleyecek ve daha sonra sonuç olan L bayt dizisini HMAC için gerçek anahtar olarak kullanacaktır. Her durumda “K” için önerilen minimum uzunluk L byte ’ tır.

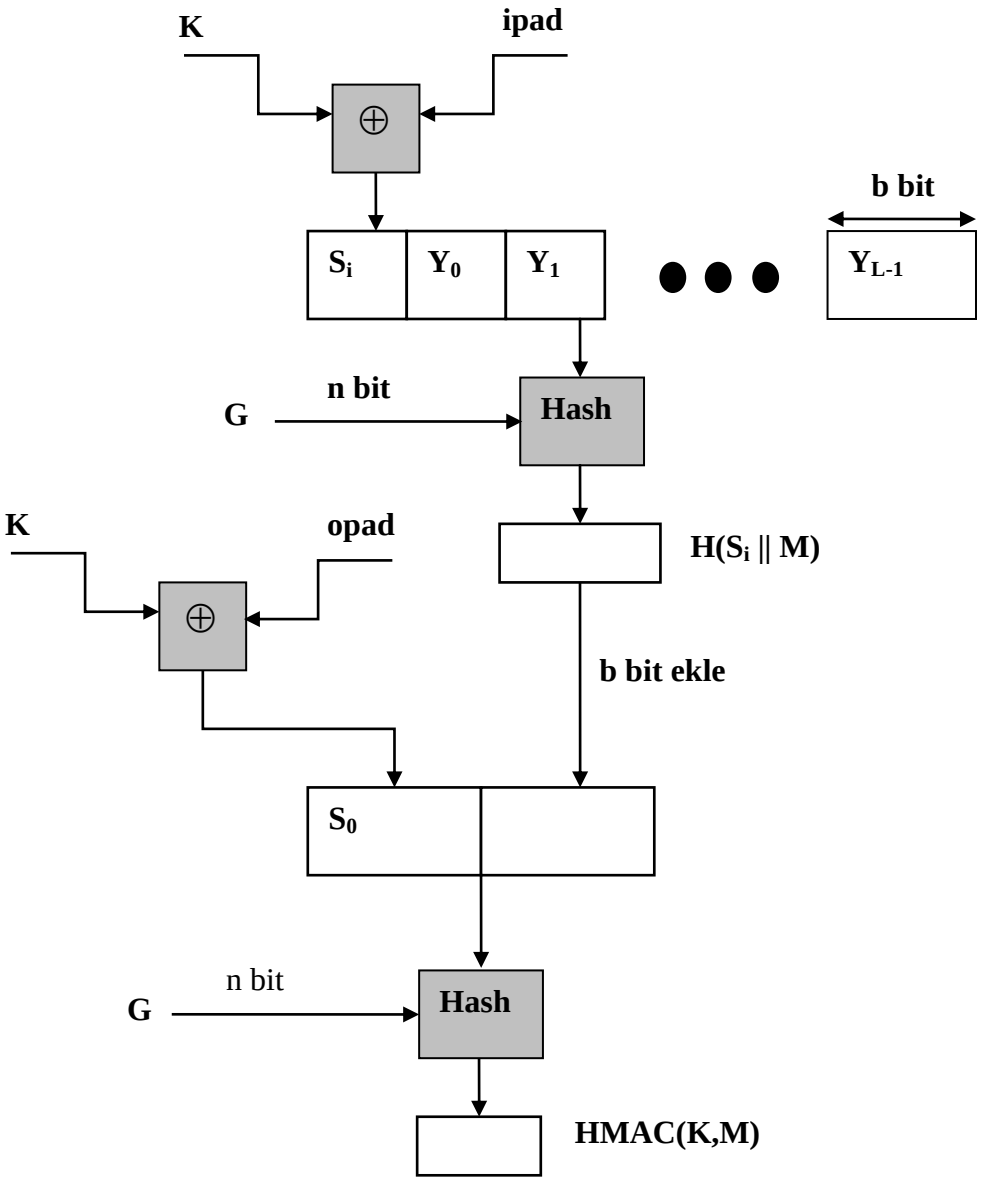
Burada, Şekil 4.4 ve Şekil 4.5 ’te olduğu gibi iki sabit ve farklı dizeleri ipad ve opad olarak tanımlanır [7].

1. ipad = byte 0x36 B kez tekrarlanan
2. opad = byte 0x5C B kez tekrarlanan

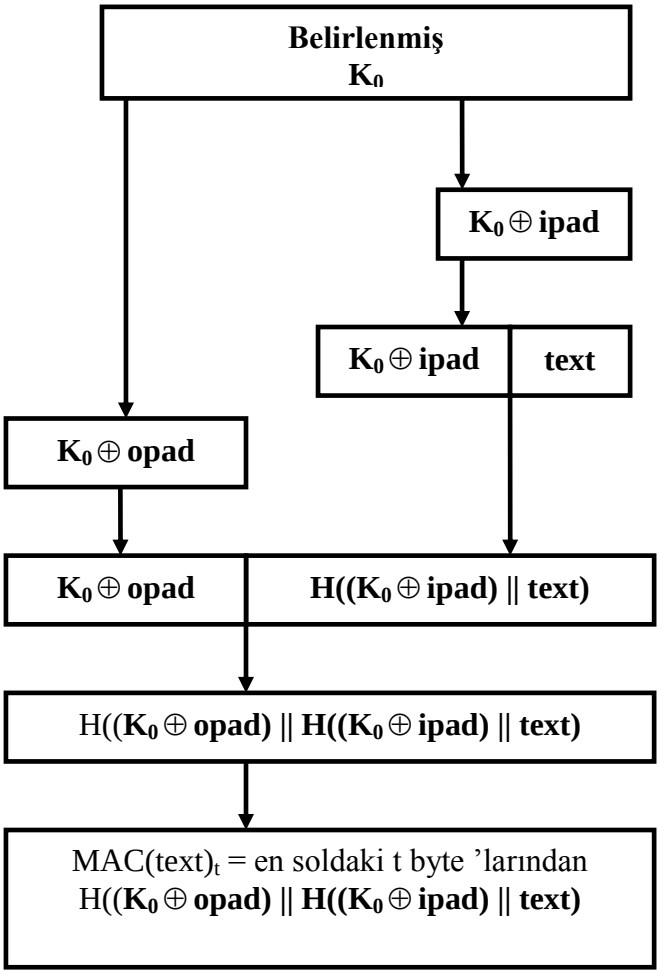
H (K XOR opad, H (K XOR ipad, metin))

Yani,

- ✓ K bayt dizesini B blok uzunluğu ile eşitlemek için “K” sonuna sıfır ekleme (örneğin “K” uzunluğu 20 byte ise ve B = 64, K ’nın sonuna 44 sıfır bayt 0x00 eklenerek)
- ✓ Anahtar, XOR (bitsel özel-OR) ipad ile
- ✓ Oluşan “B” byte karakter veriye “metin” akışını eklemek
- ✓ İpad ve metin ile oluşturulan akışa “H” uygulamak
- ✓ Anahtar, XOR (bitsel özel-OR) opad ile
- ✓ B bayt dize ile oluşturulan ipad ve metin ’e “H” uygulamak ve sonucu opad ’a eklemek.
- ✓ Bir önceki adımda çıkış olarak üretilen akışa “H” uygulamak
- ✓ Sonuç




**Şekil 4.4 HMAC Yapısı**




**Şekil 4.5 HMAC basamakları**

### 4.1.2. HMAC Anahtar Üretimi

HMAC için anahtar herhangi bir uzunlukta olabilir; ancak, "L" bayt 'tan daha az olanlar tercih edilmez, çünkü bu durum fonksiyonun güvenlik gücünü azaltacaktır. "L" baytından uzun anahtarlar kullanılabilir, ancak daha fazla uzunluk, fonksiyonun gücünü önemli derecede arttırmaz.

HMAC anahtarlar üretme yönteminde anahtar, blok uzunluktan küçük ise o zaman anahtar sonuna blokla eşitlenecek şekilde "0" eklenir. Anahtar uzunluğu blok uzunluğundan büyük olduğu zaman anahtar özet fonksiyonuna tabi tutulur. Elde edilen özetleme çıktısını bloğa eşitlemek için anahtarın sonuna "0" eklenir [16–17]. Anahtarlar periyodik olarak yenilenmelidir. Günümüz saldırıları anahtar değişiklikleri için önerilebilecek belirli bir frekans belirtmez. Çünkü bu saldırılar pratik olarak mümkün değildir, ancak periyodik anahtar yenileme fonksiyonun ve anahtarların potansiyel zayıflığına karşı temel bir güvenlik uygulamasına maruz kalan bir anahtarın zararını sınırlandırır.

## 5. KAOTİK TEMELLİ ÖZETLEME FONKSİYONLARI

Kriptografik özetleme fonksiyonu, bilgi güvenliği için basit bir tekniktir ve modern kriptografide önemli bir rol oynar. Özetleme fonksiyonu "H", girdi olarak herhangi bir uzunluktaki mesajı alır ve sabit uzunlukta mesaj özeti üretir.

Özetleme fonksiyonları da dahil olmak üzere şifreleme algoritmaları için karışıklık ve difüzyon iki temel kriterdir. Difüzyon, birçok şifreli bit üzerinden bir tek düz sembol etkisiyle yayılacak şekilde düz istatistiksel yapının oluşmaması ve görüntülenememesi için kullanılır. Karışıklık ise düz istatistikleri şifreli istatistiklerin bağımlılığını karıştırdığı dönüşümler olarak kullanmak anlamına gelir [18].

Özetleme fonksiyonlarının çoğu, orijinal mesajın rastgele bir sürecinde ileti özeti üretir. Daha sonra, kaos sistemi de rastgele bir davranış oluşturur, fakat aynı zamanda kaos sistemi tamamen belirleyicidir. Kaos, dünyada her zaman mevcut olan deterministik süreçtir. Onun rastgele hareketi nedeniyle, parametre değerleri ve başlangıç koşulları için duyarlılık, ergodiklik, karışıklık ve difüzyon özellikleri; kaotik kriptografiyi modern şifrelemenin önemli bir dalı haline getirmiştir. Özetleme değerleri tekrarlanan çadır harita tarafından elde edilir. Kaostan tam olarak yararlanmak için doğrusalsızlık, ergodiklik gibi tasarıyı karıştırmak özellikleri kullanılır. Kaosun bu özellikleri özetleme fonksiyonunun tüm performans gereksinimlerini etkin ve esnek bir şekilde yerine getirir [19–20].

Mevcut kaos tabanlı özetleme fonksiyonlarının genel yönetimini şu şekildedir [20]:

- Sabit uzunlukta işleme birimlerinin bir dizi halinde bekleyen mesajı ve doğrusal dönüşüm yoluyla harita içine ondalık sayılara bölünür.
- Belirli bir kaotik modeli için, başlangıç, yineleme ve algoritma gizli anahtarı gibi parametre ve başlangıç değeri belirlenir.
- Birkaç özel yineleme değeri seçilir, bu değerler ikili formata dönüştürülür ve özetleme değerini elde etmek için yan yana koyulur.

Kaotik çadır haritalama algoritması aşağıdaki denklem ile formüle edilir.

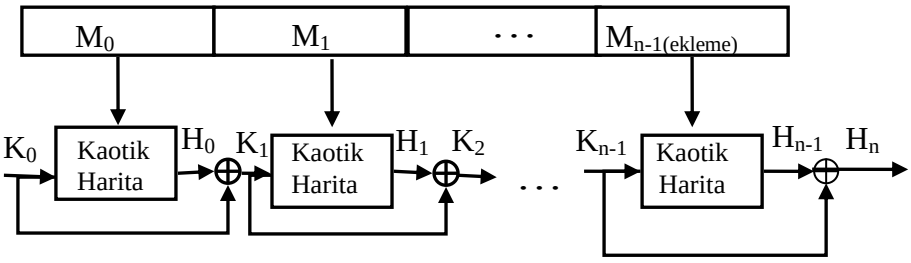
$$T(x)=\begin{cases} rx, & 0 \le x < 0.5, \\ r(1-x), & 0.5 \le x \le 1, \end{cases} \qquad r \in \left[\sqrt{2},2\right] \tag{5.1}$$

$r \in [0,1]$ aralığında iken, herhangi bir kaotik davranış olmadan birkaç yinelemeden sonra hesaplama sonuçları aynı değere gelir.
$r \in [1, \sqrt{2}]$ aralığında iken hesaplama periyodik olarak görünür.
$r \in [\sqrt{2}, 2]$ aralığında iken gösterilen periyodun kaybolmasıyla kaotik bir sistem oluşur.

Kaotik harita, blok uzunluğu 1024 bit olan $M_{n-1}$ blok uzunluğunu şifreler ve sabit uzunlukta 128 bit çıktı üretir [21].

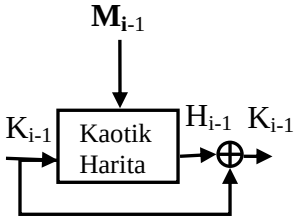
Kaotik haritalar 1024 bitlik mesaj blokları üzerinde işlem yaparlar. Eğer 1024 bitten az blok oluşursa, ilk olarak orijinal ileti M 'nin uzunluğu 1024 olacak şekilde, verinin sonuna bir adet "1" ve geri kalanlar için ise "0" eklenir. Daha sonra eklenmiş mesaj n tane 1024 bitlik bloklara ayrılır (Şekil 5.1). Sonuç olarak, $H_n$ şöyle hesaplanır:

$$H_n = K_{n-1} \oplus H_{n-1} = (K_{n-2} \oplus H_{n-2}) \oplus H_{n-1}$$
$$= ... = (K_0 \oplus H_1) \oplus H_2 \oplus ... \oplus H_{n-1} \quad (5.2)$$




**Şekil 5.1 Genel kaotik harita yapısı**

Algoritmanın özü Şekil 5.1 'de gösterilen kaotik özetleme fonksiyonu modülüdür. Şekil 5.2 'deki gibi $K_{i-1}$ in kontrolü altında, $H_{i-1}$ 'e uyan her $M_{i-1}$, i = 1, 2, . . . ,n için bu modül tarafından kodlanır ve sonra $K_i = K_{i-1} \oplus H_{i-1}$ hesaplanarak $K_i$ elde edilir. Kaotik çadır harita algoritmasında, $r_0$, $K_0$, $x_0$ başlangıç değerleri ve $H_n = K_n = K_{n-1} \oplus H_{n-1}$ ise son özetleme değeridir.




**Şekil 5.2. Kaotik harita yapısı**

Geliştirilen yazılımda kullanılan kaotik çadır harita basamakları şu şekildedir: İstemci tarafının başlama sayacı ve şifresi belirlenir daha sonra sayaç ve anahtar değeri HMAC fonksiyonu ile MAC 'lenir. Elde edilen sonuç Kaotik özetleme fonksiyonuna gönderilir.

Kaotik harita metoduna gelen mesaj bloğu 1024 bitten az ise ilk bitine "1", geri kalan bitlere "0" eklenir. Gelen mesaj bloğu tek blok olduğundan, 1024 bitlik mesaj bloğu 32 bitlik 32 adet bloğa ayrılır.

```
static void MesajFormat(string mesaj)
{
   string bits = ToBits(mesaj);

   if (bits.Length < 1024)
   {
      bits += "1";
      while (bits.Length < 1024)
         bits += "0";
   }
   for (int i = 0; i < 32; i++)
   {
      M[i] = Convert.ToUInt32(bits.Substring(i * 32, 32), 2);
   }
}
```

Kaotik çadır haritalamada iki adet parametre kullanılır($T(K_i,M_i)$).Bu parametrelerden $K_i$ girişi "x", $M_i$ girişi r ile temsil edilmiştir [21]. $K_i \oplus H_i$ hesaplamasında $K_i$ değerleri çadır haritadaki (1) denkleme göre yapılır.

```
static float TentMap(float x, float r)
{
   if (x >= 0)
   {
      if (x < 0.5)
         return r * x;
      else
         if (x <= 1)
            return r * (1 - x);
   }
   return 0f;
}
```

```
static float K(int n)
{
   if (n == 0)
      return TentMap(0.3f, 1.8f);
   else
      return TentMap(K(n - 1), Normalize((double)M[n - 1]));

}

static float Normalize(double girdi)
{
   double sonuç = Math.Round(girdi % 0.584d, 15);
   sonuç += 1.414d;
   return (float)sonuç;
}
```

Çadıra harita ile elde edilen çıkış, $K_i$ değeri ile XOR 'lanır. Bunun sonucu olarak üretilen $K_i$ bir sonraki $K_{i-1} \oplus H_{i-1}$ işlemine koyulur. Son işlemde edilen 128 bitlik verinin 8 biti kesilip alınır.

# 6. GELİŞTİRİLEN YAZILIM

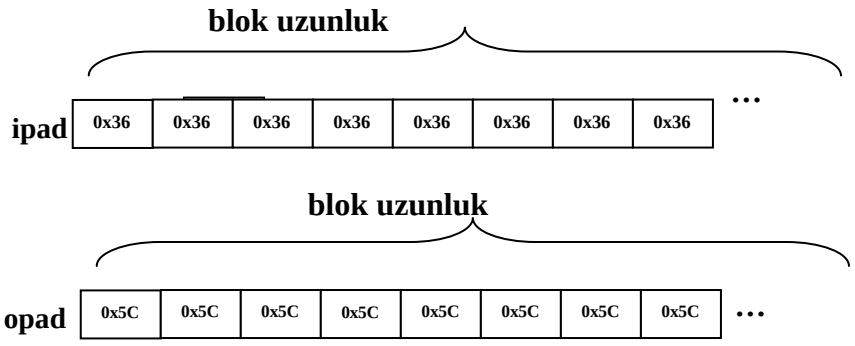
## 6.1. Bir HOTP Değeri Elde Etmek

Tek kullanımlık şifre tekniği, kullanıcının kullandığı şifre doğrulama onayından geçtikten sonra bir daha kullanılamaz ve kullanıcının her giriş isteğinde, yeni ve eskisinden farklı bir şifre ile kimlik doğrulanma istenmesi mantığına dayanır.

Tek kullanımlık parola sistemiyle etkileşimde bulunan iki taraf yer alır [22]. İstemci tarafı sahip olduğu sayaç ile kendine özgü anahtar bilgisini HOTP işlemine göndermektedir. HOTP işleminde çağrılan HMAC 'in temel işlevi:

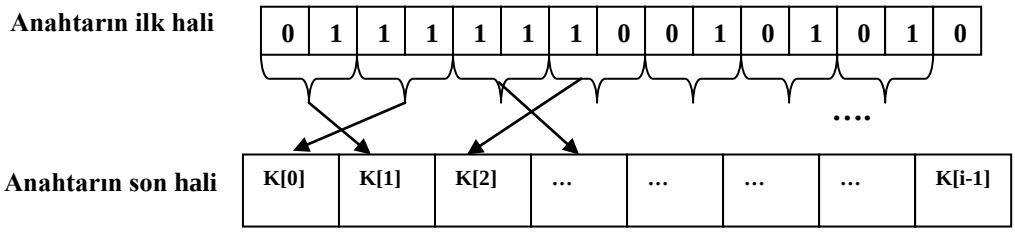
**H(($K_0 \oplus$ opad) || H(($K_0 \oplus$ ipad) || text)** 'dir

HMAC işleminde blok uzunluğu kadar oluşturulan ipad ve opad değerlerinin ataması yapılır.



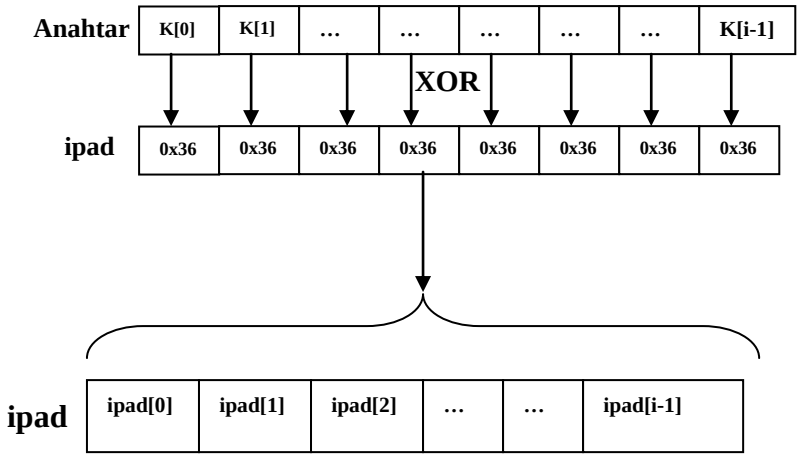

**Şekil 6.1 ipad ve opad işlemleri**

Bu işlemden sonra anahtar bitleri üzerinde yer değiştirme işlemi uygulanır.

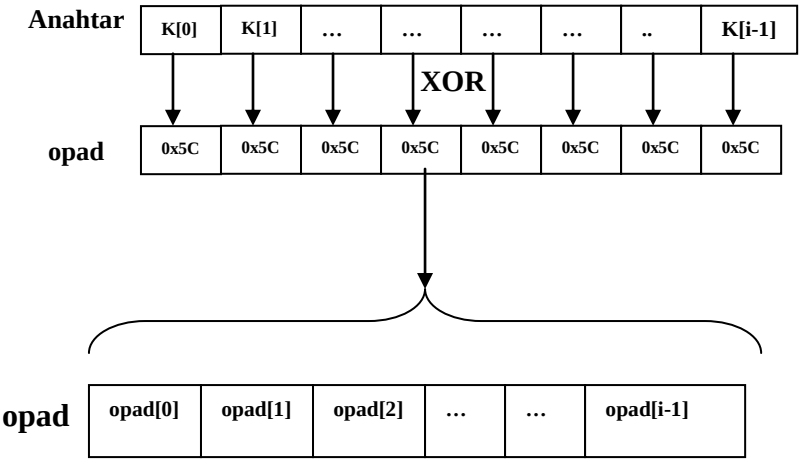



**Şekil 6.2 Anahtarı karıştırma**

Oluşturulan anahtar, ipad ve opad ile XOR ’lanır.

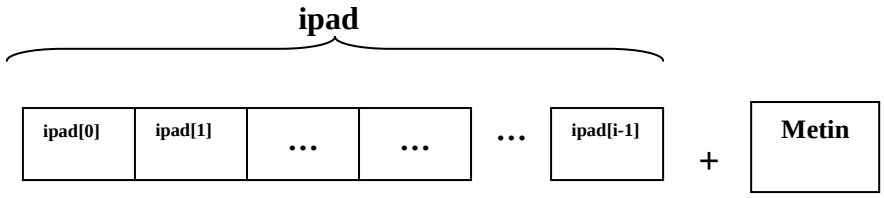



**Şekil 6.3 Anahtar ile ipad XOR işlemi**




**Şekil 6.4 Anahtar ile opad XOR işlemi**

ipad anahtarla XOR işleminden geçirildikten sonra kullanıcının belirlediği mesaj ipad ’e eklenir.

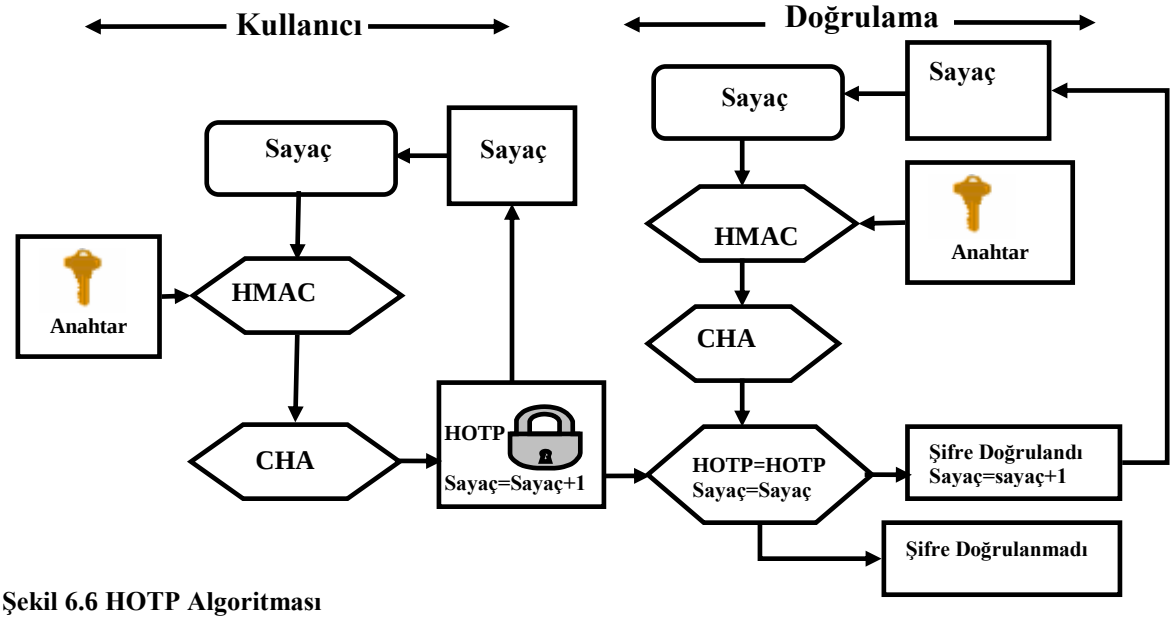



**Şekil 6.5 İpad ile metin birleştirilmesi**

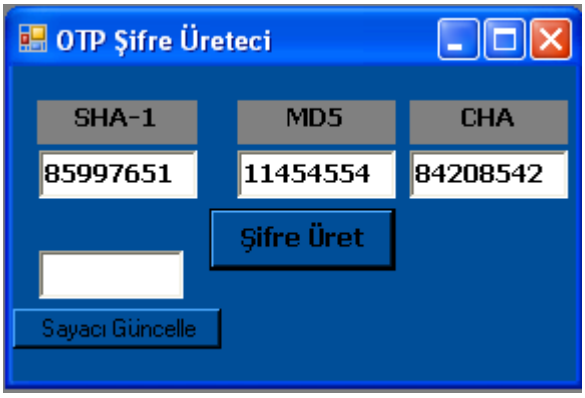
HMAC tabanlı HOTP algoritmasında, HMAC hesaplamasındaki mesajı temsil eden monotonik artan sayaç değeri ile istemci/sunucu tarafından simetrik gizli bir anahtar paylaşılarak başlatılır. HOTP değerini oluşturmak için, HMAC-SHA–1, HMACMD5 veya HMAC-CHA algoritması kullanılır.

Ortak gizli anahtar ve sayaç HMAC işlevinde ipad ve opad işleminden geçirildikten sonra oluşturulan mesaj, özetleme fonksiyonuna gönderilmektedir. Özetleme fonksiyonu sonucu oluşan değer kesilip HOTP değeri elde edilmektedir [23].

Sunucu ve istemci tarafının sayaç değerleri senkronize olarak başlatılmaktadır. İstemci tarafı üretilen HOTP değeri ile doğrulama yapmak istediğinde, sunucu, istemcinin ilerleme basamaklarına göre, HOTP işlemi ile istemci bilgilerini HMAC işlevi ile ipad ve opad işleminden geçirdikten sonra Kaotik, SHA–1 veya MD5 özetleme fonksiyonuna tabi tutar. Şekil 6.6 'da olduğu gibi Kaotik, SHA–1 veya MD5 özetleme fonksiyonu ile çıkan değer HOTP değerine dönüştürülmektedir. Sunucu, istemci tarafından elde edilen HOTP ve sayaç değerlerini karşılaştırılır. Eğer iki taraftan da elde edilen HOTP ve sayaç değerleri eşit ise doğrulama sağlanmış olur.

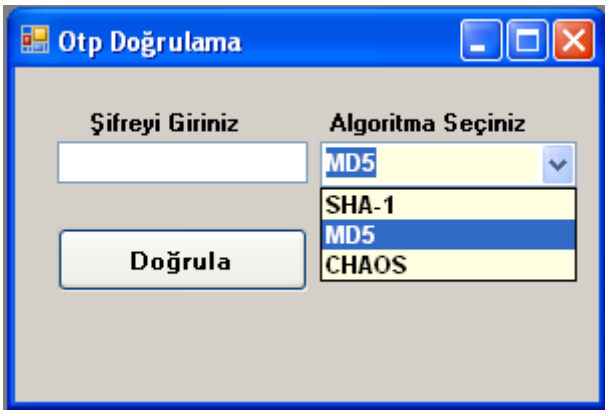



**Şekil 6.6 HOTP Algoritması**



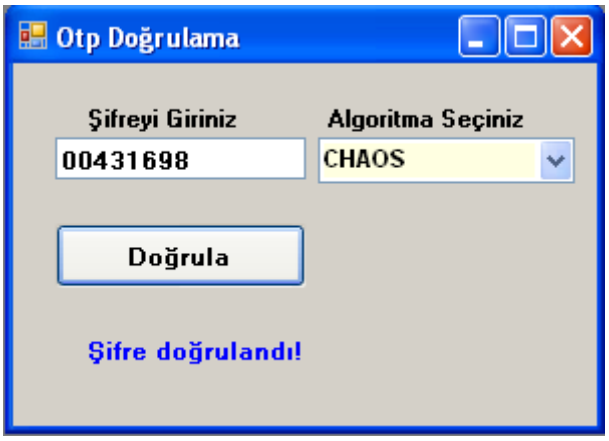

**Şekil 6.7 HOTP Yazılımı Üreteç**

Şekil 6.7 'de görüldüğü üzere şifre üret düğmesine her tıklandığında program yukarıdaki algoritma basamaklarını kullanarak kullanıcıya tek kullanımlık şifre değerini üretmektedir.

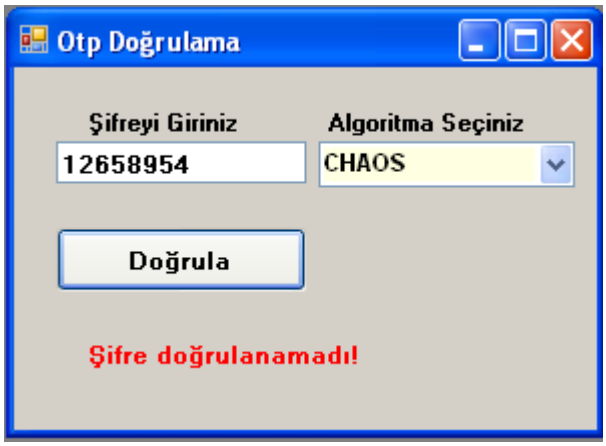



**Şekil 6.8 HOTP Yazılımı Doğrulama-I**

Kullanıcı erişmek istediği verilere veya sistemlere, bu elde ettiği tek kullanımlık şifreyi doğrulatmak durumundadır. Bunun için Şekil 6.8 'de görüldüğü üzere doğrulama tarafı kullanıcıdan aldığı tek kullanımlık şifreyi yukarıdaki algoritma basamaklarını kullanarak ürettiği değerleri karşılaştırmaktadır.




**Şekil 6.9 HOTP Yazılımı Doğrulama-II**




**Şekil 6.10 HOTP Yazılımı Doğrulama-III**

Eğer kullanıcı elde ettiği tek kullanımlık şifreleri üretmiş ve kullanmamış ise veya yanlış giriş yapmış ise ve hakkı olan giriş sayısı geçmiş ise o zaman senkronizasyon işlemi başlayacaktır.

# 7. ÜRETİLEN HOTP DEĞERLERİNİN GÜVENİRLİĞİ

## 7.1. Karışıklık ve Difüzyon Analizi

Karışıklık ve difüzyon, kriptografik algoritmalar için iki temel tasarım kriteridir. İkili format içindeki mesaj özeti için her bir bit sadece "1" veya "0" dır. Böylece en doğru difüzyon etkisi, başlangıç koşullarındaki doğru değişimler olmalıdır. Bu değişimler her bir bitin %50 değişim olasılığını sağlar.

Karışıklık ve difüzyon testi şu şekilde yapılmıştır: Ortak gizli anahtar ve sayaç ile özetleme değeri oluşturulmuştur. Daha sonra Ortak gizli anahtardaki bir bit rastgele seçilmiş ve bu bit değiştirilerek yeni bir özet değeri oluşturulmuştur. Bu iki özet değeri birbiriyle karşılaştırılmış ve değiştirilen bit sayısı $X_i$ 'olarak hesaplanmıştır.

**Tablo 7.1. Kaotik harita ile elde edilen özetleme değerleri**

| Sayaç | Hexadecimal Değeri |
|---|---|
| 0 | 6B9516C1E31D9E49946AE93E1CE261B6 |
| 1 | 4560F7E1CDE87F69BA9F081E32178096 |
| 2 | 4B8FCFF1C3074779B470300E3CF8B886 |
| 3 | 665BD1E1EED3596999A42E1E112CA696 |
| 4 | 7D1F9561F5971DE982E06A9EA68E2516 |
| 5 | 4834A0F1C0BC2879B7CB5F0E3F43D786 |

Genellikle dört istatistik şu şekilde ifade edilmektedir:

Bit sayısının ortalama değişimi : $\overline{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$, (7.1)

Ortalama olasılık değişikliği : $P = (\overline{X}/128)x100\%$ (7.2)

Değişmiş bit sayısının standart varyansı : $\Delta X = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(X_i - \overline{X})^2}$ (7.3)

Standart varyans: $\Delta P = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(X_i/128 \ - P)^2}\, x100\%$ (7.4)

Bu tür bir test N sayısı kadar uygulanır (N = 128, 256, 512, ve 1024 gibi). Burada, N toplam istatistik sayısı, i.simülasyon yapıldığında $X_i$ kadar bit sayısı değişir, $X_i$ ve P simgeleri değiştirilen bit sayısı ve değişen olasılık demektir. Sırasıyla $\Delta X$ ve $\Delta P$ difüzyon

ve karışıklığın kararlılığını gösterir. N = 128, 256, 512, ve 1024 ile testleri sayesinde, sırasıyla, Tablo 7.2 'de ilgili veriler listelenmiştir.

Tablo 7.2 'deki verilerin analizine dayanan şu sonuçlar çıkarılabilir: Ortalama değiştirilen bit sayısı ve değiştirilen yüzdelik, % 18,913 olarak değişmiştir. Bu sonuç yaklaşık olarak 24 bitin değişimini gösterir. İstatistiksel etki, saldırganın kesinlikle bilinmeyen birkaç düz-şifreli metinleri diğer düz metin ve metinlerin taklidini yapamayacağını veya azaltamadığını gösterir.

**Tablo 7.2. İstatistiksel performans**

| | **N=128** | **N=256** | **N=512** | **N=1024** | **Ortalama** |
|---|---|---|---|---|---|
| $X_i$ | 24.109 | 24.468 | 24.062 | 24.197 | 24.209 |
| P(%) | 18.835 | 19.116 | 18.798 | 18.904 | 18.913 |
| $\Delta X$ | 6.565 | 6.809 | 6.488 | 6.611 | 6.618 |
| $\Delta P$ | 18.717 | 18.680 | 18.665 | 18.655 | 18.681 |

## 7.2. Tahmin Saldırısı

Bazı zayıf şifreler tahmin saldırılarına karşı açık olur. Saldırgan doğrulama mesajını yerel(local) olarak saklar ve tahmini şifreler kullanarak, tahminin doğruluğu teyit etmeye çalışır. Burada, kaotik özetleme fonksiyonu çıkış uzunluğu 128 bittir (blok uzunluğu 16 byte 'dır). Kaotik özetleme fonksiyonunda saldırgan hesaplanan (gizli anahtar ile birlikte) doğru ileti kimlik doğrulama etiketlerini elde etmek zorundadır. Bu da bilinen düz metinlerden yaklaşık $2^{64}$ 'tür. Bunun anlamı kaotik özetleme fonksiyonu altındaki en az $2^{64}$ bloğun işlenmesini gerektirir ki bu gerçek bir senaryoda imkansız bir iştir. Ancak bu saldırı kaotik özetleme fonksiyonunun çarpışma davranışında ciddi kusurlar oluşursa gerçekleşebilir. Burada elde edilen HMAC çıkışı kesilmekte ve bitlerin bir kısmı çıkarılmaktadır. Bu kesilen çıkışın güvenlik avantajı, saldırganın özetleme çıkışı hakkında daha az bilgiye sahip olmasını sağlamaktır. Üretilen Kaotik HOTP değerlerinin 8 haneli olması ve istemcinin belli sayıda HOTP değerlerini doğrulamadığında sistemin senkronizasyon işlemini başlatması tahmin saldırılarını önlemekte yeterli olacaktır.

## 7.3. Dinleme Saldırısı

Burada saldırgan kendisine ait olmayan bilgileri dinler ve ele geçirmeye çalışır. Kullanıcı kimlik doğrulama için bilgi girişi yaptığı zaman, kullanıcı şifresini yakalamaya çalışır. Burada şifreleme simetrik olduğundan başarıya ulaşamayacaktır.

### 7.4. Tekrarlama Saldırıları

Saldırgan, daha önce elde ettiği bilgileri kullanarak yasal kullanıcı gibi sisteme giriş yapmaya çalışır. Ancak burada HMAC 'a dayalı HOTP şifreleri kullanıldığından, özetleme fonksiyonunun zincirleme tekniğinden dolayı başarısız olacaktır.

### 7.5. Sahte Kullanıcı Saldırısı

Saldırgan daha önce engellediği iletişimleri yeniden şekillendirerek yasal kullanıcıymış gibi sisteme erişmeye çalışır. Bunu gerçekleştirebilmesi için mesajın sahtesini oluşturması gerekir. Ancak saldırgan kullanılan anahtar bilgisine sahip olmadığın başarıya ulaşamayacaktır.

### 7.6. Gizli Sunucu Saldırıları

Eğer saldırgan gizli sunucu kullanarak yasal kullanıcıyı kandırmaya çalışırsa, sahte bir geçerli doğrulma mesajı üretmesi gerekir. Ancak saldırgan anahtar bilgisine sahip olmadığından, böyle bir mesajı oluşturamayacaktır.

### 7.7. Eş zamanlı Oturum Saldırısı

Saldırgan kullanıcı ve sunucu arasındaki iletişimle ilgili geçerli giriş mesajını oluşturarak yasal kullanıcı gibi hareket eder. Ancak saldırgan anahtar bilgisine sahip olmadığından böyle bir mesaj üretemeyecektir.

### 7.8. Çalınan Doğrulama Saldırısı

Uygulamaların çoğunda sunucu metin şifreleri yerine, özetlenmiş şifreleri saklar. Saldırgan sunucudan çaldığı doğrulama şifrelerini kullanarak yasal kullanıcı gibi davranarak bunu kullanır. Ancak burada sunucu tarafında herhangi bir doğrulama tablosu saklanmadığından, bu tür bir saldırı başarısız olacaktır.

Ayrıca HOTP sisteminin güvenliği Kaotik özetleme fonksiyonunun kriptografik özelliklerine bağlıdır. Saldırgan, burada kullanılan Kaotik haritanın davranışlarını bilmek zorundadır. Ayrıca elde ettiği değer doğru olsa bile, sunucu üzerinde doğrulama yapmaya çalıştığında, sunucu sayacına sahip olması gerekmektedir.

## 8. SONUÇLAR

Ağlar üzerinde tehdit ve saldırıların artması nedeniyle bilgi güvenliğini sağlamak günümüz teknolojisinin vazgeçilmezlerinden olmuştur. Bilgi güvenliğinin temini için en güvenli yolu kullanmak gerekmektedir. Sabit şifrelerle bir sistemin güvenliğini sağlamak mümkün değildir. Bu nedenler kullanımı kolay ve dinamik bir şifre sistemi geliştirme ihtiyacı doğurmuştur.

Bu tez çalışmasında HMAC tabanlı üretilen tek kullanımlık şifre yazılımı geliştirilmiştir. Özetleme fonksiyonu olarak doğrusal olmayan, başlangıç şartlarına bağımlı, hareketleri kestirilemeyen ve ergodiklik gibi özelliklere sahip olan kaotik harita kullanılmıştır. Bununla birlikte MD5 ve SHA–1 özetleme fonksiyonları da kullanılarak HOTP 'ler elde edilebilecek şekilde bir yazılım geliştirilmiştir. Geliştirilen yazılımın üç özetleme fonksiyonu kullanarak HOTP değerleri üretmesi ve tek bir sunucu üzerinde doğrulamayı gerçekleştirebilmesi, bu yazılıma güç kazandırmıştır.

MD5 ve SHA–1 özetleme fonksiyonlarının güvenirliklerinin kalmadığı bilinmektedir. Oysaki, HMAC tabanlı kullanılan özetleme fonksiyonları için herhangi bir saldırı günümüze kadar tespit edilememiştir [23]. HMAC tabanlı HOTP algoritmasına yapılacak olan bir saldırıda, saldırgan, istemci ve sunucu tarafından paylaşılan gizli ortak anahtarı, kullanıcının sayacını, kullanılan özetleme fonksiyonunu ayrıca kaotik özetleme fonksiyonu kullanıyor ise kaotik haritayı bilmesi gerekecektir. Bu da neredeyse imkansızdır.

Geliştirilen yazılımla üretilen şifrelerin rakamlardan oluşması, kullanıcı için esneklik, sekiz haneli olması da, sistemin olabilecek bir saldırıya karşı daha güvenli olmasını sağlayacaktır.

**KAYNAKLAR**


1. **Tsai C.S, Lee C.C, Hwang M.S**, September 2005," Password Authentication Schemes: Current Status and Key issues", international Journal of Network security,Vol.3 No.2,PP.101-115.
2. **Liao I.E., Lee C.C., and Hwang M.S**., 2005, "A Password Authentication Scheme Over Insecure Networks," Accepted in Journal of Computer and System Sciences.
3. **Archer H.J.**, 2002, "OPA: A One-time Password System", Parallel Processing Workshops, Proceedings. International Conference , Page(s): 25 – 29.
4. **Vaidya B, Park H.J., Yeo S.S., Rodrigues J.J.P.C.,** 15 March 2011," Robust one-time password authentication scheme using smart card for home network environment", Computer Communications, Volume 34, Issue 3, Pages 326-336,.
5. http://tr.wikipedia.org/wiki/ARM_mimarisi, 17 Ocak 2011.
6. "www.kamusm.gov.tr" web sayfası, Ocak 2011.
7. **Stallings W.** , 2005, "Cryptography and Network Security Principles and Practices" Fourth Edition, Prentice Hall.
8. O.D.T.Ü., Şubat 2004, uygulamalı matematik enstitüsü, kriptografi bölümü, kriptolojiye giriş ders notları,
9. http://www.cryptographyworld.com/rsa.htm, Şubat 2011
10. http://world.std.com/~franl/crypto/rsa-guts.html, Şubat 2011
11. **Vipul G., Ajith A., Sugata S., Sang Y.H**., 2005, "The N/R One Time Password System",Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference Volume:1 , Page(s): 733 - 738 Vol.1.
12. **Benthal L. and U**., 2007, "Manageable One-Time Password for Consumer Applications", Consumer Electronics, 2007. ICCE 2007. Digest of Technical Papers. International Conference, Page(s): 1 - 2
13. **Alghathbar M.K. , Mahmoud H.A.,** 2009 ," Noisy Password Scheme: A New One Time Password System" Electrical and Computer Engineering, CCECE '09. Canadian Conference, Page(s): 841 - 846
14. **Eastlake D., Jones P.**, September 2001, "US Secure Hash Algorithm–1" (SHA1),
15. http://www.schneier.com/blog/archives/2005/02/sha1_broken.html, 10 Ocak 2011

16. Federal Information Processing Standards Publication, 2002, “The Keyed-Hash Message Authentication Code (HMAC)”, FIPS PUB 198.
17. **Krawczyk H., Bellare M., Canetti R.,**1997, “HMAC: Keyed-Hashing for Message Authentication”, Request for Comments: 2104.
18. **Haijun R., Yong W., Qing X., Huaqian Y**., 2009, “A novel method for one-way hash function construction based on spatiotemporal chaos”, Chaos, Solitons and Fractals 42,2014–2022.
19. **Amin M., Faragallah O.S. , Abd El-Latif A.A**., 2009, “Chaos-based hash unction (CBHF) for cryptographic applications”, ScienceDirect, Chaos, Solitons and Fractals 42, 767–772
20. **Xiao D., Peng W., Liao X., Xiang T**., 2010, “Collision analysis of one kind of chaos-based hash function”, ScienceDirect Physics Letters A 374,1228–1231
21. www.esiea-recherche.eu/.../slides_iAWACS09_Danchev-Maqableh_CBHF.ppt, Ocak 2011.
22. **Imamoto K., Sakurai K.,** 2004, "A Design of Diffie-Hellman Based Key Exchange Using One-time ID in Pre-shared Key Model", Proceedings of the18th International Conference on Advanced Information Networking and Application.
23. **M. Raihi D., . Hoornaert F, Ranen O.,** 2005, “HOTP: An HMAC-Based One-Time Password Algorithm”, Request for Comments: 4226.

**EK–1: HOTP ÜRETEÇ YAZILIMI**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace OTP_Smart
{  public partial class Form1 : Form
   {   static int ce_sayaç;
       public static int sayaç = 0;
       public Form1()
       {
          InitializeComponent();        }

       private void button1_Click(object sender, EventArgs e)
       {   textBox2.Text = OTP_Dogrulama.HOTP.Şifrele();
          textBox3.Text = OTP_Dogrulama.HOTPMD5.Şifrele();
          textBox4.Text = OTP_Dogrulama.HOTPCHA.Şifrele();  }

       private void button2_Click(object sender, EventArgs e)
       {   sayaç = int.Parse(textBox1.Text);
          OTP_Smart.Sayac.Yaz(sayaç);
          OTP_Smart.Sayac_Cha.Yaz(sayaç);
          OTP_Smart.Sayac_MD5.Yaz(sayaç);
       }

       private void Form1_Load(object sender, EventArgs e)
       { ce_sayaç = OTP_Smart.Sayac.Oku();        }
using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{    class HOTP
    {   static int parola_uzunluğu = 8;
     public static int Sayaç = 0;
       public static String Şifre = "19850476";
       public static string Şifrele()
       {Sayaç = OTP_Smart.Sayac.Oku();
       String mesaj = Sayaç.ToString();
       String key = Şifre;
       String hmac_sonuç = HMAC.Şifrele(key, mesaj);
       byte[] dizi = new byte[4];
       byte[] tampon = Converter.ToByteArray(hmac_sonuç);
       dizi[0] = Converter.toWord(tampon[0], tampon[1]);
       dizi[1] = Converter.toWord(tampon[2], tampon[3]);
```

```
        dizi[2] = Converter.toWord(tampon[4], tampon[5]);
        dizi[3] = Converter.toWord(tampon[6], tampon[7]);
        dizi[0] = (byte) (dizi[0] & (byte) 0x7f);
        dizi[1] = (byte) (dizi[1] & (byte) 0xff);
        dizi[2] = (byte) (dizi[2] & (byte) 0xff);
        dizi[3] = (byte) (dizi[3] & (byte) 0xff);
        Sayaç += 1;
        OTP_Smart.Sayac.Yaz(Sayaç);
        return Converter.toInt(dizi[0], dizi[1], dizi[2], dizi[3]).ToString().Substring(1,
parola_uzunluğu).ToString();}  }}
using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{    class HOTPCHA
    {   static int parola_uzunluğu = 8;
        public static int Sayaççha = 0;
        public static String Şifre = "19850476";
        public static string Şifrele() {
        Sayaççha = OTP_Smart.Sayac_Cha.Oku();
        String mesaj = Sayaççha.ToString();
        String key = Şifre;
        String hmac_sonuç = HMACCHA.Şifrele(key, mesaj);
        byte[] dizi = new byte[4];
        byte[] tampon = ConverterCHA.ToByteArray(hmac_sonuç);
        dizi[0] = ConverterCHA.toWord(tampon[0], tampon[1]);
        dizi[1] = ConverterCHA.toWord(tampon[2], tampon[3]);
        dizi[2] = ConverterCHA.toWord(tampon[4], tampon[5]);
        dizi[3] = ConverterCHA.toWord(tampon[6], tampon[7]);
        dizi[0] = (byte) (dizi[0] & (byte) 0x7f);
        dizi[1] = (byte) (dizi[1] & (byte) 0xff);
        dizi[2] = (byte) (dizi[2] & (byte) 0xff);
        dizi[3] = (byte) (dizi[3] & (byte) 0xff);
        Sayaççha += 1;
        OTP_Smart.Sayac_Cha.Yaz(Sayaççha);
             return ConverterCHA.toInt(dizi[0], dizi[1], dizi[2],
dizi[3]).ToString().Substring(1, parola_uzunluğu).ToString();
        }       }    }

using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{  class HOTPMD5
    { static int parola_uzunluğu = 8;
        public static int Sayaçmd5 = 0;
        public static String Şifre = "19850476";
        public static string Şifrele()
        {Sayaçmd5 = OTP_Smart.Sayac_MD5.Oku();
```

```
        String mesaj = Sayaçmd5.ToString();
        String key = Şifre;
        String hmac_sonuç = HMACMD5.Şifrele(key, mesaj);
        byte[] dizi = new byte[4];
        byte[] tampon = ConverterMD5.ToByteArray(hmac_sonuç);
        dizi[0] = ConverterMD5.toWord(tampon[0], tampon[1]);
        dizi[1] = ConverterMD5.toWord(tampon[2], tampon[3]);
        dizi[2] = ConverterMD5.toWord(tampon[4], tampon[5]);
        dizi[3] = ConverterMD5.toWord(tampon[6], tampon[7]);
        dizi[0] = (byte) (dizi[0] & (byte) 0x7f);
        dizi[1] = (byte) (dizi[1] & (byte) 0xff);
        dizi[2] = (byte) (dizi[2] & (byte) 0xff);
        dizi[3] = (byte) (dizi[3] & (byte) 0xff);
          Sayaçmd5 = Sayaçmd5 + 1;
           OTP_Smart.Sayac_MD5.Yaz(Sayaçmd5);
           return ConverterMD5.toInt(dizi[0], dizi[1], dizi[2], dizi[3]).ToString().Substring(1,
parola_uzunluğu).ToString(); } }}

using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
   { class CHA_Algoritma
   {static uint[] M = new uint[32];
    static float TentMap(float x, float r)
      { if (x >= 0)
         { if (x < 0.5)
               return r * x;
            else
               if (x <= 1)
                  return r * (1 - x);           }
         return 0f;       }

      static public string Şifrele(string mesaj)
      {  MesajFormat(mesaj);
         uint[] tampon = H(31);
         string sonuç = "";
         for (int i = 0; i < 4; i++)
         {   string t = Convert.ToString(tampon[i], 16);
             sonuç += t;
         } return sonuç;       }

      static void MesajFormat(string mesaj)
      {  string bits = ToBits(mesaj);
        if (bits.Length < 1024)
         {   bits += "1";
            while (bits.Length < 1024)
               bits += "0";    }
      for (int i = 0; i < 32; i++)
```

```
        { M[i] = Convert.ToUInt32(bits.Substring(i * 32, 32), 2);} }

      static uint[] H(int n)
      { uint[] tampon = new uint[4];
        if (n > 1)
        {  tampon = H(n - 1);
           float k = K(n - 1);
           for (int i = 0; i < 4; i++)
            tampon[i] = tampon[i] ^ (uint)(k * 1024 * 1024 * 1024);
           return tampon;          }
        else
        {  tampon[0] = Convert.ToUInt32("67452301", 16);
           tampon[1] = Convert.ToUInt32("efcdab89", 16);
           tampon[2] = Convert.ToUInt32("98badcfe", 16);
           tampon[3] = Convert.ToUInt32("10325476", 16);
           float k = K(0);
           for (int i = 0; i < 4; i++)
              tampon[i] = tampon[i] ^ (uint)(k*1024*1024*1024);
           return tampon;          }       }


      static float K(int n)
      {    if (n == 0)
             return TentMap(0.3f, Normalize((double)M[n]));
           else
             return TentMap(K(n - 1), Normalize((double)M[n - 1]));       }
      static float Normalize(double girdi)
      {   double sonuç = Math.Round(girdi % 0.585d, 15);
          sonuç += 1.414d;
          return (float)sonuç;       }
 static string ToBits(string str)
      {   byte[] bytes = Converter.ToByteArray(str);
          String binary = "";
          for (int j = 0; j < bytes.Length; j++)
          {   byte b = bytes[j];
             int val = b;
             for (int i = 0; i < 8; i++)
             {  binary += (val & 128) == 0 ? 0 : 1;
                val <<= 1;              }          }
          return binary;       }    }}
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;
namespace OTP_Dogrulama
{
   class MD5_Algoritma
   { public static string Şifrele(string katar)
      {byte[] data = ConverterMD5.ToByteArray(katar);
```

```
            MD5 md = new MD5CryptoServiceProvider();
            // This is one implementation of the abstract class SHA1.
            byte[] hash = md.ComputeHash(data);
            return ConvertToHex(System.Text.Encoding.UTF7.GetString(hash, 0,
hash.Length));}
     public static string ConvertToHex(string asciiString)
        { string hex = "";
            foreach (char c in asciiString)
            {int tmp = c;
               hex += String.Format("{0:x2}",
(uint)System.Convert.ToUInt32(tmp.ToString()));
            }           return hex.ToUpper();       }    }}
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;


namespace OTP_Dogrulama
{ class SHA1_Algoritma
   { public static string Şifrele(string katar)
      {  byte[] data = Converter.ToByteArray(katar);
         SHA1 sha = new SHA1CryptoServiceProvider();
         // This is one implementation of the abstract class SHA1.
         byte[] hash = sha.ComputeHash(data);
         return ConvertToHex(System.Text.Encoding.UTF7.GetString(hash, 0,
hash.Length)); }
   public static string ConvertToHex(string asciiString)
      {  string hex = "";
         foreach (char c in asciiString)
         {  int tmp = c;
            hex += String.Format("{0:x2}",
(uint)System.Convert.ToUInt32(tmp.ToString())); }
         return hex.ToUpper();       }    }}

using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{  class ConverterCHA
   { public static byte toWord(byte sayı1, byte sayı2)
      { return (byte)((sayı1 << 8) | (sayı2 & 0xff));       }
      public static int toInt(byte sayi1, byte sayi2, byte sayi3, byte sayi4)
      {  return (int)((sayi1 << 24) | (sayi2 << 16) | (sayi3 << 8) | (sayi4 & 0xff));  }
  public static byte[] ToByteArray(string str)
      {byte[] dizi = new byte[str.Length];
         int index = 0;
         foreach (char c in str)
         { dizi[index] = (byte)c;
```

```
            index++;            }
          return dizi;        }

       public static string toString(byte[] data)
       {string s = "";
          foreach (byte b in data)
          {s += (char)b;        }
          return s;        }    }}
using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{ class ConverterMD5
   { public static byte toWord(byte sayı1, byte sayı2)
      {  return (byte)((sayı1 << 8) | (sayı2 & 0xff));        }
   public static int toInt(byte sayi1, byte sayi2, byte sayi3, byte sayi4)
      {  return (int)((sayi1 << 24) | (sayi2 << 16) | (sayi3 << 8) | (sayi4 & 0xff)); }
      public static byte[] ToByteArray(string str)
      { byte[] dizi = new byte[str.Length];
         int index = 0;
         foreach (char c in str)
         { dizi[index] = (byte)c;
            index++;       }
         return dizi;       }
  public static string toString(byte[] data)
      {  string s = "";
         foreach (byte b in data)
         { s += (char)b;      }
         return s;        }    }}
using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{  class Converter
   {  public static byte toWord(byte sayı1, byte sayı2)
      {  return (byte)((sayı1 << 8) | (sayı2 & 0xff));  }
  public static int toInt(byte sayi1, byte sayi2, byte sayi3, byte sayi4)
      {return (int)((sayi1 << 24) | (sayi2 << 16) | (sayi3 << 8) | (sayi4 & 0xff)); }
   public static byte[] ToByteArray(string str)
      { byte[] dizi = new byte[str.Length];
         int index = 0;
         foreach (char c in str)
         {  dizi[index] = (byte)c;
            index++;        }
         return dizi;       }
    public static string toString(byte[] data)
      { string s = "";
         foreach (byte b in data)
         { s += (char)b;           }
```

```
          return s;       }    }}
using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{  class HMACCHA
   {     static int block_size = 16;
          static byte[] opad;
          static byte[] ipad;
          static byte[] key_dizi;
  public static String Şifrele(String key, String mesaj)
          {  key_dizi = new byte[16];
             opad = new byte[block_size];
             ipad = new byte[block_size];
  while (key.Length < 32)
                key = key + "0";
             for (int i = 0; i < block_size; i++)
             {  opad[i] = 0x5C;
                ipad[i] = 0x36;              }
                key_dizi[0] = byte.Parse(key.Substring(2, 2));
                key_dizi[1] = byte.Parse(key.Substring(0, 2));
                key_dizi[2] = byte.Parse(key.Substring(6, 2));
                key_dizi[3] = byte.Parse(key.Substring(4, 2));
                key_dizi[4] = byte.Parse(key.Substring(10, 2));
                key_dizi[5] = byte.Parse(key.Substring(8, 2));
                key_dizi[6] = byte.Parse(key.Substring(14, 2));
                key_dizi[7] = byte.Parse(key.Substring(12, 2));
                key_dizi[8] = byte.Parse(key.Substring(18, 2));
                key_dizi[9] = byte.Parse(key.Substring(16, 2));
                key_dizi[10]= byte.Parse(key.Substring(22, 2));
                key_dizi[11]= byte.Parse(key.Substring(20, 2));
                key_dizi[12]= byte.Parse(key.Substring(26, 2));
                key_dizi[13]= byte.Parse(key.Substring(24, 2));
                key_dizi[14]= byte.Parse(key.Substring(30, 2));
                key_dizi[15]= byte.Parse(key.Substring(28, 2));
   for (int i = 0; i < block_size; i++)
                {   ipad[i] = (byte)(key_dizi[i] ^ (byte)ipad[i]);
                   opad[i] = (byte)(key_dizi[i] ^ (byte)opad[i]);  }
 return CHA_Algoritma.Şifrele(ConverterCHA.toString(opad) +
CHA_Algoritma.Şifrele(mesaj + ConverterCHA.toString(ipad)));}}}

using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{   class HMACMD5
   {  static int block_size =16;
      static byte[] opad;
      static byte[] ipad;
```

```
        static byte[] key_dizi;
public static String Şifrele(String key, String mesaj)
        { key_dizi = new byte[16];
            opad = new byte[block_size];
            ipad = new byte[block_size];

            while (key.Length < 32)
                key = key + "0";
            for (int i = 0; i < block_size; i++)
            {  opad[i] = 0x5C;
                ipad[i] = 0x36;           }
              key_dizi[0] = byte.Parse(key.Substring(2, 2));
                key_dizi[1] = byte.Parse(key.Substring(0, 2));
                key_dizi[2] = byte.Parse(key.Substring(6, 2));
                key_dizi[3] = byte.Parse(key.Substring(4, 2));
                key_dizi[4] = byte.Parse(key.Substring(10, 2));
                key_dizi[5] = byte.Parse(key.Substring(8, 2));
                key_dizi[6] = byte.Parse(key.Substring(14, 2));
                key_dizi[7] = byte.Parse(key.Substring(12, 2));
                key_dizi[8] = byte.Parse(key.Substring(18, 2));
                key_dizi[9] = byte.Parse(key.Substring(16, 2));
                key_dizi[10] = byte.Parse(key.Substring(22, 2));
                key_dizi[11] = byte.Parse(key.Substring(20, 2));
                key_dizi[12] = byte.Parse(key.Substring(26, 2));
                key_dizi[13] = byte.Parse(key.Substring(24, 2));
                key_dizi[14] = byte.Parse(key.Substring(30, 2));
                key_dizi[15] = byte.Parse(key.Substring(28, 2));
                for (int i = 0; i < block_size; i++)
                {  ipad[i] = (byte)(key_dizi[i] ^ (byte)ipad[i]);
                    opad[i] = (byte)(key_dizi[i] ^ (byte)opad[i]); }
            return MD5_Algoritma.Şifrele(ConverterMD5.toString(opad) +
MD5_Algoritma.Şifrele(mesaj + ConverterMD5.toString(ipad))); } }}
using System;
using System.Collections.Generic;
using System.Text;
namespace OTP_Dogrulama
{   class HMAC
    {  static int block_size = 16;
        static byte[] opad;
        static byte[] ipad;
        static byte[] key_dizi;
public static String Şifrele(String key, String mesaj)
        {  key_dizi = new byte[16];
            opad = new byte[block_size];
            ipad = new byte[block_size];
while (key.Length < 32)
                key = key + "0";
            for (int i = 0; i < block_size; i++)
            {  opad[i] = 0x5C;
```

```
            ipad[i] = 0x36;           }
            key_dizi[0] = byte.Parse(key.Substring(2, 2));
            key_dizi[1] = byte.Parse(key.Substring(0, 2));
            key_dizi[2] = byte.Parse(key.Substring(6, 2));
            key_dizi[3] = byte.Parse(key.Substring(4, 2));
            key_dizi[4] = byte.Parse(key.Substring(10, 2));
            key_dizi[5] = byte.Parse(key.Substring(8, 2));
            key_dizi[6] = byte.Parse(key.Substring(14, 2));
            key_dizi[7] = byte.Parse(key.Substring(12, 2));
            key_dizi[8] = byte.Parse(key.Substring(18, 2));
            key_dizi[9] = byte.Parse(key.Substring(16, 2));
            key_dizi[10] = byte.Parse(key.Substring(22, 2));
            key_dizi[11] = byte.Parse(key.Substring(20, 2));
            key_dizi[12] = byte.Parse(key.Substring(26, 2));
            key_dizi[13] = byte.Parse(key.Substring(24, 2));
            key_dizi[14] = byte.Parse(key.Substring(30, 2));
            key_dizi[15] = byte.Parse(key.Substring(28, 2));
            for (int i = 0; i < block_size; i++)
            {  ipad[i] = (byte)(key_dizi[i] ^ (byte)ipad[i]);
               opad[i] = (byte)(key_dizi[i] ^ (byte)opad[i]); }
            return SHA1_Algoritma.Şifrele(Converter.toString(opad) +
SHA1_Algoritma.Şifrele(mesaj + Converter.toString(ipad))); } }}
using System;
using System.Collections.Generic;
using Microsoft.Win32;
using System.Text;
namespace OTP_Smart
{  class Sayac_Cha
   {  static RegistryKey reg;
      static int sonuç = 11111111;
      public static int Oku()
      {  reg = Registry.CurrentUser.OpenSubKey("otp_cha");
         if (reg == null)
            Yarat();
         sonuç = int.Parse(reg.GetValue("otpchaos_sayac").ToString());
         return sonuç;       }
    public static void Yaz(int sayı)       {
         reg = Registry.CurrentUser.OpenSubKey("otp_cha", true);
         if (reg == null)
            Yarat();
         reg.SetValue("otpchaos_sayac", sayı);       }
         static void Yarat()
      {   reg = Registry.CurrentUser.CreateSubKey("otp_cha");
         reg.SetValue("otpchaos_sayac", sonuç);       }    }}
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Win32;
namespace OTP_Smart
```

```
{   class Sayac_MD5
   { static RegistryKey reg;
      static int sonuç = 11111111;
public static int Oku()
      {  reg = Registry.CurrentUser.OpenSubKey("otp");
         if (reg == null)
            Yarat();
         sonuç = int.Parse(reg.GetValue("otpmd5_sayac").ToString());
         return sonuç;       }
         public static void Yaz(int sayı)
      { reg = Registry.CurrentUser.OpenSubKey("otp", true);
         if (reg == null)
            Yarat();
         reg.SetValue("otpmd5_sayac", sayı);       }
      static void Yarat()
      {  reg = Registry.CurrentUser.CreateSubKey("otp");
         reg.SetValue("otpmd5_sayac", sonuç);       }    }}
using System;
using Microsoft.Win32;
using System.Collections.Generic;
using System.Text;
namespace OTP_Smart
{   class Sayac
   {  static RegistryKey reg;
      static int sonuç = 11111111;
 public static int Oku()
      {  reg = Registry.CurrentUser.OpenSubKey("otp");
         if (reg == null)
            Yarat();
         sonuç = int.Parse(reg.GetValue("otp_sayac").ToString());
         return sonuç;       }
      public static void Yaz(int sayı)       {
         reg = Registry.CurrentUser.OpenSubKey("otp", true);
         if (reg == null)
            Yarat();
         reg.SetValue("otp_sayac", sayı);      }
      static void Yarat()
      {   reg = Registry.CurrentUser.CreateSubKey("otp");
         reg.SetValue("otp_sayac", sonuç);       }    }}
```

**EK–2: HOTP DOĞRULAMA YAZILIMI**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace otp_dogrulama
{   public partial class Form1 : Form
   { public static int giris_siniri = 0;
       static int seçim;
       static int  ce_sayaç;
      public static string key = "19850476";
        public Form1(){
         InitializeComponent();     }
      private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)       {
         if (comboBox1.SelectedIndex == 0)
         {              seçim = 0;          }
         else if (comboBox1.SelectedIndex == 1)
         {              seçim = 1;          }
         else
         {    seçim = 2;          }       }
      private void button1_Click(object sender, EventArgs e)
      {   Form2 senkronizsayon = new Form2();
         ce_sayaç = OTP_Smart.Sayac.Oku("otp_smart_sayac");
         textBox2.Text = ce_sayaç.ToString();
         bool doğruMu = false;
         int sayaç = int.Parse(textBox2.Text);
         for (int i = sayaç+75; i >= sayaç; i--)
         {if (HOTP.Şifrele(i, key, seçim) == textBox1.Text.Trim())
            {    doğruMu = true;
               OTP_Smart.Sayac.Yaz(HOTP.Sayaç, "otp_smart_sayac");
               textBox2.Text = HOTP.Sayaç.ToString(); break;} }
         if (doğruMu)          {
           label4.Text = "Şifre doğrulandı!";
           label4.ForeColor = Color.Blue;
           giris_siniri = 0;          }
         else { OTP_Smart.Sayac.Yaz(sayaç, "otp_smart_sayac");
            label4.Text = "Şifre doğrulanamadı!";
            label4.ForeColor = Color.Red;
               giris_siniri += 1;
               if (giris_siniri == 5)    {
                  senkronizsayon.Show();
                  this.Hide();
                  textBox1.Enabled = false;
                  button1.Enabled = false;} } }
      private void Form1_Load(object sender, EventArgs e)
      {  ce_sayaç = OTP_Smart.Sayac.Oku("otp_smart_sayac");
```

```
          textBox2.Text = ce_sayaç.ToString();
          comboBox1.SelectedIndex = 1;              }
       private void button2_Click_1(object sender, EventArgs e)
       {    int counter;
          counter = int.Parse(textBox2.Text);
          OTP_Smart.Sayac.Yaz(counter, "otp_smart_sayac");}}}
using System;
using System.Collections.Generic;
using System.Text;
namespace otp_dogrulama
{    class HMAC    {
       static int block_size = 16;
       static byte[] opad;
       static byte[] ipad;
       static byte[] key_dizi;
       public static String Şifrele(String key, String mesaj,int seç)
       {    key_dizi = new byte[16];
          opad = new byte[block_size];
          ipad = new byte[block_size];
         while (key.Length <32)
             key = key + "0";
          while (mesaj.Length < 32)
             mesaj = mesaj + "1";
          for (int i = 0; i < block_size; i++)
          {  opad[i] = 0x5C;
             ipad[i] = 0x36;  }
             key_dizi[0] = byte.Parse(key.Substring(2, 2));
             key_dizi[1] = byte.Parse(key.Substring(0, 2));
             key_dizi[2] = byte.Parse(key.Substring(6, 2));
             key_dizi[3] = byte.Parse(key.Substring(4, 2));
             key_dizi[4] = byte.Parse(key.Substring(10, 2));
             key_dizi[5] = byte.Parse(key.Substring(8, 2));
             key_dizi[6] = byte.Parse(key.Substring(14, 2));
             key_dizi[7] = byte.Parse(key.Substring(12, 2));
             key_dizi[8] = byte.Parse(key.Substring(18, 2));
             key_dizi[9] = byte.Parse(key.Substring(16, 2));
             key_dizi[10]= byte.Parse(key.Substring(22, 2));
             key_dizi[11]= byte.Parse(key.Substring(20, 2));
             key_dizi[12]= byte.Parse(key.Substring(26, 2));
             key_dizi[13]= byte.Parse(key.Substring(24, 2));
             key_dizi[14]= byte.Parse(key.Substring(30, 2));
             key_dizi[15]= byte.Parse(key.Substring(28, 2));
          for (int i = 0; i < block_size; i++)
          {   ipad[i] = (byte)(key_dizi[i] ^ (byte)ipad[i]);
             opad[i] = (byte)(key_dizi[i] ^ (byte)opad[i]);}
         if (seç == 0)
            return SHA1_Algoritma.Şifrele(Converter.toString(opad) +
SHA1_Algoritma.Şifrele(mesaj + Converter.toString(ipad)));
         else if (seç == 1)
```

```
          return MD5_Algoritma.Şifrele(Converter.toString(opad) +
MD5_Algoritma.Şifrele(mesaj + Converter.toString(ipad)));
         else
       return CHA_Algoritma.Şifrele(Converter.toString(opad) +
CHA_Algoritma.Şifrele(mesaj + Converter.toString(ipad)));}}}
using System;
using System.Collections.Generic;
using System.Text;
namespace otp_dogrulama
{    class HOTP    {
       static int parola_uzunluğu = 8;
       public static int Sayaç = 0;
    public static string Şifrele(int counter, string Şifre,int seçim)
    {      String key,mesaj;
           mesaj = counter.ToString();
           key = Şifre;
           String hmac_sonuç = HMAC.Şifrele(key, mesaj,seçim);
           byte[] dizi = new byte[4];
           byte[] tampon = Converter.ToByteArray(hmac_sonuç);
           dizi[0] = Converter.toWord(tampon[0], tampon[1]);
           dizi[1] = Converter.toWord(tampon[2], tampon[3]);
           dizi[2] = Converter.toWord(tampon[4], tampon[5]);
           dizi[3] = Converter.toWord(tampon[6], tampon[7]);
           dizi[0] = (byte)(dizi[0] & (byte)0x7f);
           dizi[1] = (byte)(dizi[1] & (byte)0xff);
           dizi[2] = (byte)(dizi[2] & (byte)0xff);
           dizi[3] = (byte)(dizi[3] & (byte)0xff);
           Sayaç =counter + 1;
return Converter.toInt(dizi[0], dizi[1], dizi[2], dizi[3]).ToString().Substring(1,
parola_uzunluğu).ToString(); }}}
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;
namespace otp_dogrulama{
   class MD5_Algoritma
   { public static string Şifrele(string katar)
      {   byte[] data = Converter.ToByteArray(katar);
          MD5 md = new MD5CryptoServiceProvider();
          byte[] hash = md.ComputeHash(data);
          return ConvertToHex(System.Text.Encoding.UTF7.GetString(hash, 0,
hash.Length))}
       public static string ConvertToHex(string asciiString)
       {    string hex = "";
          foreach (char c in asciiString)
          {   int tmp = c;
             hex += String.Format("{0:x2}",
(uint)System.Convert.ToUInt32(tmp.ToString()));
          } return hex.ToUpper();  }   }}
```

```
using System;
using Microsoft.Win32;
using System.Collections.Generic;
using System.Text;
namespace OTP_Smart
{    class Sayac
   {  static RegistryKey reg;
      static int sonuç = 11111111;
      public static int Oku(string clientId) {
         reg = Registry.CurrentUser.OpenSubKey("otp");
         if (reg == null)
            Yarat();
         sonuç = int.Parse(reg.GetValue(clientId).ToString());
         return sonuç;    }
      public static void Yaz(int sayi, string clientId)
      {   reg = Registry.CurrentUser.OpenSubKey("otp", true);
         if (reg == null)
            Yarat();
         reg.SetValue(clientId, sayi);  }
      static void Yarat() {
         reg = Registry.CurrentUser.CreateSubKey("otp");
         reg.SetValue("otp_smart_sayac", sonuç);}} }
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Microsoft.Win32;
namespace otp_dogrulama{
   public partial class Form2 : Form
   {   Form1 Frm1 = new Form1();
      public static int sonuç = 0;
      public static string key = "1234";
      public static string ozel_kelime = "omer";
      public Form2()
{       InitializeComponent();
      }        private void button1_Click(object sender, EventArgs e)
      { if (key == textBox1.Text.Trim())
         {if (ozel_kelime == textBox2.Text.Trim())
            Form1.giris_siniri = 0;
            label4.Text = "";
            ac(); }
         else{label4.ForeColor = Color.Red;
            label4.Text = "Yanlış Veri Girişi";}}
          public void kapat()
      {   textBox3.Visible = false;
         label3.Visible = false;
```

```
            button2.Visible = false;    }
        public void ac()
        {   textBox3.Visible = true;
            label3.Visible = true;
            button2.Visible = true;}
        private void Form2_Load(object sender, EventArgs e)
        {   kapat();    }
        private void button2_Click(object sender, EventArgs e)
        {   int Sayac;
            Sayac = int.Parse(textBox3.Text);
            OTP_Smart.Sayac.Yaz(Sayac, "otp_smart_sayac");
            Frm1.textBox1.Enabled = true;
            Frm1.button1.Enabled = true;
            Frm1.Show();
            this.Close();       }       }    }}
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;
namespace otp_dogrulama
{   class SHA1_Algoritma
    { public static string Şifrele(string katar)
        {    byte[] data = Converter.ToByteArray(katar);
            SHA1 sha = new SHA1CryptoServiceProvider();
            byte[] hash = sha.ComputeHash(data);
            return ConvertToHex(System.Text.Encoding.GetEncoding("iso-8859-
1").GetString(hash, 0, hash.Length));  }
        public static string ConvertToHex(string asciiString)
        {    string hex = "";
            foreach (char c in asciiString)
            { int tmp = c;
                hex += String.Format("{0:x2}",
(uint)System.Convert.ToUInt32(tmp.ToString()));
            }           return hex.ToUpper(); } }}
using System;
using System.Collections.Generic;
using System.Text;
namespace otp_dogrulama{
    class Converter    {
        public static byte toWord(byte sayi1, byte sayi2)
        {return (byte)((sayi1 << 8) | (sayi2 & 0xff)); }
 public static int toInt(byte sayi1, byte sayi2, byte sayi3, byte sayi4)
 { return (int)((sayi1 << 24) | (sayi2 << 16) | (sayi3 << 8) | (sayi4 & 0xff));       }
        public static byte[] ToByteArray(string str)
        {  byte[] dizi = new byte[str.Length];
            int index = 0;
            foreach (char c in str)
            {  dizi[index] = (byte)c;
                index++;            }
```

```
            return dizi;        }
        public static string toString(byte[] data)
        {       string s = "";
            foreach (byte b in data)
            {  s += (char)b;          }
            return s;        }   }}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace otp_dogrulama{
    class CHA_Algoritma    {
        static uint[] M = new uint[32];
          static float TentMap(float x, float r)
        {    if (x >= 0)
            { if (x < 0.5)
                    return r * x;
                else   if (x <= 1)
                        return r * (1 - x);            }
            return 0f;        }
        static public string Şifrele(string mesaj)
        {    MesajFormat(mesaj);
            uint[] tampon = H(31);
            string sonuç = "";
            for (int i = 0; i < 4; i++)
            {  string t = Convert.ToString(tampon[i], 16);
              sonuç += t;    }
            return sonuç;   }
        static void MesajFormat(string mesaj)
        {  string bits = ToBits(mesaj);
            if (bits.Length < 1024)
            {        bits += "1";
                while (bits.Length < 1024)
                    bits += "0";           }
            for (int i = 0; i < 32; i++)
            { M[i] = Convert.ToUInt32(bits.Substring(i * 32, 32), 2);
            }        }
         static uint[] H(int n)
        { uint[] tampon = new uint[4];
            if (n > 1)
            {tampon = H(n - 1);
                float k = K(n - 1);
              for (int i = 0; i < 4; i++)
    tampon[i] = tampon[i] ^ (uint)(k * 1024 * 1024 * 1024);
                return tampon;           }
            else           {
```

```
            tampon[0] = Convert.ToUInt32("67452301", 16);
            tampon[1] = Convert.ToUInt32("efcdab89", 16);
            tampon[2] = Convert.ToUInt32("98badcfe", 16);
            tampon[3] = Convert.ToUInt32("10325476", 16);
            float k = K(0);
            for (int i = 0; i < 4; i++)
               tampon[i] = tampon[i] ^ (uint)(k * 1024 * 1024 * 1024);
            return tampon;          }       }
static float K(int n)
      {     if (n == 0)
            return TentMap(0.3f, Normalize((double)M[n]));
         else
  return TentMap(K(n - 1), Normalize((double)M[n - 1]));     }
      static float Normalize(double girdi)
      {  double sonuç = Math.Round(girdi % 0.585d, 15);
         sonuç += 1.414d;
         return (float)sonuç;   }
      static string ToBits(string str)       {
         byte[] bytes = Converter.ToByteArray(str);
         String binary = "";
         for (int j = 0; j < bytes.Length; j++)
         {   byte b = bytes[j];
            int val = b;
            for (int i = 0; i < 8; i++)
            { binary += (val & 128) == 0 ? 0 : 1;
               val <<= 1;    }          }
         return binary;       }   }}
namespace otp_dogrulama
{    public partial class Form2 : Form
   { Form1 Frm1 = new Form1();
      public static int sonuç = 0;
      public static string key = "1234";
      public static string ozel_kelime = "admin";
      public Form2()
      {InitializeComponent();  }
      private void button1_Click(object sender, EventArgs e)
      {  if (key == textBox1.Text.Trim())
         {if (ozel_kelime == textBox2.Text.Trim())
           Form1.giris_siniri = 0;
            label4.Text = "";
            ac();       }
         else{
            label4.ForeColor = Color.Red;
            label4.Text = "Yanlış Veri Girişi";} }
      public void kapat()
      {  textBox3.Visible = false;
         label3.Visible = false;
         button2.Visible = false;}
      public void ac()       {
```

```
    textBox3.Visible = true;
    label3.Visible = true;
    button2.Visible = true;     }

private void Form2_Load(object sender, EventArgs e)
{kapat();  }
private void button2_Click(object sender, EventArgs e)
{   int Sayac;
    Sayac = int.Parse(textBox3.Text);
    OTP_Smart.Sayac.Yaz(Sayac, "otp_smart_sayac");
    Frm1.textBox1.Enabled = true;
    Frm1.button1.Enabled = true;
    Frm1.Show();
    this.Close();      }
```

**EK–3: FriendlyARM Mini2440**


**Mini2440 + 3.5" Renkli LCD & Dokunmatik Ekran**

**Teknik Özellikleri:**

- **Dimension:** 100 x 100 mm
- **CPU:** 400 MHz Samsung S3C2440A ARM920T (Max freq. 533 MHz)
- **RAM:** 64 MB SDRAM, 32 bit 100 MHz Bus
- **Flash:** 256 MB NAND Flash and 2 MB NOR Flash ile BIOS
- **EEPROM:** 1024 Byte 24C08 (I2C)
- **Ext. Memory:** SD-kart girişi
- **Serial Ports:** 1x DB9 bağlantı (RS232), total: 3x seri port, PCB üzerinde
- **USB:** 1x USB Host, 1x USB Sürücü
- **Audio Output:** 3.5 mm stereo jack
- **Audio Input:** on PCB + kondansatör mikrofon
- **Ethernet:** RJ-45 10/100M (DM9000)
- **RTC:** Dahili Pilli Gerçek Zamanlı Saat
- **Beeper:** On board PWM buzzer
- **Camera:** 20 pin Kamera etkileşimi
- **LCD Interface:**

**STN Displays:** 4 bit çift tarama, 4 bit tekli tarama veya 8 bit tekli tarama, ekranı türü Tek renkli, 4 gri düzeyleri, 16 gri düzeyleri, 256 renk veya 4096 renk Max: 1024x768, 4096 renk

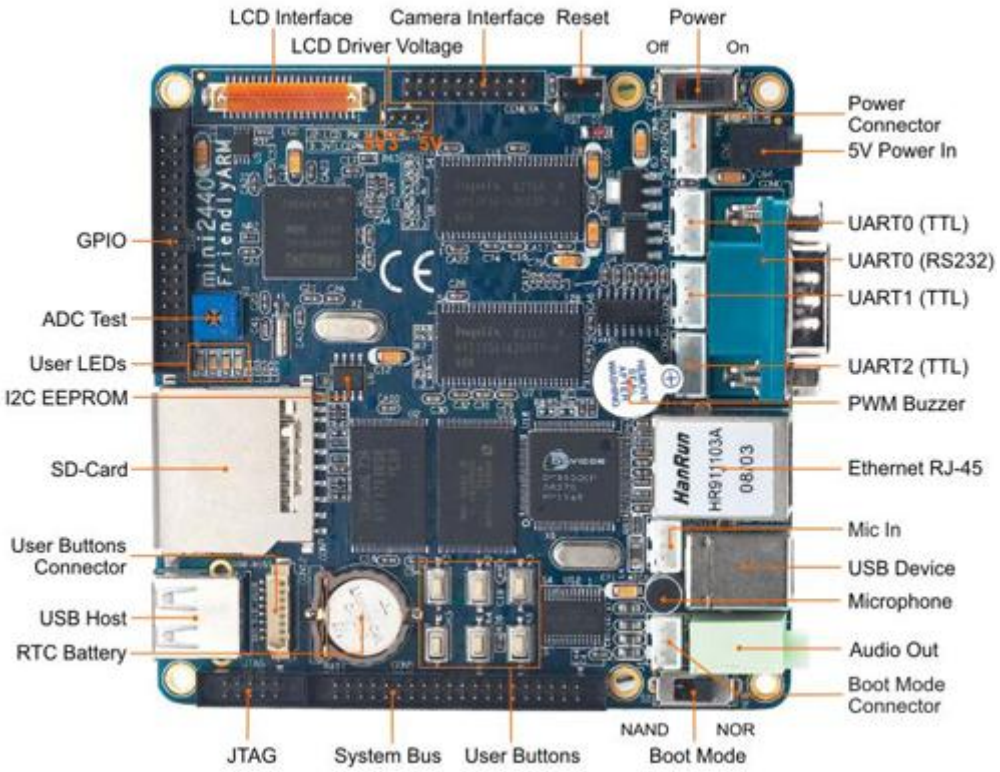
**TFT Displays:** 1, 2, 4 or 8 bpp paletli renkli görüntü16 ve 24 bpp paletsiz gerçek renkli ekran, Max: 1024x768, 64k renk.

- **Touch Panel:** 4 tel direnci
- **User Inputs:** 6x itme tuşları ve A/D pot
- **User Outputs:** 4x LEDs
- **Expansion:** 40 pin Sistem Bus, 34 pin GPIO (2.0mm)
- **Debug:** 10 pin JTAG (2.0mm)
- **Power Connector:** 5V with switch and LED
- **OS Support**

Android
Linux 2.6
Windows CE 5.0 , 6.0

**Paket İçeriği:**

- Mini2440 + 3.5 inch LCD & Touch Panel
- USB A-B Kablo
- RJ45 Patchcable
- Seri Kablo
- LPT - JTAG Adaptör
- Dokunmatik kalem
- Güç adaptörü



# ÖZGEÇMİŞ

1981-Diyarbakır doğumlu olan Ömer TÜRK ilk, orta ve lise öğrenimimi Diyarbakır'da tamamladı. 2000 yılında kazandığı Kırgızistan-Türkiye Manas Üniversitesi Bilgisayar Mühendisliği Bölümünden 2005 yılında mezun oldu. 2009 yılından itibaren Mardin Artuklu Üniversitesi Midyat Meslek Yüksekokulu Bilgisayar Programcılığı programında Öğretim Görevlisi olarak görev yapmaktadır. 2009 yılında Fırat Üniversitesi, Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Yazılım Anabilim Dalında yüksek lisans yapma hakkı kazanmıştır.