T.C
TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KRİPTOGRAFİK HASH FONKSİYONLARININ İNCELENMESİ

Cihat ERDOĞAN
Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Yrd. Doç. Dr. M. Tolga SAKALLI

Edirne 2012

T.C

TRAKYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

KRİPTOGRAFİK HASH FONKSİYONLARININ İNCELENMESİ

Cihat ERDOĞAN

Yüksek Lisans Tezi

Bilgisayar Mühendisliği Anabilim Dalı

Bu tez 09/07/ 2012 tarihinde aşağıdaki jüri tarafından kabul edilmiştir.

Yrd. Doç. Dr. M. Tolga SAKALLI
Danışman
Juri Başkanı

Yrd. Doç. Dr. Ercan BULUŞ
Üye

Yrd. Doç. Dr. Tarık YERLİKAYA
Üye

T.C
TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KRİPTOGRAFİK HASH FONKSİYONLARININ İNCELENMESİ

Cihat ERDOĞAN
Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Yrd. Doç. Dr. M. Tolga SAKALLI

Edirne 2012

Yüksek Lisans Tezi
Trakya Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Bölümü

ÖZET

Gün geçtikçe gelişen teknoloji ve bilim sayesinde bundan bir kaç sene önce tasarlanan kriptografik modeller günümüzde geçerliliğini yavaş yavaş yitirmektedir. Çünkü bu modeller zamanla güvenilirliklerini kaybetmektedir ve günün gerektirdiği zorunlulukları, gereklilikleri karşılayamamaktadır.

Kriptografik hash fonksiyonları sayısal imza, dosya bütünlüğü, şifre koruması, kimlik doğrulama protokolleri, rootkit tespiti gibi birçok alanda kullanılmaktadır. Günümüze kadar en yaygın olarak kullanılan ve bilinen kriptografik hash fonksiyonları MD ailesi (MD2, MD3, MD4 ve MD5) ve SHA ailesidir (SHA0, SHA1, SHA2). Wang ve ekibinin MD ve SHA ailesine yaptığı saldırılar MD5 ile SHA0 ve SHA1'i tehdit etmektedir. Bu tehdit SHA2 hash fonksiyonunun güvenliğini de şüpheye düşürmüştür. Bu yüzden NIST, daha güvenli bir özet fonksiyonu tasarlanması için, açık katılımlı bir yarışma düzenlemiştir.

2012 yılının sonunda açıklanması beklenen yarışma sonucu güvenlik alanında çalışan birçok bilim adamı tarafından dikkatle takip edilmektedir. Bu tez hash fonksiyonlarının yapısının daha iyi anlaşılması için yaygın olarak kullanılmakta olan MD5 fonksiyonun küçültülmüş bir örneğinin tasarımı üzerinedir.

Anahtar Kelimeler: Özet Fonksiyonlar, Özet Fonksiyonların Uygulamaları, Özet Fonksiyonların Tasarımı, Özet Fonksiyonların Analizi, MD5 Özet Fonksiyonu

Yıl: 2012
Sayfa: 96

Msc. Thesis
Trakya University Graduate School of Natural and Applied Sciences
Department of Computer Engineering

ABSTRACT

Thanks to ever-developing technology and science, cryptographic models designed a few years ago are losing their validity gradually today since these models lose credibility over time and they don't satisfy the requirements any more needed by the day.

Cryptographic hash functions are used in many fields such as digital signatures, file integrity, password protection, authentication protocols and rootkit detection. Until now, the most widely used and well-known cryptographic hash functions are MD (MD2, MD3, MD4 and MD5), and SHA families (SHA0, SHA1, SHA2). Wang and his team have attacked on MD and SHA families and threatened MD5, SHA0 and SHA1. This threat has also reduced the security of SHA2 hash function. Therefore, NIST has organized open participation contest to design of a more secure hash function.

The contest's result is expected to be disclosed by the end of 2012 and is also to be followed by many scientists working in the field of security. The purpose of this thesis is to design a minimized MD5 function for a better understanding of the structure of widely used hash functions.

Keywords: Hash Functions, Applications of Hash Functions, Design of Hash Functions, Analysis of Hash Functions, MD5 Hash Function

Year: 2012
Page: 96

ÖNSÖZ

Bu çalışmanın hazırlanmasında bana yol gösteren, her an farklı bakış açıları ve yöntemlerle ufkumu açan, her türlü destek ve yardımını sürekli aldığım değerli danışman hocam Yrd. Doç. Dr. M. Tolga SAKALLI'ya, çalışmalarımda ve Yüksek Lisans eğitimim boyunca bana yardımcı olan değerli hocam Yrd. Doç. Dr. Ercan BULUŞ'a, güncel bilgi ve becerilerle bizleri aydınlatan tüm hocalarıma, gerek tezim için yapmış olduğum her türlü araştırmaların gerçekleştirilmesinde yardımcı olan tüm arkadaşlarıma ve her an yanımda olan sevgili eşime sonsuz teşekkürlerimi sunarım.

Temmuz 2012 Cihat ERDOĞAN

İÇİNDEKİLER TABLOSU

ÖZET iv
ABSTRACT v
ÖNSÖZ vi
TABLOLAR LİSTESİ ix
ŞEKİLLER LİSTESİ x
KISALTMALAR ve SİMGELER LİSTESİ xi
1. GİRİŞ 1
2. MATEMATİKSEL ALT YAPI 2
1.1. Kümeler 2
2.1.1. Alt Kümeler 2
2.1.2. Sonlu Kümeler 3
2.2. Kümelerin Kartezyen Çarpımı 3
2.3. Bağıntı 4
2.4. Fonksiyonlar 4
2.4.1. Bire Bir Fonksiyon 5
2.4.2. Örten Fonksiyon 6
2.5. Bölme Algoritması 8
2.6. Asal Sayılar 8
2.7. Modulo İşlemi 8
2.8. Saymanın Temel Kuralları 9
2.8.1. Toplama Yolu ile Sayma 9
1.8.2. Çarpma Yolu ile Sayma 10
2.8.3. Faktöriyel Kuralı 10
2.9. Permütasyon, Kombinasyon ve Olasılık 11
2.9.1. Permütasyon 11
2.9.2. Tekrarlı Permütasyon 11
2.9.3. Dairesel Permütasyon 12
2.9.4. Kombinasyon 13
2.9.5. Olasılık 13
2.10. Sayı Sistemleri 16
2.10.1. Ondalık Sayı Sistemi 16
2.10.2. İkili Sayı Sistemi 17
2.10.3. Onaltılık Sayı Sistemi 19
3. KRİPTOGRAFİK HASH FONKSİYONLARI 21
3.1. Hash Fonksiyonlarının Genel Sınıflandırması 22

3.1.1. Değişiklik Bulma Kodları (Modification DetectionCodes-MDCs) 22
3.1.2. Mesaj Kanıtlama Kodları (Message Authentication Codes-MACs) 23
3.2. Hash Fonksiyonlarında Temel Kavram Ve Özellikler 24
3.2.1. Ön Görüntü Direnci (Preimage Resistance)..... 24
3.2.2. İkinci Ön Görüntü Direnci (2nd Preimage Resistance) 24
3.2.3. Çakışma Direnci (Collision Resistance) 24
3.3. Hash Fonksiyonlarının Özellikleri Arasındaki İlişkiler..... 27
3.4. Tekrarlı Hash Fonksiyonlarının Genel Yapıları..... 28
3.5. Hash Fonksiyonlarında Biçimlendirme ve Başlangıç Bileşenleri..... 29
3.6. Anahtarsız Hash Fonksiyonları (MDCs)..... 30
3.6.1. Blok Şifrelere Dayanan Hash Fonksiyonları..... 31
3.6.2. Alışılmış Hash Fonksiyonları..... 33
3.6.3. Modüler Aritmetiğe Dayanan Hash Fonksiyonları 42
3.7. Anahtarlı Hash Fonksiyonları (MACs)..... 42
3.7.1. Blok Şifrelere Dayanan MACs..... 43
3.7.2. MDC'ye Dayanan MAC(MDCs'den MACs' in Oluşturulması) 44
3.7.3. Hash'e Dayalı MAC 45
3.8. Bilinen Bazı Kriptografik Hash Algoritmalarının Özellikleri..... 46
3.9. Hash Algoritmalarında Güvenlik ve Saldırı Çeşitleri..... 46
3.9.1. Bir MDC'nin Bit Boyutuna Saldırı 52
3.9.2. MAC Anahtar Uzayına Saldırı 53
3.9.3. Bir MAC'in Bit Boyutuna Saldırı 53
4. HASH FONKSİYONLARININ UYGULAMA ALANLARI..... 55
4.1. Dosya Bütünlüğü 55
4.2. Şifre Koruması..... 56
5. KÜÇÜLTÜLMÜŞ MD5 ALGORİTMASI: MİNİ-MD5 58
5.1. Mini-MD5'te Ekleme (Padding) İşlemi..... 58
5.2. Mini-MD5'te Başlangıç Değerleri (Initial Values) 60
5.3. Mini-MD5'te Kullanılan İlkel Fonksiyonlar..... 60
5.4. Mini-MD5'in Çalışma Şekli..... 61
6. SONUÇ 81
KAYNAKLAR..... 82
ÖZGEÇMİŞ..... 85

TABLOLAR LİSTESİ

Tablo 2.1. Bilgisayar Bilimlerinde Kullanılan Aritmetik Fonksiyonlar 7
Tablo 2.2. Örneklem Uzay 15
Tablo 2.3. Hexadecimal Sembollerin Binary Karşılıkları 20
Tablo 3.1. MD4'e Dayalı Seçilmiş Hash Fonksiyonlarının Özeti 33
Tablo 3.2. MD5'de Kullanılan İlkel Fonksiyonlar 35
Tablo 3.3. Güvenli Hash Algoritmalarının Özellikleri 37
Tablo 3.4. ASCII Tablosu 41
Tablo 3.5. Kriptografik Hash Algoritmalarının Özellikleri 46
Tablo 3.6. *t*-bitlik MAC Anahtarlı, n-bitlik Hash Fonksiyonlarının Dizayn Amacı. 52
Tablo 5.1. Mini-MD5 ve MD5 Fonksiyonlarının Karşılaştırılması 58
Tablo 5.2. Başlangıç Değerleri 60
Tablo 5.3. Mini-MD5 İçerisinde Kullanılan İlkel Fonksiyonlar 61
Tablo 5.4. Şekil 5-3'te Kullanılan İfadelerin Anlamları 62
Tablo 5.5. Başvuru Çizelgesi (LookupTable) 62
Tablo 5.6. Mini-Md5 Fonksiyonu Oluşturan İlkel Fonksiyonların Değerleri 63

ŞEKİLLER LİSTESİ

Şekil 2.1. f Fonksiyonunun Venn Şeması (Bire Bir Fonksiyon) 6
Şekil 2.2. f Fonksiyonunun Venn Şeması (Örten Fonksiyon) 7
Şekil 2.3. Ondalık Sayı Sisteminin Genel Biçimi 17
Şekil 2.4. Ondalık Sayı Sisteminin Kuvvet Serisi Şeklindeki Açılımı 17
Şekil 2.5. İkili Sayı Sisteminin Genel Biçimi 18
Şekil 2.6 İkilik Sayı Sisteminin Kuvvet Serisi Şeklindeki Açılımı 18
Şekil 2.7. Onaltılık Sayı Sisteminin Genel Biçimi 19
Şekil 2.8. Onaltılık Sayı Sisteminin Kuvvet Serisi Şeklindeki Açılımı 19
Şekil 3.1. Dijital İmzanın Çalışma Şekli 21
Şekil 3.2. Hash Fonksiyonlarının ve Uygulama Alanlarının Sınıflandırılması 23
Şekil 3.3. MD5 Ana Döngüsü 35
Şekil 3.4. MAC Algoritmasına Dayalı CBC 45
Şekil 3.5. Saldırganların Amaçları 51
Şekil 4.1. Dosya Bütünlüğü 55
Şekil 4.2. Dosya Bütünlüğü Kontrol Programı 56
Şekil 4.3. Web Ara Yüzü 57
Şekil 4.4. Kullanıcı Verileri Tablosu 57
Şekil 5.1. 128 Bitlik Bir Bloğun Yapısı ve Ekleme (Padding) İşlemi 59
Şekil 5.2. 128 Bitlik Bir Veriye Uygulanan Ekleme İşlemi 59
Şekil 5.3. Mini-MD5 Fonksiyonun Çalışma Şekli 61
Şekil 5.4. İlkel Fonksiyonun İşlem Adımları 1 65
Şekil 5.5.İlkel Fonksiyonun İşlem Adımları 2 65

KISALTMALAR ve SİMGELER LİSTESİ

ASCII: American Standard Code for Information Interchange

CBC: Zincirleme Blok Şifre (Cipher Block Chaining)

ÇDHF: Çakışma Dirençli Hash Fonksiyonları

DSA: Sayısal İmza Algoritması

EBCDIC: Genişletilmiş İkilik kodlu Ondalık Değişim Kodu(Extended Binary Coded Decimal Interchange Code)

MAC: Mesaj Kanıtlama Kodları (Message Authentication Codes)

MDC: Değişiklik Bulma Kodları (Modification Detection Codes)

MD5: Message Digest Algorithm 5

NIST: National Institute of Standards and Technology

TYHF: Tek Yönlü Hash Fonksiyonları

TYF: Tek Yol Fonksiyonu

SHA: Güvenli Özet Algoritması (Secure Hash Algorithm)

IV: Başlangıç Değerleri (Initial Values)

$\mathbb{N}$: Doğal Sayılar Kümesi

$\mathbb{R}$: Reel Sayılar Kümesi

$\mathbb{Z}$: Tam Sayılar Kümesi

$\in$: Elemanıdır

$\wedge$: Bit düzeyinde "ve" işlemi

$\vee$: Bit düzeyinde "veya" işlemi

$\oplus$: Bit düzeyinde "xor" işlemi

$\neg$: Bit düzeyinde "değil" işlemi

|| : Bağlantı İşlemi

1. GİRİŞ

Bu tez, hash fonksiyonlarının yapısının daha iyi anlaşılması için yaygın olarak kullanılmakta olan MD5 fonksiyonun küçültülmüş bir örneğinin tasarımı üzerinedir. Bunun için öncelikle hash fonksiyonlarının tasarımında kullanılan matematiksel alt yapı anlatılmıştır. Sonra hash fonksiyonlarının yapısından bahsedilmiş ve bu fonksiyonların tasarımında kullanılan yöntemlere değinilmiştir. Ardından hash fonksiyonlarının kullanım alanları açıklanmıştır. Sonraki bölümde tasarlamış olduğumuz *Küçültülmüş MD5 Algoritması: Mini-MD5*'in yapısı anlatılmıştır. Sonuç bölümde ise yapılmış olan çalışmanın sağladığı faydalardan bahsedilmiştir.

Teknolojinin hızla geliştiği günümüzde bu gelişimin sonucu olarak ortaya çıkan güvenlik açıklarının ne kadar önemli olduğu bilinmektedir. Bu güvenlik boşluğunu kapatmaya çalışan kriptoloji, şahıslar ve kamu kurumları arasında olan iletişim sistemlerinin, oluşumundaki ve işleyişindeki her türlü dalla alakalıdır.

Modern kriptolojide temel rol oynayan *kriptografik hash fonksiyonları*, geleneksel hash fonksiyonları ile bağlantılı olmasına rağmen çoğunlukla kriptografik olmayan bilgisayar uygulamalarında da kullanılırlar. Tüm bu durumlarda da ortak olan büyük tanım bölgelerinin küçük değer bölgelerine dönüştürülmesidir. Hash fonksiyonu girdi olarak bir mesajı alır ve *hash değeri, özet değeri*, *hash veya özet* ile belirtilen bir çıktı üretir. Başka bir deyişle bir hash fonksiyonu keyfi sonlu boyutlu bit şeritlerini *n-bit* diyebileceğimiz sabit uzunluklu şeritlere dönüştürür. T tanım bölgesi ve D değer bölgesi için h: $T \rightarrow D$ ve $|T| > |D|$'dir. Burada çakışmaların olması yani aynı çıktıya sahip girdi çiftlerinin bulunması kaçınılmazdır. h, *t-bitlik* girdilerin ($t > n$) tanım bölgesine sınırlandığı taktirde yaklaşık 2^{t-n} girdi birer çıktıya dönüşecektir. Rasgele seçilen iki girdinin aynı çıktıyı sağlama olasılığı yaklaşık 2^{-n} olup t'den bağımsız olacaktır. Bir kriptografik hash fonksiyonu; kanıtlama, mesaj bütünlüğü gibi farklı bilgi güvenliği uygulamalarını daha kullanışlı hale getirmek için ilave güvenlik özelliklerinin eklendiği bir hash fonksiyonudur. Kriptografik hash fonksiyonlarının temel fikri bir hash değerinin, bir girdi şeridi için bazen damga, sayısal parmak izi veya mesaj özeti olarak da adlandırılan yoğun temsilcisi görevini görmesidir. En iyi bilinen hash fonksiyonları MD5 ve SHA'dır.

2. MATEMATİKSEL ALT YAPI

Bu bölümde tez esnasında kullanılan ve tezin anlaşılması açısından önemli olan bazı matematiksel tanım ve teorilere yer verilmiştir. Bu tanım ve teorilerin ispatları (Grimaldi, 2004)'den elde edinile bilinir.

1.1. Kümeler

Tanım 2.1: Eğer $x \in y$ olacak şekilde bir y nesnesi varsa, x'e bir küme denir. Küme, hangi nesnelerden oluştuğu tam olarak bilinen en basit matematiksel yapıdır. Kümede yer alan nesnelere de öğe veya üye denir (Lipschutz and Schiller, 1998).

Burada önemli olan kümenin sınırlarının belirlenmiş olmasıdır. Örneğin 3'ten büyük ve 9'dan küçük tek sayılar kümesinin sınırları bellidir ve 5 bu kümenin elemanıyken, 9 bu kümenin elemanı değildir.

2.1.1. Alt Kümeler

Tanım 2.2: A ve B iki küme olmak üzere A'nın her elemanı B'nin de elemanı oluyorsa, A'ya B'nin alt kümesi denir ve $A \subseteq B$ şeklinde gösterilir. B'ye de A'nın kapsayan kümesi denir. A, B kümeleri için $A \subseteq B$ ve $A \neq B$ ise, A'ya B'nin öz alt kümesi denir ve bu durum için $A \subset B$ şeklinde yazılır. Her küme kendisinin bir alt kümesidir. Boş küme her kümenin alt kümesidir.

2.1.2. Sonlu Kümeler

Tanım 2.3: s(A), A kümesinin eleman sayısını göstermek üzere; s(A) = N için N bir doğal sayı ise A kümesi sonlu bir kümedir, değil ise sonsuz bir kümedir denir. Sonlu kümenin elemanları sayılabilir fakat sonsuz kümenin elemanları sayılamaz. Örneğin 1000'den küçük çift tamsayılar kümesi sonlu bir küme, ℕ = {1, 2, 3, …}doğal sayılar kümesi sonsuzdur.

Bu bölümde küme teorisini derinlemesine incelenmemiş olup, sadece diğer konuların daha iyi anlaşılması için küme teorisinden yüzeysel olarak bahsedilmiştir.

2.2. Kümelerin Kartezyen Çarpımı

Tanım 2.4: A ve B herhangi iki küme olsun. $x \in A$ ve $y \in B$ olmak üzere *(x, y)* gösterimine birinci bileşeni *x*; ikinci bileşeni *y* olan sıralı ikili adı verilir.

Tanım 2.5: A ve B herhangi iki küme olsun. $x \in A$ ve $y \in B$ olmak üzere bütün (x, y) sıralı ikililerin oluşturduğu kümeye A ve B kümelerinin kartezyen çarpımı denir ve $A \times B$ ile gösterilir.

$A \times B = \{(x, y) : x \in A \wedge y \in B\}$ olarak yazılır.

Örnek 2.2.1.

A = {1, 2, 3} ve B = {a, b} kümeleri için

$$A \times B = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$$

ve

$$B \times A = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$$

olur.

2.3. Bağıntı

Tanım 2.6: A ve B herhangi boş olmayan iki küme olsun. $A \times B$'nin her β alt kümesine A kümesinden B kümesine bir bağıntı denir. $A \times A$'nın herhangi bir alt kümesine ikili bağıntı denir.

Örnek 2.3.1:

A = {1, 2, 3} ve B = {a, b} kümeleri için

- $\{(3, b)\}$
- $\{(2, a), (2, b)\}$
- $A \times B = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$

A'dan B'ye birer bağıntıdır.

2.4. Fonksiyonlar

Tanım 2.7: A ile B boş kümeden farklı iki küme ve A kümesinden B kümesine bir bağıntı f olsun. f bağıntısı A kümesinin her elemanını, B kümesinin bir ve yalnız bir elemanına eşliyorsa, bu bağıntıya, A kümesinden B kümesine fonksiyon denir.

Bu fonksiyon f: A→ B ya da, $A \xrightarrow{f} B$ şeklinde gösterilir.

O halde, f ⊂ A x B bağıntısında;

1. Her x ∈ A için (x, y) ∈ f olacak şekilde en az bir y ∈ B vardır.
2. (x, y) ∈ f ve (x, z) ∈ f ise y = z'dir.

Bu iki koşulu sağlıyorsa, f; A den B ye bir fonksiyondur.

A kümesinden B kümesine, f bir fonksiyon olsun. A kümesine tanım kümesi, B kümesine değer kümesi denir. A kümesinin eşlendiği elemanların oluşturduğu kümeye de, A kümesinin görüntü kümesi denir. f (A) ile gösterilir ve f (A) ⊂ B'dir.

Örnek 2.4.1.

A = {-1, 0, 1, 2} ve B = {-1, 0, 1, 2, 3, 4} kümeleri verilsin. A kümesinden B kümesine bir f fonksiyonu f = {(x, y) | y = x + 1 ve (x, y) ⊂ A x B} kuralı ile tanımlanmış olsun.

Tanım kümesi: A = {-1, 0, 1, 2}

Görüntü kümesi: B = {-1, 0, 1, 2, 3, 4}

Değer kümesi: f (A) = {0, 1, 2, 3}

f fonksiyonunun liste yöntemi ile yazılışı, f = {(-1, 0), (0, 1), (1, 2), (2, 3)} olur. Burada (1, 2) ikilisi f fonksiyonunu bir elemanıdır.

2.4.1. Bire Bir Fonksiyon

Tanım 2.8: A kümesinden B kümesine bir f fonksiyonu tanımlanmış olsun. A kümesinin farklı elemanlarının görüntüleri de farklı ise f fonksiyonuna, bire bir fonksiyon denir.

O halde, A kümesinin birbirinden farklı her x_1 ve x_2 elemanları için $f(x_1) \neq f(x_2)$ ya da, $x_1 = x_2$ ise $f(x_1) = f(x_2)$ olmalıdır.

Örnek 2.4.2.

A = {-1, 0, 1, 2} ve B = {-3, -1, 1, 3, 4, } kümeleri veriliyor. f: A→B, f(x) = 2x - 1 fonksiyonunun bire bir fonksiyon olduğu gösterilmiştir.

f (x) = 2x - 1 fonksiyonunda,

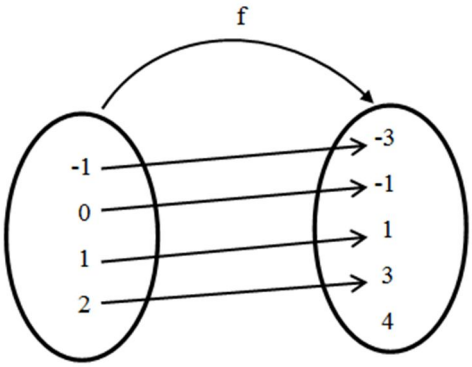
x = -1 için, f(-1) = 2(-1) - 1 = -2 - 1 = -3'tür.

x = 0 için, f(0) = 2 (0) - 1 = 0 - 1 = - 1'dir.

x = 1 için, f(1) = 2(1) - 1 = 2 - 1 = 1'dir.

x = 2 için, f(2) = 2(2) - 1 = 4 - 1 = 3'tür.

f (A) = {- 3, -1, 1, 3} olur.



Şekil 2.1. f Fonksiyonunun Venn Şeması (Bire Bir Fonksiyon)

Şekil 2.1'de görüldüğü üzere f(x) = 2x - 1 fonksiyonunda A tanım kümesinin farklı elemanları B görüntü kümesinde farklı elemanlar ile eşlenmektedir. O halde, f: A→B bire bir fonksiyondur.

Örnek 2.4.3.

g: ℝ→ℝ (ℝ: Reel Sayılar Kümesi) olmak üzere $g(x) = x^2 - x$ fonksiyonunda

x = 0 için, $g(0) = 0^2 - 0 = 0 - 0 = 0$'dır.

x = 1 için, $g(1) = 1^2 - 1 = 1 - 1 = 0$'dır.

g(0)=g(1) ve 0≠1 olduğu için g fonksiyonu bire bir fonksiyon değildir.

Pratikte hash fonksiyonlarının Örnek 2.4.3'te gösterildiği gibi çakışmalara karşı dirençli olması gerekmektedir. Hash fonksiyonlarının güvenilirliği bu tür çakışmalara karşı dirençli olmasına bağlıdır.

2.4.2. Örten Fonksiyon

Tanım 2.9: A kümesinden B kümesine bir fonksiyon tanımlanmış olsun. f(A) = B ise f fonksiyonuna örten fonksiyon denir. B değer kümesinde açıkta eleman kalmıyorsa, f bir örten fonksiyondur. O halde, y ∈ B için y = f (x) olacak şekilde en az bir x ∈ A varsa, f: A→B fonksiyonu örten fonksiyondur.

Örnek 2.4.4.

A = {2, 3, 4, 5} ve B = {-2, -1, 0, 1} kümeleri veriliyor.

f: A → B, f(x) = x - 4 fonksiyonunun örten fonksiyon olduğunu gösterelim.

f(x) = x - 4 fonksiyonunda,

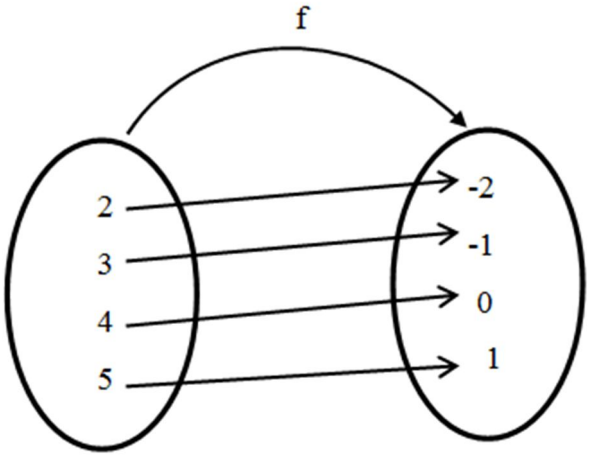
x = 2 için, f (2) = 2-4 = -2'dir.

x = 3 için, f (3) = 3- 4 = -1'dir.

x = 4 için, f (4) = 4 - 4 = 0'dır.

x = 5 için, f (5) = 5- 4 = 1'dir.

f (A) = B = { -2, -1, 0, 1} olur.



Şekil 2.2. f Fonksiyonunun Venn Şeması (Örten Fonksiyon)

Şekil 2.2'de görüldüğü gibi f(x) = x - 4 fonksiyonunda, B değer kümesinde açıkta eleman kalmadığından f: A→B örten bir fonksiyondur ve f(A) = B'dir.

Matematik biliminde yer alan birçok aritmetik fonksiyon Bilgisayar Bilimlerinde çokça kullanılmaktadır. Bunlardan bazıları Tablo 2.1'de verilmiştir.

Tablo 2.1. Bilgisayar Bilimlerinde Kullanılan Aritmetik Fonksiyonlar

Toplama fonksiyonu: a + b, a - b = a + (-b)
Çarpma fonksiyonu: a × b, a÷b= a× (1 ÷b)
Üst fonksiyonu: $x^n = \underbrace{x \times x \times x \times \ldots \times x}_{n \text{ tane}}$, n ∈ ℕ
Mutlak değer fonksiyonu:| x |, x negatif sayısı ise –x'e pozitif sayı ise x'e eşittir.
Modül fonksiyonu: a mod b, a sayısının b sayısına bölümünden kalanı ifade eder. 11 mod 2 = 1'dir.

2.5. Bölme Algoritması

Tanım 2.10: $a \neq 0$ olmak üzere a, b tamsayıları için $b = a \times q + r$; $o \leq r < |a|$ olacak şekilde bir tek q ve r tamsayı çifti vardır. Burada b'nin a'ya bölünmesiyle bir tek şekilde elde edilen q, r tamsayılarına sırasıyla bölüm ve kalan denir. Bu teoreme "Bölme Algoritması" denir.

Bölme işleminde a bölü b; "a / b" şeklinde gösterilir. "a / b" işlemi için $a = b \times n$ eşitliğini sağlanıyorsa, b a'nın bir böleni ve a da b'nin bir katıdır.

Örnek 2.5.1.
$14 = 2 \times 7$ için 2 ve 7 14'ün bölenleri, 14'te 2 ve 7'nin katıdır. Bununla beraber 1 ve 14'te 14'ün bölenleridir, $14 = 14 \times 1$. Her sayı kendisine ve 1'e bölünür.

2.6. Asal Sayılar

Tanım 2.11: $p > 1$ tamsayısı verilsin. Eğer p'nin ± 1 ve $\pm p$ den başka böleni yoksa p tamsayısı bir asal sayıdır denir. Asal olmayan bir tamsayıya bileşik sayı olarak adlandırılır.

Örnek 2.6.1.
30'dan küçük asal sayılar şunlardır: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.
6 sayısı bir birleşik sayıdır çünkü bölenlerini inceleyecek olursak 1, 2, 3 ve 6 olduğunu görürüz. Aynı zaman da 6 sayısının 2×3 şeklinde yazılabilmesi de, bileşik sayıların asal sayıların çarpımı şeklinde yazılabileceğini gösterir.

2.7. Modulo İşlemi

Modulo işlemi, hesaplamada bir sayının diğer bir sayıya bölümünden arta kalan sayıyı verir.

Tanım 2.12: n ∈ $\mathbb{Z}^+$ ve n > 0 olmak üzere a, b ∈ $\mathbb{Z}$ olsun. A'nın n'ye bölümünden kalan b ise, a'nın n'ye göre mod'u b'dir denir ve $a \equiv b \; mod \; n$ şeklinde yazılır.

Örnek 2.7.1.

$13 \equiv 3 \; mod \; 5$

$21 \equiv 3 \; mod \; 6$

2.8. Saymanın Temel Kuralları

Saymanın temel kuralları bu bölümde sırayla açıklanmıştır.

2.8.1. Toplama Yolu ile Sayma

Tanım 2.13: İki küme ortak elemana sahip değilse ($A \cap B = \emptyset \; ise$) bu iki kümeye ayrık denir. A ve B sonlu iki ayrık küme olsun.

s(A) = m ve s(B) = n → s(A∪B) = s(A) + s(B) = m + n'dir.

Buna göre, ayrık iki işlemden biri m yolla diğeri n yolla yapılabiliyorsa bu işlemlerden biri veya diğeri m + n yolla yapılabilir. Bu yönteme toplama kuralı denir.

Örnek 2.8.1.

Bir mağazada 25 çanta ve 130 ayakkabı vardır. Bir çanta veya bir ayakkabı kaç değişik yolla seçilebilir?

Çözüm: Ç kümesi çantaların A kümesi ayakkabıların kümesi olsun. Ç ve A kümesinin ortak elemanı olmadığı için bu iki küme ayrıktır.

S(Ç)=25 ve s(A)=130→ s(Ç ∪ A)= s(Ç) + s(A) = 130 + 25 = 155 değişik yolla bir çanta veya bir ayakkabı seçilebilir.

1.8.2. Çarpma Yolu ile Sayma

Tanım 2.14: Bir işlem m_1 farklı yolla, bunu izleyen işlem bir önceki işleme bağlı olarak m_2 farklı yolla, … , n inci işlem bir önceki işleme bağlı olarak m_n farklı yolla yapılabilsin. Bu olayın tamamı $m_1 \times m_2 \times \ldots \times m_n$ farklı yola yapılabilir. Buna çarpma yolu ile sayama denir.

Örnek 2.8.2.
Farklı özellikte 5 fizik, 4 kimya ve 6 matematik kitabı arasından 1 fizik, 1 kimya ve 1 matematik kitabı seçmek istiyoruz. Bu seçim kaç değişik şekilde yapılabilir?
Çözüm: Çarpma kuralına göre 5 fizik, 4 kimya ve 6 matematik kitabı arasında 1 fizik, 1 kimya ve 1 matematik kitabı
$5 \times 4 \times 6 = 120$ farklı yolla seçilebilir.

2.8.3. Faktöriyel Kuralı

Tanım 2.15: $n \in \mathbb{N}$ olmak üzere 1'den n'e kadar olan sayıların çarpımına n faktöriyel denir ve *n!* ile gösterilir.
$n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$
Not: 1! = 1 ve 0! = 1'dir.

Örnek 2.8.3.
Elimizde 5 farklı renkten oluşan (sarı, kırmızı, yeşil, mavi, beyaz) oyuncak bulunmaktadır. Bunları soldan sağa doğru kaç farklı şekilde sıralayabiliriz?
Çözüm: Öncelikle bir renk seçilir. Bu seçim 5 renkten biri olmalıdır. Örneğin sarı olsun. Geri kalan renklerden biri daha seçilir. Bu da 4 renkten biri olmalıdır. Örneğin kırmızı olsun. Bu şekilde son oyuncağa kadar seçim yapılacak olursa toplamda $5 \times 4 \times 3 \times 2 \times 1 = 5! = 120$ farklı şekilde sıralama işlemi yapılabilir.

2.9. Permütasyon, Kombinasyon ve Olasılık

Bu bölümde permütasyon, kombinasyon ve olasılık konuları ele alınmıştır.

2.9.1. Permütasyon

Tanım 2.16: Bir küme elemanlarının belirli bir sıraya göre dizilişlerinin her birine bir permütasyon denir.$r,\ n \in \mathbb{N}^+$olmaküzere n elemanlı bir kümenin, birbirinden farklı r tane elemanından oluşan sıralı r'lilerden her birine bir kümenin r'li permütasyonu denir.

$r, n \in \mathbb{N}^+$ ve $r < n$ koşulu ile, n elemanlı bir A kümesinin r'li permütasyonlarının sayısı, $P(n,r) = \frac{n!}{(n-r)!}$'dir.

Örnek 2.9.1.

Bir koşu yarışına 8 sporcu katılmıştır. İlk 3'e giren yarışmacılar kaç farklı şekilde oluşabilir?

Çözüm: 8 sporcunun 3'ü sıralamaya gireceği için, sıralama

$P(8,3) = \frac{8!}{(8-3)!} = \frac{8!}{5!} = \frac{8 \times 7 \times 6 \times 5!}{5!} = 336$ farklı şekilde oluşabilir.

2.9.2. Tekrarlı Permütasyon

Tanım 2.17: n tane nesnenin; n_1 tanesi 1. çeşitten, n_2 tanesi 2. çeşitten, n_3 tanesi 3. çeşitten, …, n_r tanesi r. çeşitten olsun.

n= $n_1 + n_2 + n_3 + \ldots + n_r$

olmak üzere n tane nesnenin n'li permütasyonlarının (farklı sıralamalarının) sayısı

$\binom{n}{n_1, n_2, n_3, \ldots, n_r} = \frac{n!}{n_1! \times n_2! \times n_3! \times \ldots \times n_r!}$'dir.

Örnek 2.9.2.

"MUTLULUK" kelimesindeki harflerin sırası değiştirilerek, anlamlı ya da anlamsız 8 harfli ve birbirinden farklı kaç grup oluşturulabilir?

Cevap:

MUTLULUK kelimesindeki harf sayısı: 8 (n = 8);

M harflerinin sayısı: 1 ($n_1 = 1$)

U harflerinin sayısı: 3 ($n_2 = 3$)

T harflerinin sayısı: 1 ($n_3 = 1$)

L harflerinin sayısı: 2 ($n_4 = 2$)

K harflerinin sayısı: 1 ($n_5 = 1$)

$$\binom{8}{1,3,1,2,1} = \frac{8!}{1! \times 3! \times 1! \times 2! \times 1!} = \frac{8 \times 7 \times 6 \times 5 \times 4 \times 3!}{3! \times 2!} = 3360$$

farklı anlamalı yada anlamsız 8 harfli birbirinden farklı grup oluşturulabilir.

2.9.3. Dairesel Permütasyon

Tanım 2.18: Birbirinden farklı n tane elemanın, bir çember etrafındaki sıralanışının her birine n tane elemanın dairesel permütasyonu denir. n tane elemandan birinin yeri sabit gibi düşünülüp diğerlerinin bu elemana göre sıralanışı göz önüne alınırsa, n tane elemanın dönel permütasyonlarının sayısı (n-1)! olur.

Örnek 2.9.3.

3 çocuk ve anne babadan oluşan 5 kişilik bir aile yuvarlak masada yemek yiyecektir. Bu aile masa etrafında kaç değişik şekilde oturabilir?

Cevap: 3 çocuk ve anne babadan oluşan 5 kişi yuvarlak masa etrafında (5-1)! = 4! = 24 farklı şekilde oturabilir.

2.9.4. Kombinasyon

Tanım 2.19: $r, n \in \mathbb{N}^+$ ve r ≤ n olmak üzere, n elemanlı bir A kümesinin r elemanlı alt kümelerinin her birine A kümesinin r'li kombinasyonu denir. n elemanlı kümenin r'li kombinasyonlarının sayısı $C(n,r) = \frac{n!}{(n-r)! \times r!}$'dir ve $C(n,r)$, $\binom{n}{r}$ ve C_n^r şeklinde gösterilir.

Örnek 2.9.4.
3 çocuk ve anne babadan oluşan 5 kişilik bir aile yuvarlak masada yemek yiyecektir. Bu aile masa etrafında kaç değişik şekilde oturabilir?
Cevap: 3 çocuk ve anne babadan oluşan 5 kişi yuvarlak masa etrafında (5-1)! = 4! = 24 farklı şekilde oturabilir.

2.9.5. Olasılık

Bu konudaki tanımlar yapılırken çoğunlukla (Erdoğan M. ve Yılmaz G.) kaynağından faydalanılmıştır.

Tanım 2.20: "Rastsal" (random) bir deneyin olabilecek tüm sonuçlarına "örneklem uzayı" (sample space), bu örneklem uzayının her bir üyesine de "örneklem noktası" (sample point) denir.

Örnek 2.9.5.

İki madeni para ile yazı-tura atma deneyinin 4 örneklem noktalı bir örneklem uzayı vardır:
Y = {YY, YT, TY, TT}

Tanım 2.21: Rastsal bir deneye ait örneklem uzayının olası her bir alt kümesine "rastsal olay" (random event) denir.

Örnek 2.9.6.
Bir yazı ve bir tura gelmesi olayı: {YT, TY}

Tanım 2.22: Bir olayın gerçekleşmesi diğer bir olayın oluşmasını önlüyorsa, bu iki olay "karşılıklı dışlamalı" (mutually exclusive) olaylardır.

Örnek 2.9.7.

{YY, YT, TY} ve {TT} karşılıklı dışlamalıdır.

Tanım 2.23: Değerleri rastsal bir deney sonucu belirlenen değişkene "rastsal değişken" (random variable) ya da kısaca "rd" (rv) denir.

- Rastsal değişkenler genellikle X, Y, Z gibi büyük harflerle ve aldıkları değerler de x, y, z gibi küçük harflerle gösterilir.
- Rastsal bir değişken ya "kesikli" (discrete) ya da "sürekli" (continuous) olur.
- Kesikli bir rd ancak sonlu sayıda farklı değerler alabilir.
  Örnek: Zar.
- Sürekli bir rd ise belli bir aralıkta her sayısal değeri alabilir.
  Örnek: Rastsal olarak seçilmiş bir kişinin boyu.

Tanım 2.24: A, örneklem uzayındaki bir olay olsun. Rastsal deney sürekli yinelendiğinde, A olayının gerçekleşme sıklık oranına A olayına ait "olasılık" (probability) denir, P(A) ya da Prob(A) ile gösterilir.

P(A) gerçek değerli bir "işlev" (function) olup, şu özellikleri taşır:

- Her A için $0 \leq P(A) \leq 1$'dir. (1 = %100)
- A; B; C; … örneklem uzayını oluşturuyorsa şu geçerlidir:
  $P(A + B + C + \dots) = 1$
- 3 A, B ve C karşılıklı dışlamalı olaylar ise şu geçerlidir:
  $P(A + B + C) = P(A) + P(B) + P(C)$

Örnek 2.9.8.

Altı yüzlü bir zarı atma deneyi düşünülsün:

Bu deneyde örneklem uzayı= {1; 2; 3; 4; 5; 6} biçimindedir ve

$P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1 = 6$'dır.

Ayrıca, $P(1) + P(2) + P(3) + P(4) + P(5) + P(6) = 1$ olur.

Tanım 2.25: Sonlu bir Ö = {ö$_1$, ö$_2$, ö$_3$, …} örneklem uzay ve P olasılık fonksiyonu olmak üzere P(ö$_1$) = P(ö$_2$) = … = P(ö$_n$) ise Ö örneklem uzayına eş olumlu (olasılı) örneklem uzay denir.

Ö eş olumlu örneklem uzay ve s(Ö) = n ise ö$_i$ ∈ Ö, ö$_i$ örneklem noktasının olma olasılığı P(ö$_i$) = $\frac{1}{n}$'dir. Ö bir eş olumlu örneklem uzay ve A ⊂ Ö olmak üzere, A olayının olma olasılığı

$$"A" \; olayının \; olma \; olasılığı \; = \frac{"A" \; kümesinin \; eleman \; sayısı}{"Ö" \; kümesinin \; eleman \; sayısı}$$

$$P(A) = \frac{s(A)}{s(Ö)}'dür.$$

Örnek 2.9.9.

Bir torbada renkleri dışında bütün özellikleri aynı olan 2 kırmızı, 4 mavi ve 2 yeşil kalem vardır. Torbadan, içine bakmadan bir kalem çekildiği zaman bunun mavi olma olasılığı nedir?

Çözüm: Kırmızı, mavi ve yeşil kalemler sırasıyla k, m, y harfleriyle gösterilsin.

Örneklem Uzay → Ö={k$_1$, k$_2$, m$_1$, m$_2$, m$_3$, m$_4$, y$_1$, y$_2$}, s(Ö)=8'dir.

A={Mavi kalem seçilmesi}={m$_1$, m$_2$, m$_3$, m$_4$}, s(A)=4

$$P(A) = \frac{s(A)}{s(Ö)} = \frac{4}{8} = \frac{1}{2}'dir.$$

Örnek 2.9.10.

2 tane 6 yüzlü hilesiz zar, hilesiz bir şekilde atıldığında aynı sayının gelme olasılığı nedir?

Çözüm: Örneklem Uzayı Tablo 2.2'de verilmiştir.

Tablo 2.2. Örneklem Uzay

II. Zar I. Zar	**1**	**2**	**3**	**4**	**5**	**6**
1	(1,**1**)	(1,**2**)	(1,**3**)	(1,**4**)	(1,**5**)	(1,**6**)
2	(2,**1**)	(2,**2**)	(2, **3**)	(2, **4**)	(2, **5**)	(2, **6**)
3	(3,**1**)	(3, **2**)	(3, **3**)	(3, **4**)	(3, **5**)	(3, **6**)
4	(4,**1**)	(4, **2**)	(4, **3**)	(4, **4**)	(4, **5**)	(4, **6**)
5	(5,**1**)	(5, **2**)	(5, **3**)	(5, **4**)	(5, **5**)	(5, **6**)
6	(6,**1**)	(6, **2**)	(6, **3**)	(6, **4**)	(6, **5**)	(6, **6**)

s(Ö)=36'dır. A={(1,1), (2,2,), (3,3,), (4,4), (5,5,), (6,6)}, s(A)=6

$$P(A) = \frac{\text{s(A)}}{s(\text{Ö})} = \frac{6}{36} = \frac{1}{6}'dır.$$

2.10. Sayı Sistemleri

Sayılar, sayma olgusunun başlaması ile birlikte insanların yaşamına girmiş ve yazının bulunmasıyla birlikte kayıt edilmeye başlanmıştır. Bu bölümde sayılar bilgisayarda kullanılış yönleriyle ele alınmıştır.

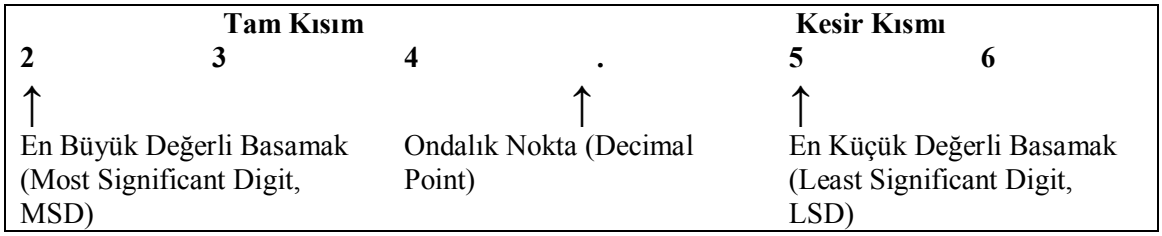
Her bilgisayarın ikili durum makinesi olması, burada kullanılan sayı sisteminin ("binary") ikili sistem olmasını getirmiştir. Hâlbuki güncel yaşamda onlu sayılar bir başka deyişle on tabanlı sayılar kullanılmaktadır. O halde onlu sayıların bilgisayarlara aktarılabilmesi için bir dönüştürme düzeneğine gereksinim duyulacaktır. Bununla beraber ikili sisteme yakınlıkları nedeniyle sekizlik, onaltılık sistemlerde bu bölümde ele alınacaktır.

2.10.1. Ondalık Sayı Sistemi

Günlük yaşantımızda kullandığımız sayı sistemi ondalık (decimal) sayı sistemidir. Ayrıca 10 tabanlı sistem olarak da adlandırılır ve bu sistemde on tane sembol kullanılır.

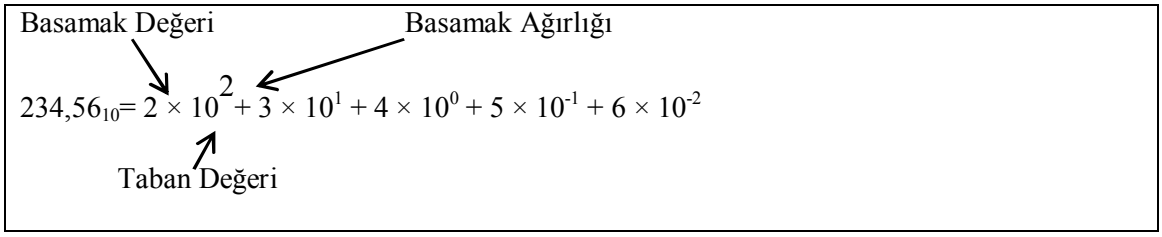
Semboller: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Ondalık sayı sisteminin genel biçimi ve terminolojisi Şekil 2.3'te verilmiştir.



Şekil 2.3. Ondalık Sayı Sisteminin Genel Biçimi

$234{,}56_{10}$= 234,56D



Şekil 2.4. Ondalık Sayı Sisteminin Kuvvet Serisi Şeklindeki Açılımı

Bu açılım iki yöne genişletilirse; a basamak ağırlığı ve basamak değeri B_a olmak üzere 10 tabanlı herhangi bir S sayısının genelleştirilmiş ifadesi şu şekilde gösterilebilir:

$$S = \sum_{a=-\infty}^{\infty} B_a \, 10^a$$

Örnek 2.10.1.

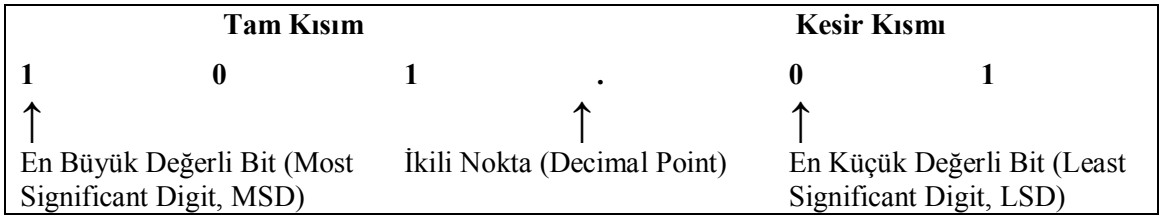
$571{,}632_{10} = 5\times 10^2 + 7\times 10^1 + 1\times 10^0 + 6 \times 10^{-1} + 3 \times 10^{-2} + 2 \times 10^{-3}$

2.10.2. İkili Sayı Sistemi

İkili (Binary) sayı sistemi, sayısal elektronik sistemlerinde yaygın olarak kullanılır. Günlük yaşantımızda kullandığımız ondalık sayı sisteminden iki yönlü dönüşüm yapılarak kullanılır. Bu sistemde, Boole cebrinde doğru ve yanlışı belirtmek üzere iki tane sembol kullanılır.

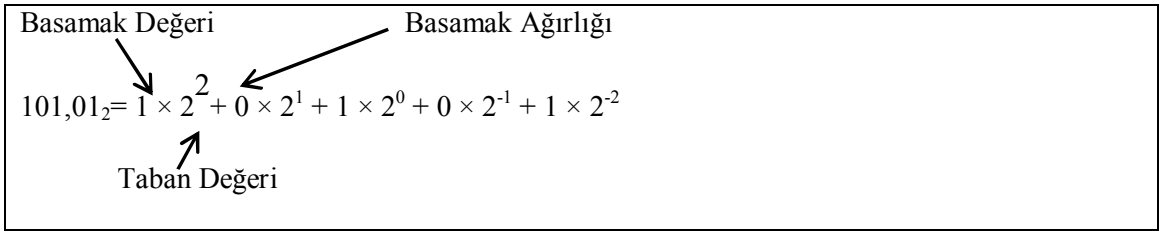
Semboller: 0,1

İkili sayı sisteminin genel biçimi ve terminolojisi Şekil 2.5'te verilmiştir.



Şekil 2.5. İkili Sayı Sisteminin Genel Biçimi

$101{,}01_2$= 101,01B



Şekil 2.6 İkilik Sayı Sisteminin Kuvvet Serisi Şeklindeki Açılımı

Bu açılım iki yöne genişletilirse; a basamak ağırlığı ve basamak değeri B_a olmak üzere 2 tabanlı herhangi bir S sayısının genelleştirilmiş ifadesi şu şekilde gösterilebilir:

$$S = \sum_{a=-\infty}^{\infty} B_a \, 2^a$$

Örnek 2.10.2.

- $10_2 = (1\times 2^1) + (0 \times 2^0) = 2_{10}$
- $11101_2 = (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0\times 2^1) + (1 \times 2^0)$
  $= 16 + 8 + 4 + 0 + 1 = 19_{10}$
- $1011{,}101_2 = (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1\times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2} + 1 \times 2^{-3})$
  $= 8 + 0 + 2 + 1 + \frac{1}{2} + 0 + \frac{1}{8} = 11 + \frac{5}{8} = 11{,}625_{10}$

2.10.3. Onaltılık Sayı Sistemi

Onaltılık (Hexadecimal, Hex) sayı sistemi, sayısal elektronik sistemlerinde mikroişlemci temelli uygulamalarda yaygın olarak kullanılır. Günlük yaşantımızda kullandığımız ondalık sayı sisteminden iki yönlü dönüşüm yapılarak kullanılır. Bu sistemde, ondalık sayı sisteminde kullanılan sembollere ek olarak, dokuzdan büyük değerlere karşılık İngiliz alfabesinin ilk beş harfi ile birlikte on altı tane sembol kullanılır.

Semboller 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Onaltılık sayı sisteminin genel biçimi ve terminolojisi Şekil 2.7'de verilmiştir.

Tam Kısım			**Kesir Kısmı**	
2	**B**	**4**	**3**	**E**
↑		↑	↑	
En Büyük Değerli Basamak (Most Significant Digit, MSD)		Onaltılık Nokta (Hexadecimal Point)	En Küçük Değerli Basamak (Least Significant Digit, LSD)	

Şekil 2.7. Onaltılık Sayı Sisteminin Genel Biçimi

On altı tabanlı sayı sisteminin gösterimi ve sayıların kuvvet serisi şeklindeki açılımına bir örnek Şekil 2.8'de verilmiştir.

Basamak Değeri — Basamak Ağırlığı

$2B4{,}3E_{16} = 2 \times 16^{2} + 11 \times 16^{1} + 4 \times 16^{0} + 3 \times 16^{-1} + 14 \times 16^{-2}$

Taban Değeri

Şekil 2.8. Onaltılık Sayı Sisteminin Kuvvet Serisi Şeklindeki Açılımı

Bu açılım iki yöne genişletilirse; a basamak ağırlığı ve basamak değeri B_a olmak üzere 16 tabanlı herhangi bir S sayısının genelleştirilmiş ifadesi şu formülle gösterilebilir:

$$S = \sum_{a=-\infty}^{\infty} B_a \, 16^a$$

Bu ifadede taban değeri "*T*" olarak değişken alınırsa *T* tabanına göre genel durum şu şekilde elde edilebilir:

$$S = \sum_{a=-\infty}^{\infty} B_a T^a$$

Örnek 2.10.3.

$2B4{,}3E_{16} = 2 \times 16^2 + 11 \times 16^1 + 4 \times 16^0 + 3 \times 16^{-1} + 14 \times 16^{-2}$

$= 512 + 176 + 4 + \frac{3}{16} + \frac{14}{256} = 692 + 0{,}1875 + 0{,}0546875$

$= 692{,}2421875_{10}$

Onaltılık sayı sistemi ile ikilik sayı sistemi arasında çevirme işlemi için kullanabileceğimiz hexadecimal karakterlerin binary karşılıkları Tablo2.3'te verilmiştir.

Tablo 2.3. Hexadecimal Sembollerin Binary Karşılıkları

$0000_2 = 0_{16}$	$0100_2 = 4_{16}$	$1000_2 = 8_{16}$	$1100_2 = C_{16}$
$0001_2 = 1_{16}$	$0101_2 = 5_{16}$	$1001_2 = 9_{16}$	$1101_2 = D_{16}$
$0010_2 = 2_{16}$	$0110_2 = 6_{16}$	$1010_2 = A_{16}$	$1110_2 = E_{16}$
$0011_2 = 3_{16}$	$0111_2 = 7_{16}$	$1011_2 = B_{16}$	$1111_2 = F_{16}$

Örnek 2.10.4.

- $1011_2 = B_{16}$
- $10011110_2 = 9E_{16}$ (4 bitlik parçalara ayırdığımızda $1001_2 = 9_{16}$ ve $1110_2 = E_{16}$ olduğuna dikkat ediniz.)
- $10011001_2 = 128 + 16 + 8 + 1 = 153_{10} = 99_{16}$

Hexadecimal sayı sistemleri birçok alanda kullanılmaktadır. Buna en güzel örnek web sayfalarındaki renk kodlarının hexadecimal olarak tutulmasıdır.

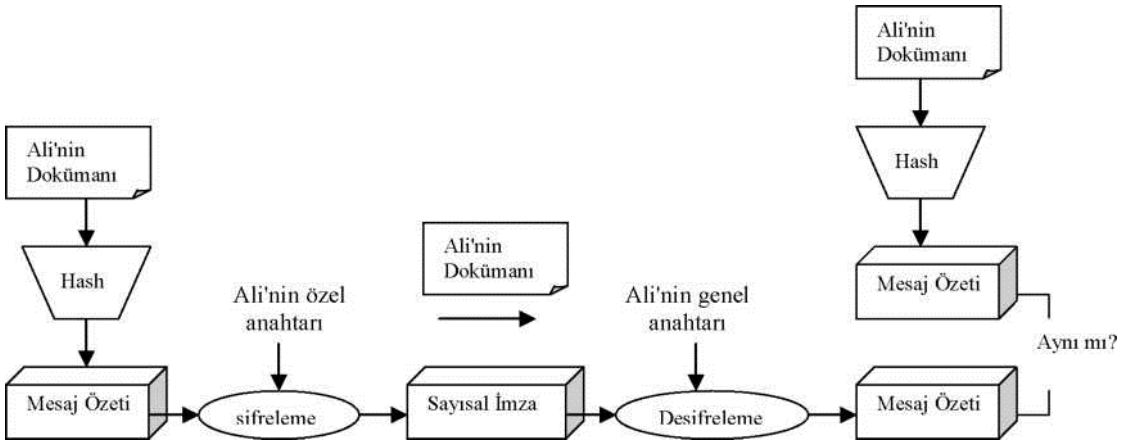
3. KRİPTOGRAFİK HASH FONKSİYONLARI

Kriptografik sistemlerin olmazsa olmaz özelliklerinden biri güvenilirliktir. Sonraki aranan özellik ise mesajı başkalarının anlayamayacağı fakat alıcının anlayabileceği şekilde göndermektir. Bundan sonra gerekli olan ise mesajı okunaklı olmayan şekilde gönderen fonksiyonlardır. Herhangi biri mesajı ele geçirirse anlayamamalı ve onu çevirememelidir. Kriptografinin böyle fonksiyonlara ihtiyacı vardır. Çünkü bu fonksiyonların ana özellikleri tersinin hesaplanmasının güç olmasıdır.

Hash fonksiyonları veri bütünlüğü ile sayısal imza tasarıları için kullanılırlar. Açık anahtarlı bir algoritmayla hazırlanan sayısal imza, gönderilen bilginin sayısal içeriğinin değiştirilmediğinin ve gönderen tarafın kimliğinin ispatı için atılır. Teknik olarak sayısal imza, imzalanmış belgenin özünü, özetini (hash) içeren, elektronik mesaja eklenmiş bilgidir. İçerikte yapılacak bir değişiklik hash'i geçersiz kılacaktır.

Bir mesaj ilk olarak özetlenir ve sonrasında hash değeri mesajın bir temsilcisi gibi orijinal mesaj yerine imzalanır. Örneğin bir x' mesajının orijinal x mesajı ile aynı olup olmadığına bakılırken; x''nün hash değeri hesaplanır ve korunan hash değeri ile kıyaslanır. Değerler eşitse girdilerinde eşit olduğu kabul edilir. Bu durumda mesaj değiştirilmemiş demektir.

Hash fonksiyonu yardımıyla bir şahsın dokümanı üzerinde sayısal imzanın çalışma ilkesini Şekil 3.1'de olduğu gibi özetleyebiliriz.



Şekil 3.1. Dijital İmzanın Çalışma Şekli

Tanım 3.1: Bir hash fonksiyonunun sahip olması gereken iki özellik şunlardır:

- *Sıkıştırma (Compression):* Keyfi sonlu bit uzunluğundaki bir *x* girdisini *n-bit* uzunluğundaki sabit *h(x)* çıktısına dönüştürür.
- *Hesaplama Kolaylığı (EasyComputation): h* ve bir *x* girdisi verildiğinde *h(x)*'i hesaplamak kolay olmalıdır.

Başka bir ifadeyle hash fonksiyonu gönderilecek mesajdan matematiksel yollarla sabit uzunlukta sayısal bilgi üretme işlemidir. Üretilen sayısal bilgi *"mesaj özeti"* olarak bilinir. Mesaj özeti anlamsız bir bilgidir. Hash fonksiyonu geri dönüşümü olmayan bir fonksiyondur; yani mesajın özetine bakarak mesajın kendisini elde etmek mümkün değildir.

3.1. Hash Fonksiyonlarının Genel Sınıflandırması

Kriptografik özet fonksiyonlarını genel olarak Anahtarlı ve Anahtarsız Hash Fonksiyonları olmak üzere 2 gruba ayrılmıştır. Bunlar da kendi içlerinde alt dallara ayrılır. Bu alt dallar sırayla bu bölümde ele alınmıştır.

3.1.1. Değişiklik Bulma Kodları (Modification DetectionCodes-MDCs)

Fazla yaygın olmamakla birlikte Mesaj Bütünlük Kodları olarak bilinen MDCs'ler, bazı kaynaklarda Manipulation Detection Codes olarak ta bilinirler. MDCs'in amacı temsilci bir görüntü veya mesajın *özetini(hash değerini)* sağlamaktır. MDCs, *anahtarsız hash fonksiyonlarının* bir alt sınıfıdır. Bu hash fonksiyonları hesaplanırken herhangi bir anahtar kullanılmaz. MDCs'in temel odaklarının belirli sınıfları şunlardır:

- Tek Yol Hash Fonksiyonları (TYHF): Belli bir hash değerine özetlenen bir girdiyi bulmak güçtür. *h(x) = y* verildiğinde *x*'i bulmak zordur.

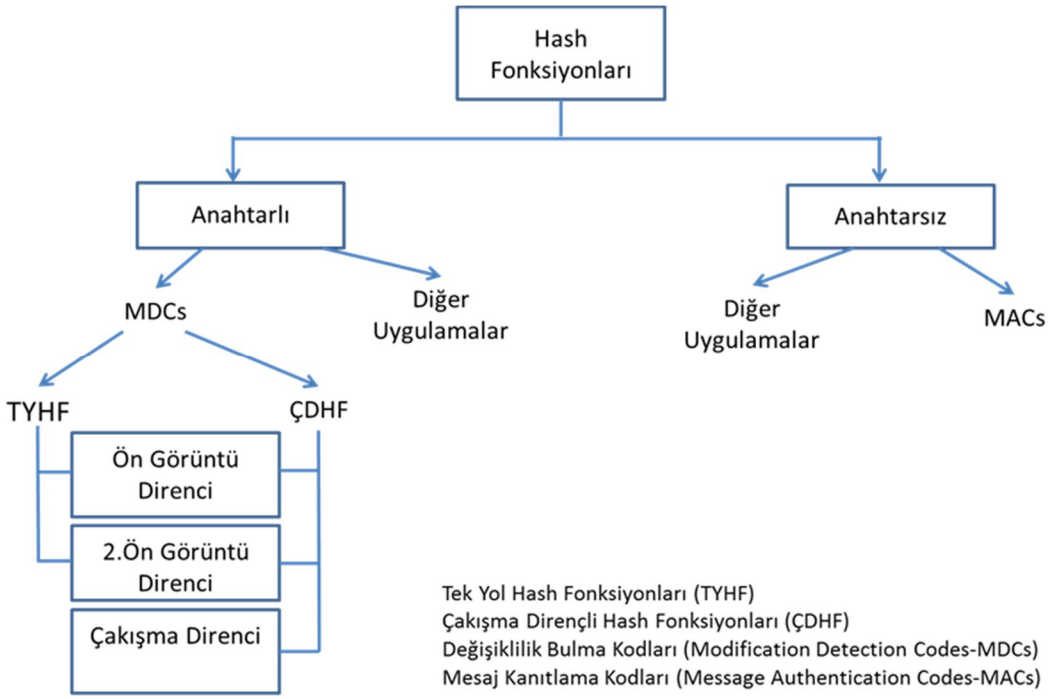
- Çakışma Dirençli Hash Fonksiyonları (ÇDHF): Aynı hash değerine sahip iki mesaj (veri) bulmak güçtür.

3.1.2. Mesaj Kanıtlama Kodları (Message Authentication Codes-MACs)

Mesaj kanıtlayıcı kodların amacı herhangi ilave bir mekanizma kullanmadan mesajın kaynağı ve bütünlüğüne ilişkin teminatların sağlanmasını kolaylaştırmaktır. MACs fonksiyonel iki parametreye sahiptir. Biri mesaj diğeri ise gizli anahtardır. Bu hash fonksiyonları *anahtarlı hash fonksiyonlarının* bir alt sınıfıdır.

MDC durumunda girdi olarak bir mesaj verildiğinde herhangi biri hash değerini hesaplayabilir. MAC durumunda ise girdi olarak bir mesaj verildiğinde ancak anahtar bilgisine sahip olan kişi hash değerini hesaplayabilir.

Şekil 3.2'de hash fonksiyonlarının yukarıda kısaca açıklanan basitleştirilmiş sınıflandırılması gösterilmiştir.



Şekil 3.2. Hash Fonksiyonlarının ve Uygulama Alanlarının Sınıflandırılması

3.2. Hash Fonksiyonlarında Temel Kavram Ve Özellikler

Hash Fonksiyonlarının temel özellikleri ve kullanılan temel kavramlar bu bölümde ele alınmıştır.

3.2.1. Ön Görüntü Direnci (Preimage Resistance)

x, x' girdi; *y, y'* çıktı ve *h* anahtarsız hash fonksiyonu olmak üzere; önceden belirtilmiş bütün çıktılar için hash değeri bu çıktı olan herhangi bir *x* girdisi hesaplanabilir olmamalıdır. Başka bir değişle; karşılık geldiği girdi bilinmeyen herhangi bir *y* verildiğinde $h(x') = y$ olacak şekilde herhangi bir *x'* ön görüntüsü bulmak kolay olmamalıdır.

3.2.2. İkinci Ön Görüntü Direnci (2nd Preimage Resistance)

Belirlenen herhangi bir girdi ile aynı çıktıya sahip ikinci bir girdinin bulunması hesaplanabilir olmamalıdır. Yani bir *x* verildiğinde $h(x) = h(x')$ olacak şekilde $x \neq x'$ olmak üzere ikinci bir x' ön görüntüsü bulmak kolay olmamalıdır.

3.2.3. Çakışma Direnci (Collision Resistance)

Aynı çıktıya sahip farklı iki x, x' girdileri bulmak kolay hesaplanabilir olmamalıdır. Yani girdilerin seçimi serbest olmak üzere $h(x) = h(x')$ olacak şekilde herhangi iki x, x' girdisi bulmak kolay olmamalıdır.

Yukarıda verdiğimiz tanımlar için alternatif olarak aşağıdaki terimler de kullanılır.

- Ön Görüntü Direnci = Tek Yol Fonksiyonu

- İkinci Ön Görüntü Direnci = Zayıf Çakışma Direnci
- Çakışma Direnci = Güçlü Çakışma Direnci

Bir sayısal imza tasarısı düşünelim ki içinde *x* mesajından daha çok *h(x)* hash değerine uygulanan imza olsun. Burada *h* bir ikinci ön görüntü direncine sahip bir MDC olmalıdır. Aksi takdirde bir *C* düşmanı, *A* şahsının *h(x)* üzerindeki imzasını gözlemleyebilir. Bu takdirde *h(x) = h(x')* olacak şekilde bir *x'* bulur ve *x'*'yü *A*'nın imzaladığını iddia edebilir. Eğer *C* gerçekten *A* nın imzaladığı mesajı seçebilirse bu takdirde *C*'nin ikinci ön görüntü bulmaktansa yalnızca bir *(x, x')* çakışma ikilisi bulması gerekir. Bu durumda çakışma direnci de gerekmektedir.

Tanım 3.2: Tek Yol Hash Fonksiyonu (TYHF)

Bir *h* hash fonksiyonu Tanım 3.1'e ek olarak ön görüntü ve ikinci ön görüntü özelliklerini sağlarsa bu fonksiyona *tek yol hash fonksiyonu* denir. Bir başka değişle;

- Keyfi sonlu boyutlu *x* girdisini *n* sabit uzunluğundaki *h(x)* çıktısına dönüştürür.
- Bir *x* girdisi verildiğinde *h(x)*'i hesaplamak kolaydır.
- *h(x) = y* olacak şekildeki *x*'i bulmak kolay olmamalıdır (Ön Görüntü Direnci).
- Bir *x* girdisi verildiğinde *h(x) = h(y)* özelliğini sağlayan *y* bulmak kolay olmamalıdır (İkinci Ön Görüntü Direnci).

Tanım 3.3: Çakışma Dirençli Hash Fonksiyonu (ÇDHF)

Tanım 3.1'e ek olarak ikinci ön görüntü direnci ve çakışma direnci özelliklerini sağlayan hash fonksiyonlarıdır.
Literatürde bu fonksiyonlar için;

- Tek Yol Hash Fonksiyonu = Zayıf Tek Yol Hash Fonksiyonu
- Çakışma Dirençli Hash Fonksiyonu = Güçlü Tek Yol Hash Fonksiyonu alternatif terimleri kullanılır.

Uygulamalar esnasında hash özelliklerinden gerçekten hangilerinin ve neden gerektiği anlaşılmalıdır. Mesela çakışma dirençli hash fonksiyonlarını oluşturmak tek yol hash fonksiyonlarını oluşturmaktan daha güçtür. Eğer güvensiz bir grup veya şahıs hash fonksiyon girdilerinin kesin içeriğini kontrol ederse bir MDC, çakışmaya dirençli

hash fonksiyonu olmalıdır. Yalnız bir şahsı içeren durumlarda tek yol hash fonksiyonu olması yeterlidir.

TYHF'larında girdi bitleri ve çıktı bitleri bağlantılı olmamalıdır. Sadece bitlerin küçük bir sayısı için $h(x) = h(x')$ olacak şekilde farklı x ve x' girdileri bulmak güç olmalıdır. Bu özelliğe *yakın çakışma direnci* diyebiliriz. Herhangi bir alt şeridi geri almak giriş girdisini geri almak kadar güç olmalıdır. Hatta girdilerin bir kısmı bilinse bile kalan kısmı bulmak zor olmalıdır. Bu özelliğe de *kısmi ön görüntü direnci* diyebiliriz. Eğer t girdi bitleri bilinmeden kalmışsa ortalama 2^{t-1} hash işlemi ile bunlar bulunmaya çalışılır.

Tanım 3.4: Tek Yol Fonksiyonu(TYF)

Bir tek yol fonksiyonu; f'nin tanım bölgesindeki her x için $f(x)$ i hesaplamak kolay olacak şekilde bir f fonksiyonudur. Fakat f'nin değer bölgesindeki bütün y'ler için $y = f(x)$ olacak şekilde herhangi bir x bulmak kolay hesaplanabilir değildir.

Yukarıda tek yol fonksiyonu, tanım bölgesi sabit boyutlu girdilere dayanan tek yol hash fonksiyonlarından farklı olarak tanımlanmıştır. Ayrıca ikinci ön görüntü direncine sahip olması gerekmez. Birçok tek yol fonksiyonu aslında sıkıştırma değildir. Çoğu görüntü elemanları tek ön görüntüye sahiptir ve bunlar için ikinci ön görüntü direncinin sağlanması mantıksızdır.

Hash fonksiyonları genellikle tek yol özelliğini gerektiren, sıkıştırma özelliğini gerektirmeyen uygulamalarda kullanılır. Bu yüzden hash fonksiyonlarını girdilerle çıktılar arasındaki ilişkiye dayanarak üç sınıfa ayırmak kullanışlı olacaktır.

1. Genel Hash fonksiyonları: Bu fonksiyonlar Tanım 3.1'deki gibi olup genellikle keyfi uzunluktaki girdileri *n-bitlik* çıktılara sıkıştıran tek yol özelliklerinin eklendiği fonksiyonlardır.
2. Sıkıştırma Hash Fonksiyonları (Sabit Boyutlu Hash Fonksiyonları): Bu fonksiyonlar Tanım 3.1'deki fonksiyonlar olup, tanım bölgesi sabit uzunlukta girdilere kısıtlanan tek yol özelliklerinin eklendiği fonksiyonlardır. Örneğin *m-bitlik* girdileri *n-bitlik* çıktılara sıkıştırır, $m > n$'dir.
3. Sıkıştırma Olmayan Tek Yol Fonksiyonları: Bu fonksiyonlar $n = m$ hariç sabit boyutlu hash fonksiyonlarıdır. Tek yol permütasyonları içerirler ve daha açık olarak, tersine çevrilemeyen hesaplanabilir fonksiyonlar olarak tanımlanabilirler.

3.3. Hash Fonksiyonlarının Özellikleri Arasındaki İlişkiler

Teorem 3.1: "Hash fonksiyonlarının çakışma direnci, ikinci ön görüntü direncini gerektirir."

İspat: h çakışma direncine sahip bir fonksiyon olsun. Bir x_j girdisi belirleyelim. Eğer h ikinci ön görüntü direncine sahip değilse bu takdirde $h(x_i) = h(x_j)$ olacak şekilde bir x_i girdisi bulunabilir. Bu durumda aynı hash çıktısına sahip farklı (x_i, x_j) girdi çifti bulunmuş olur ki bu ise çakışma direnci ile çelişki oluşturur.

Teorem 3.2: "Çakışma direnci ön görüntü direncini garanti etmez."

g çakışma dirençli bir hash fonksiyonu olsun ve keyfi uzunluktaki girdiyi *n-bit* uzunluğundaki çıktıya dönüştürsün. h fonksiyonu

$$h(x) = \begin{cases} 1 \parallel x, & x; n\ bit\ uzunluğu\ ise \\ 0 \parallel g(x), & diğer\ durumlarda \end{cases}$$

olarak tanımlansın. Burada ||, bağlantı belirtsin. Bu takdirde h, *(n +1)-bitlik* çakışma direnci olan fakat ön görüntü direncine sahip olmayan bir hash fonksiyonudur. Basit bir örnek olarak; sabit uzunluktaki girdiler üzerindeki birim fonksiyonunun çakışma ve ikinci ön görüntü direnci vardır fakat ön görüntü direnci yoktur. $h(x) = x$ birim fonksiyonunda x verildiğinde ön görüntüyü hesaplamak oldukça kolayken, $h(x) = h(x')$ olacak şekilde farklı x, x' girdileri bulmak ve x verildiğinde $h(x) = h(x')$ olacak şekilde x' girdisi bulmak oldukça güçtür. Bu örnek gösterir ki çakışma direnci belirli hash çıktılarının ön görüntülerini bulma zorluğunu garanti etmez.

Teorem 3.3: Çakışma dirençli herhangi f sıkıştırma fonksiyonu keyfi uzunluklu girdiler alınarak çakışma dirençli bir h hash fonksiyonuna genişletilebilir.

Algoritma 3.1 kullanılarak bu işlem hızlı ve verimli bir şekilde yapılabilir. Algoritma, böyle bir hash fonksiyonu bulmayı bir sıkıştırma fonksiyonu bulmaya indirgemiştir.

3.4. Tekrarlı Hash Fonksiyonlarının Genel Yapıları

Birçok anahtarsız hash fonksiyonu tekrarlı bir sürece göre düzenlenir. Öyle ki keyfi uzunluktaki girdiler, girdinin sabit uzunlukta bloklarının başarılı işlemlere tabi tutulmasıyla özetlenirler. Bir tekrarlı hash fonksiyonu, *t+n bitlik* bir girdiyi *n-bitlik* bir çıktıya dönüştüren bir *f* sıkıştırma fonksiyonu kullanarak oluşturulur.

Keyfi sonlu uzunluktaki bir *x* hash girdisi, sabit uzunluklu *r-bitlik* x_i bloklarına bölünür. Bu işlem genellikle *r* blok uzunluğunun bir çarpımı (katı) olan bit uzunluğu elde etmek için *padding* adını verdiğimiz ekstra bit ekleme gerektirip, doldurulmamış girdinin bit uzunluğunu gösteren bir blok içerir. Bundan sonra her bir x_i bloğu dâhili sabit uzunluklu *f* hash fonksiyonuna bir girdi görevi görür. Bazı sabit *n-ler* için *n bit* uzunluğunun yeni bir ara sonucunu hesaplayan *h*'nin sıkıştırma fonksiyonu, önceki *n-bitlik* ara sonucun bir fonksiyonu ve x_j bloğunun sonraki girdisi gibidir.

H_i, *i* evresinden sonraki kısmı sonucu belirtsin. $x = x_1 x_2 ... x_t$ girdisi için tekrarlı bir hash fonksiyonu için genel işlemler şu şekildedir:

$H_0 = IV$ Başlangıç Değeri

$H_i = f(H_{i-1}, x_i)$, $1 <$ i $<$t; $h(x) = g(H_t)$

H_{i-1}, *i-1* ile *i* arasında *n-bitlik* zincir değişkeni görevini görür ve H_0 önceden tanımlanmış başlangıç değeridir. İsteğe bağlı bir *g* çıktı dönüşümü, *n-bitlik* zincir değerlerini *m-bitlik* $g(H_t)$ sonucuna dönüştürmek için kullanılır; *g* genellikle $g(H_t) = H_t$ birim dönüşümüdür.

Algoritma 3.1. Özetlemede (hashing) Merkle'in Meta Metodu

GİRDİ: Çakışma dirençli *f* sıkıştırma fonksiyonu

ÇIKTI: Anahtarsız çakışma dirençli *h* hash fonksiyonu

1. Kabul edelim ki *f*, *(n + r)-bitlik* girdiyi *n-bitlik* çıktıya dönüştürsün. (Kesinlik için *n = 128* ve *r = 512* düşünebiliriz.) *n-bitlik* hash değerlerini üreten bir *h* hash fonksiyonu *f*'den aşağıdaki gibi oluşturulur.
2. *b* uzunluğundaki bir *x* girdisini her biri *r-bit* uzunluklu $x_1 x_2 ... x_t$ bloklarına bölelim. Eğer gerekirse son x_t bloğu 0-bit ile doldurulur.
3. Ekstra x_{t+1} final bloğu, parça blok, tanımlayalım. Doğruluğu kanıtlanmış *b*'nin ikili r temsilini elde etmek için $b < 2^r$ kabul edelim.

4. 0'ın j bitlik şeridini 0^j ile gösterelim. x' in n-*bitlik* hash değeri olan $h(x) = H_{t+1} = f(H_t || x_{t+1})$ sonucu $H_0 = 0^n$; $H_i = f(H_{i-1} || x_t)$, $1 < i < t+1$'den hesaplanır.

Algoritma 3.2: MD Güçlendirmesi (Merkle-Damgard Güçlendirmesi)

b-bit uzunluğundaki $x=x_1x_2...x_t$ mesajı özetlenmeden önce (x_i'ler konuyla ilgili sıkıştırma fonksiyonuna uygun r bit uzunluğunda bloklar olmak üzere) x_{t+1}, parça bloğu eklenir. b için $b < 2^r$ olduğu kabul edilir.

Teorem 3.4: (Cascading Hash Fonksiyonları)

Eğer h_1 ve h_2'den ikisinden biri çakışma dirençli bir hash fonksiyonu ise bu takdirde $h(x) = h_1(x) \,||\, h_2(x)$ de çakışma dirençli bir hash fonksiyonudur.

Not 3.1: Kesin bit şeritlerine bağlı hash değerleri gibi, ASCII (Tablo 3.4), EBCDIC gibi farklı veri temsilleri hash değerleri hesaplanmadan önce ortak bir formata dönüştürülmelidir.

3.5. Hash Fonksiyonlarında Biçimlendirme ve Başlangıç Bileşenleri

Doldurma (Padding) ve Parça Bloklar: Blok özetleme (hashing) metotları için bir hash girdi şeridine onu blok boyutunun katı yapacak bir miktar bit genellikle özetlemeden önce eklenir.

Algoritma 3.3. Doldurma Metodu I

GİRDİ: Aşamaları işlemek için x verisi ve veri girdisinin blok boyutu n-*bit* uzunluğu verilsin.
ÇIKTI: Bit uzunluğu n'nin bir katı olan doldurulmuş x' verisi.
Bit uzunluğu n'nin bir katı olan bir x' şeridi elde etmek için gereken birkaç tane (belki hiç) 0 bit x verisine eklenir.

Algoritma 3.4. Doldurma Metodu II

GİRDİ: Aşamaları işlemek için x verisi ve veri girdisinin blok boyutu *n-bit* uzunluğu verilsin.
ÇIKTI: Bit uzunluğu n'nin bir katı olan doldurulmuş x' verisi.
x'e bir tek *1-bit* eklenir.
Bit uzunluğu n'nin bir katı olan bir x' şeridi elde etmek için gereken birkaç tane (belki hiç) 0 bit x verisine eklenir.

I. doldurma metodu belirsizdir. Orijinal verinin *0-bitleri* doldurma sırasında eklenmiş olanlardan ayırt edilemeyebilir. Böyle metotlar, verinin uzunluğu alıcı tarafından biliniyorsa kabul edilebilir. II. doldurma metodu belirsiz değildir. Her bir doldurulmuş x' mesajı, doldurulmamış bir tek x mesajına karşılık gelir.

Başlangıç Değerleri (*IVs* - initial values):

İlk değerin *(IV)* sabit veya veri girdisinin bir fonksiyonu olup olmayacağı hash fonksiyonunun hesaplamasında rasgele seçilir. Farklı dosyalar aynı anahtar kullanılarak şifrelendiği zaman başlangıç vektörü çok önemlidir. Bir hash değeri oluşturmada ve doğruluğunun kanıtlanmasında aynı *IV* kullanılmalıdır. Eğer kanıtlayıcı tarafından öncelikli olarak bilinmiyorsa mesajla birlikte aktarılmalıdır. Sonraki durumlarda bu garantili olarak bir bütünlük içinde yapılmalıdır. (Düşmanların elindeki bağımsızlık seviyelerini azaltmak için hash fonksiyon prensiplerine uygun olarak olası *IV*'nin küçük veya sabit bir cümlesi tanımlanmalıdır.)

3.6. Anahtarsız Hash Fonksiyonları (MDCs)

Genel özelliklerinden hareketle belirli hash fonksiyonlarının yapıları oluşturulmuştur. Bu başlık altında MDC olarak bilinen anahtarsız hash fonksiyonlarının bir alt sınıfını işlenmiştir.

Yapısal bir bakış açısıyla bu kodlar kendi dâhili sıkıştırma fonksiyonlarından oluşan işlemlerin doğasına dayandırılarak kategorize edilebilirler. Bu yönden tekrarlı hash fonksiyonlarının üç geniş kategorisi; blok şifrelere dayanan hash fonksiyonları,

alışılmış hash fonksiyonları ve modüler aritmetiğe dayalı hash fonksiyonları olarak ele alınırlar.

3.6.1. Blok Şifrelere Dayanan Hash Fonksiyonları

Temel fikir var olan bir blok şifre sistemini kullanmak ve performansı önemli ölçüde etkilemeden yeni özetleme özellikleri eklemektir.

Bir *(n,r) blok şifresi; r-bitlik* anahtar kullanarak *n-bitlik* açık metinden *n-bitlik* şifre metnine, tersine çevrilebilen bir fonksiyon tanımlar. Eğer *E* böyle bir şifre ise bu takdirde $E_k(x),x$'in *k* anahtarı altında şifrelenmesini belirtir.

n-bitlik blok şifrelerden oluşan hash fonksiyonlarının tartışması tek uzunluklu (*n-bit*) ve çift uzunluklu (*2n-bit*) hash değeri üretenler olmak üzere aralarında bölünürler. Burada *'tek'* ve *'çift'* blok şifre çıktılarının boyutları ile ilgilidir. 2^{64} işlemin olanaksız olduğu kabul edilirse tek uzunluklu hash fonksiyonlarının amacı yaklaşık $n = 64$ blok uzunluğunun şifreleri için bir TYHF sağlamaktır. Ya da $n = 128$ blok uzunluğunun şifrelenmesi için bir ÇDHF sağlamaktır. Çoğu çift uzunluklu hash fonksiyonları için *n-bitlik* blok şifreler yaklaşık $n = 64$ boyutunda mevcuttur. Bu boyutun tek uzunluklu hash kodları çakışma dirençli değildir. Böyle blok şifreler için amaç *2n-bit* uzunluklu ÇDHF'larının hash kodlarını elde etmektir. Bu hash fonksiyonlarında kullanılan anahtarın boyutu yaklaşık olarak şifrenin blok uzunluğu ile aynıdır. (Yani *n-bit*). Diğer taraftan hash fonksiyonları büyük anahtarlar kullanır. (Yani *2n-bit*). Böyle hash fonksiyonlarında dikkat edilecek diğer özellik şifrenin blok uzunluğuna denk bir hash çıktısı üretmek için gerekli blok şifre işlemlerinin sayısıdır.

Bu açıklama altında hash fonksiyonları sınıflandırılıp Matyas Meyer-Oesas, Davies Meyer, Miyagucci Preneel algoritmalarından ve MD4, MD5, SHA1, RIPEMD gibi algoritmaları içine alan mesaj özet ailelerinden bahsedilmiştir.

Algoritma 3.5. Matyas Meyer-Oesas Hash

GİRDİ: *x* bit dizisi

ÇIKTI: *x* in *n-bitlik* hash kodu

1. *x* girdisi *n-bitlik* bloklara bölünür ve eğer gerekli ise son bloğu tamamlamak için doldurulur. *t* tane *n-bitlik* bloktan oluşan doldurulmuş mesajı $x_1x_2x_3...x_t$ ile gösterelim. Sabit *n-bitlik IV* başlangıç değeri önceden belirtilmelidir.
2. H_t çıktısı şöyle tanımlanabilir:

   $H_0 = IV; H_i = E_{g(H_{i-1})}(X_i) \oplus X_i, 1 \leq i \leq t$

Burada *g*, *n-bitlik* girdileri *E* için uygun *K* anahtarına dönüştürür. Eğer *E*, *n-bit* uzunluğunda ise *g* birim fonksiyon olur.

Algoritma 3.6. Davies Meyer Hash

GİRDİ: *x* bit dizisi

ÇIKTI: *x*'in *n-bitlik* hash kodu

1. *x* girdisi *k-bitlik* bloklara bölünür. Burada *k* anahtar boyutudur. Ve eğer gerekirse son blok doldurulur. *t* tane *k-bitlik* bloktan oluşan doldurulmuş mesajı $x_1x_2x_3...x_t$ ile gösterelim. Sabit bir *n-bitlik* başlangıç değeri önceden belirtilmelidir.
2. H_t çıktısı aşağıdaki gibi tanımlanabilir:

   $H_0 = IV; H_i = E_{X_i}(H_{i-1}) \oplus H_{i-1}, 1 \leq i \leq t$

Algoritma 3.7. Miyaguchi-Preneel Hash

Algoritmanın tasarısı Matyas Meyer-Oesas Algoritmasına özdeştir. Çıktıdaki H_{i-1} hariç,önceki algoritmada olduğu gibi toplanır. Daha net bir ifadeyle H_i aşağıdaki gibi tanımlanır:

$H_0 = IV;\ H_i = E_{g(H_{i-1})}(X_i) \oplus X_i\, H_{i-1},\ 1 \leq i \leq t$

Burada Davies-Meyer Hash, Matyas Meyer-Oesas Hash'in dalı olarak değerlendirilebilir. Bu durumda H_{i-1} ile x_t zıt roller oynarlar. DES, Davies-Meyer algoritmasında blok şifre gibi kullanıldığında girdi, *56-bitlik* bloklarda işlemden geçirilir, hâlbuki Matyas-Meyer-Oesas ve Miyaguchi-Preneel *64-bitlik* blokları işleme tabi tutar.

3.6.2. Alışılmış Hash Fonksiyonları

Alışılmış hash fonksiyonları özetlemenin açık amaçları için özel olarak düzenlenmiştir. Bu fonksiyonlar mesaj özet ailesinden yararlanılarak oluşturulurlar. Uygulamada çok iyi ilgi alan bu fonksiyonlar MD4 hash fonksiyonuna dayanırlar. MD4, *32-bitlik* makinelerin bilgisayar yazılım uygulamalarına özgü olarak tasarlanmıştır. Tablo 3.1 bu hash fonksiyonları için gerekli parametreleri tanımlar. "Döngü x Her Biri İçin Döngü Adımları" girdi blokları üzerinde uygun sıkıştırma fonksiyonu içinde yapılan uygulamaları belirtir. Döngü tekrarlanan işlem adımlarıdır. Mesaj özet ailesini oluşturan fonksiyonlar ve özellikleri Tablo 3.1'de verilmiştir.

Tablo 3.1. MD4'e Dayalı Seçilmiş Hash Fonksiyonlarının Özeti

İsim	Bit Uzunluğu	Döngü ve Döngü Başına Adım
MD4	128	3 x16
MD5	128	4 x 16
RIPEMD-128	128	4 x 16
SHA1	160	4 x 20
RIPEMD-160	160	5 x 16

MD (Mesaj Özeti) Ailesi

A. MD2

1989 yılında tanımı verilen mesaj özet algoritması *128-bitlik* blok boyutuna sahiptir. Algoritma kırılmış olup, yeni uygulamalar için kullanılmamaktadır. Mesaj özeti *128-bit* uzunluğunda oluşturulmuştur (Kaliski B., 1992).

B. MD4

1990 yılında Ron Rivest tarafından geliştirilmiştir. MD4, *128-bitlik* bir hash değeri üretir. Mesaj *512-bitlik* bloklarda tekrarlı yapı içinde ve her blok üç farklı döngüde işlemden geçirilir. Aynı hash değerine sahip iki mesaj bulmak yaklaşık 2^{64}

uygulama gerektirirken; önceden belirlenmiş bir hash değerini sağlayan bir mesaj bulmak yaklaşık 2^{128} uygulama gerektirir. MD4 için 2^{20} sıkıştırma fonksiyonu uygulamasında çakışmalar bulunmuştur. Bu sebeple MD4'ün çakışma dirençli hash fonksiyonu olarak kullanımı tavsiye edilmemiştir (Rivest R.L., 1992).

C. MD5

Açılımı "Message Digest Algorithm 5" olan MD5, 128 bitlik özüt değeri üreten, kısmen güvensiz, pek çok güvenlik uygulamasında kullanılan, Ron Rivest tarafından 1991'de tasarlanan, genelde on altılık sayıların sıkıştırılmasıyla oluşturulmuş 32 karakterlik katarlar üreten ve dosya bütünlüğünü test etmede de kullanılan özüt algoritmasıdır. MD5, MD4'ün geliştirilmiş halidir ve her ne kadar MD4'ten çok karmaşık da olsa sonuç itibariyle 128 bitlik özet üretmektedir (RivestR.L., 1992).

İlk değer atamalarından sonra 512 bitlik giriş metni MD5 tarafından 16 adet 32 bitlik alt bloklara ayrılır. Algoritma, çıkış olarak 4 tane 32 bitlik blok üretir.

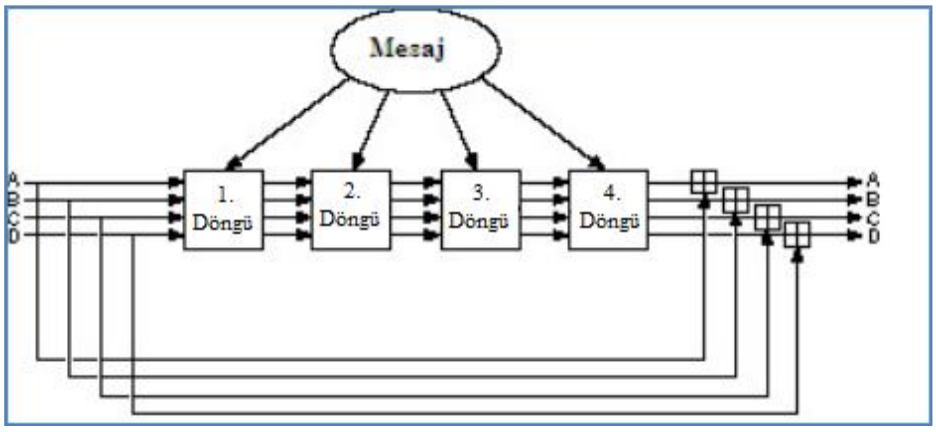
Giriş metninin sonuna 1 bitlik "1" ve gerektiği kadar sıfır eklenerek metnin uzunluğunun 512'nin katı olması sağlanır. 32 bitlik 4 değişkenin ilk değerleri şöyledir: (Rivest R.L., 1992)

A = 0 x 01 23 45 67 B = 0 x 89 ab cd ef

C = 0 x fe dc ba 98 D = 0 x 76 54 32 10

A, B, C ve D'ye *zincir değişkenleri* denir.

Algoritmadaki ana fonksiyon bu noktadan itibaren çalışmaya başlar ve mesajı 512 bitlik bloklar halinde işler. Yukarıdaki 4 değişken farklı 4 değişkene (a = A, b = B, c = C ve d = D olsun) kopyalanır. Ana fonksiyonda 4 döngü (MD4'te sadece 3 döngü vardı) vardır. Daha sonra bu döngülerdeki 16'şar işlemde a, b, c ve d değişkenlerinden üçüne ilkel bir fonksiyon uygulanır. Bu sonuç 4. değişkene eklenip oluşan yeni sonuç bit sayısı kadar sağa döndürülür. Sonucun bu hali a, b, c ve d'den birine eklenir. Son olarak da sonuç a, b, c ve d den birine yazılır (Schneier B., 1996).



Şekil 3.3. MD5 Ana Döngüsü

MD5'teki ilkel fonksiyonlar Tablo3.2'de verilmiştir.

Tablo 3.2. MD5'de Kullanılan İlkel Fonksiyonlar

Tur	İlkel Fonksiyon	Adım(64)
1	$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$	$0 \leq j \leq 15$
2	$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$	$15 \leq j \leq 31$
3	$H(X,Y,Z) = X \oplus Y \oplus Z$	$32 \leq j \leq 47$
4	$F(X,Y,Z) = Y \oplus (X \vee \neg Z)$	$48 \leq j \leq 63$

($Kullanılan\ işaretlerin\ anlamları$: $\wedge = ve$, $\vee = veya$, $\oplus = xor$, $\neg =$ değil)

2007 yılında aralarında Arjen Lastra'nın da bulunduğu bir grup kriptolog tarafından, MD5 özeti aynı olan dosyaların nasıl oluşturulacağına dair bir bildiri (Stevens M., 2006) yayınlanmıştır. Öte yandan 128 bitlik bir özüt değeri bir doğum günü saldırısı düzenleyebilmek için yeterince kısadır. Bunlara ek olarak 2004 Mart'ında başlatılan MD5CRK (Wang X., Feng D., Lai X. and Yu H., 2004) projesi de MD5'in güvenli olmadığını ispatlamayı hedeflemiştir. Bu grup 17 Ağustos 2004'te bir IBM p690 bilgisayarda MD5'e düzenledikleri analitik saldırının sadece 1 saatlik bir zaman diliminde başarıya ulaştığını duyurmuştur. 10 Mart 2006'da ise Vlastimil Klima tarafından bir dizüstü bilgisayarda 1 dakika içinde MD5 çakışmaları üretebilen tünel isimli algoritması yayınlanmıştır.MD5'in bu kadar savunmasız olmasının temelinde yatan neden MD5 algoritmasının özüt için aldığı giriş verisinin üzerinden yalnızca 1 kez

geçmesi olarak tespit edilmiştir. Tüm bunlara karşın MD5 algoritmasının salt bir değerle kullanılması bütün saldırı algoritmalarını son derece işe yaramaz hale getirdiği bildirilmiştir. (Klima V., 2006).

D. SHA Güvenli Hash Algoritması (Secure Hash Algorithm)

Bir mesaj özet algoritması ve aynı zamanda bir kriptografik hash fonksiyonu olan SHA, NSA tarafından düzenlenip, NIST tarafından yayımlanmıştır. Algoritmanın orijinal tanımlaması Güvenli Hash Standardı olarak 1993'te yayımlanmıştır. Bu standart SHA1, SHA256, SHA384 ve SHA512 olmak üzere dört güvenli çırpma (hash) algoritması belirtir. Algoritmaların hepsi de tekrarlıdır. TYHF'ları *"mesaj özeti"* olarak adlandırılan özetlenmiş bir temsilci üretmek için mesajı bazı işlemlerden geçirir. Her algoritma iki evrede tanımlanabilir: Ön işlemler ve hash hesaplaması. Ön işlemler mesajı doldurma, doldurulmuş mesajı *m* bitlik bloklara bölme ve hash hesaplamasında kullanılan başlangıç değerini saptama gibi adımları içerir. Hash hesaplaması doldurulmuş mesajdan bir mesaj listesi, cetveli üretir. Bu listeyi fonksiyonlar, sabitler ve kelime işlemleri ile beraber hash değerlerinin bir serisini üretmek için kullanır. Hash hesaplaması tarafından üretilen son hash değeri mesaj özetini belirler. Bu dört algoritma özetleme sırasında kullanılan verinin kelimeleri ve blokların boyutuna göre farklılık gösterir. Tablo 3.3 ile dört güvenli hash algoritmasının temel özellikleri verilmiştir.

Mesajın uzunluğu, mesaj içindeki bitlerin sayısıdır. Boş mesaj 0-bit uzunluğundadır. Eğer mesajın içindeki bitlerin sayısı 8'in katı ise, sadelik için mesajı onaltılık olarak ifade edebiliriz. Onaltılık bir rakam (hex digit), {0,1, ...,9,a,b,...,f} kümesinin bir elemanıdır. Bir onaltılık rakam *4-bitlik* bir diziyi ifade eder. 0=0000, 1=0001, 2=0010, 3=0011, 4=0100, 5=0101, 6=0110, 7=0111, 8=1000, 9=1001, a=1010, b=1011, c=1100, d= 1101, e=1110, f= 1111'dir. Mesela ondalık 79 sayısı ikilik tabanda 0100 1111 olup, onaltılık tabanda 4f'ye (Tablo 3.4) karşılık gelir. Bir kelime 8 onaltılık rakam dizisi şeklinde ifade edilir. Böylece bir kelime 8 x 4 = 32 *bit* olur. Gerekirse mesajın toplam uzunluğunu 512'nin katı yapmak için mesaj doldurma işlemi yapılır. 512 x *N* uzunlukta doldurulmuş bir mesajın üretilmesi için mesajın sonuna sırasıyla bir adet "1", *m* adet "0" ve *64-bitlik* bir tamsayı eklenir. *64-bitlik* bir tamsayı orijinal mesajın uzunluğudur. Bundan sonra mesaj SHA tarafından *512-bitlik N* blok olarak işlenecektir.

Bir tamsayı, bir kelime veya kelime çifti olarak gösterilebilir. 0 ile 2^{32} - 1 (dâhil) arasındaki bir tamsayı *32-bitlik* bir kelime gibi temsil edilebilir. Mesela 291 sayısı onaltılık tabanda *00000123* ile ifade edilir. Aynı şekilde 0 ile 2^{64} - 1 (dâhil) arasındaki bir tamsayı ise *64-bitlik* bir kelime gibi temsil edilebilir. Eğer Z, $0 < Z < 2^{64}$ olan bir tamsayı ise bu takdirde $Z = 2^{32}.X + Y$; $0 < X < 2^{32}$ ve $0 < Y < 2^{32}$ olur ki Z tamsayısı (x, y) kelime çifti ile temsil edilir. Bu özellik SHA1 ve SHA256 için kullanılır. Eğer Z; $0<Z<2^{128}$ olan bir tamsayı ise bu takdirde $Z = 2^{64}.X + Y$; $0 < X < 2^{64}$ ve $0 < Y < 2^{64}$ olur ki Z tamsayısı (x, y) kelime çifti ile temsil edilir. Bu özellik SHA384 ve SHA512 için kullanılır.

SHA1, güvenli hash algoritması MD4'e dayanır. NIST tarafından US federal hükümet uygulamaları için geliştirilmiştir. SHA1, DSA (sayısal imza algoritması) ile birlikte elektronik postalar, elektronik fon transferi, veri saklaması ve veri bütünlüğü garantisi ve veri kaynağı doğrulaması gerektiren tüm uygulamalarda kullanılabilir. Amacı, güvenli iletişimde mesajlara, iletilen tüm veriyi kullanarak bir mesaj özeti yaratmaktır. Varış noktasında mesaj açıldığında tekrardan hesaplanacak özet ile karşılaştırılacak mesajın, mesaj alıcıya ulaşana kadar değiştirilip değiştirilmediğini kontrol edebilecek bir güvenlik sistemi oluşturulur.

Tablo 3.3. Güvenli Hash Algoritmalarının Özellikleri

Algoritma (bit)	Mesaj Boyutu (bit)	Blok Boyutu (bit)	Kelime Boyutu (bit)	Özet Boyutu (bit)
SHA1	$<2^{64}$	512	32	160
SHA256	$<2^{64}$	512	32	256
SHA384	$<2^{128}$	1024	64	384
SHA512	$<2^{128}$	1024	64	512

Tablo3.3'ten de anlaşıldığı gibi SHA1 ve SHA256 için her mesaj bloğu 512 *bit*'e sahiptir. Yani on altı tane *32-bitlik* kelimenin bir dizisi gibi ifade edilirken; SHA384 ve SHA512 için her mesaj bloğu 1024 *bite* sahiptir. On altı tane *64-bitlik* kelimenin bir dizisi gibi ifade edilir. Son yıllarda*160-bitlik* çıktı üreten SHA1 algoritmasına yönelik saldırılar sonuç vermiş, çakışma bulmak için tahmin edilen 2^{80} işlemden daha az olan 2^{51} (Manuel S., 2008) işlemle çakışmalar bulunmuştur.

SHA1 algoritması tarafından kullanılan bazı işlem, fonksiyon ve sabitler aşağıda verilmiştir.

ROTL^n(x) dairesel sola kaydırma işlemi; *x*, *32-bitlik* bir kelime ve *n*, $0 \leq n \leq 32$ olacak şekildeki bir tamsayı olmak üzere; *ROTL*^{n}(*x*) = *(x << n)* v *(x >> 32 - n)* şeklinde tanımlanır. Bu işlem sadece SHA1 algoritmasında kullanılır. $S^n(x)$ şeklinde de gösterilebilir. SHA1, 4 döngüde gerçekleşir, her döngü 20 adımdır.16 kelimelik bloklar 80 kelimelik bloklara genişletilir. $f_0 f_1,...,f_{79}$ mantıksal fonksiyon dizisini kullanır. Her bir f_t fonksiyonu $0 \leq t \leq 79$ olmak üzere üç tane *x, y, z 32-bitlik* kelimeleri üzerinde işlem yapar ve çıktı olarak da *32-bitlik* kelimeler üretir. f_t (*x*, *y*, *z*) fonksiyonu aşağıdaki gibi tanımlanır.

$$f_t(x,y,z) = \begin{cases} (x \wedge y) \oplus (\neg x \wedge z) & , \ 0 \leq t \leq 19 \\ x \oplus y \oplus z & , 20 \leq t \leq 39 \\ (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & , 40 \leq t \leq 59 \\ x \oplus y \oplus z & , 60 \leq t \leq 79 \end{cases}$$ 'dır.

SHA1 32 - *bitlik* K_0, K_1, ..., K_{79} olmak üzere 80 sabit kelimenin bir dizisini kullanır.

Bu sabitler:

$$K_t(x,y,z) = \begin{cases} 5a827999 & , \ 0 \leq t \leq 19 \\ 6ed9ebal & , 20 \leq t \leq 39 \\ 8f1bbcdc & , 40 \leq t \leq 59 \\ ca62cld6 & , 60 \leq t \leq 79 \end{cases}$$

şeklinde belirlenmiştir.

SHA için hash hesaplaması başlamadan önce $H^{(0)}$ başlangıç hash değeri belirlenmelidir. $H^{(0)}$'daki kelimelerin sayısı ve boyutu mesaj özetinin boyutuna bağlıdır. SHA1 için başlangıç değerleri aşağıdaki gibi *32-bitlik* beş kelimeden onaltılık tabanda

$H_0^{(0)}$=67452301

$H_0^{(1)}$=efcdab89

$H_0^{(2)}$=98badcfe

$H_0^{(3)}$=10325476

$H_0^{(4)}$=c3d2e1f0

şeklinde oluşturulur.

Bir *M* mesajının hash hesaplaması yapılmadan eğer gerekiyorsa algoritmaya bağlı olarak mesajı 512 veya 1024 bitin bir katı yapmak doldurma (padding) yapılır. Doldurulan mesaj *512-bitlik N* bloğa ayrılır; ($M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$). Ön işlemler tamamlandıktan sonra aşağıdaki adımlar kullanılarak sırasıyla her bir mesaj bloğuna işlemler uygulanır. Toplama işlemi (+), 2^{32} modülüne göre yapılır.

i = 1 den *N*'e kadar:

1. $\{W_t\}$ mesaj cetveli hazırlanır;

$$W_t = \begin{cases} M_t^{(i)} & , 0 \le t \le 15 \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & , 16 \le t \le 79 \end{cases}$$ olmak üzere;

2. Beş çalışma değeri a, *b, c, d* ve *e*, (i-1) inci hash değeri ile başlatılır.

$$a = H_0^{(i)}$$
$$b = H_1^{(i)}$$
$$c = H_2^{(i)}$$
$$d = H_3^{(i)}$$
$$e = H_4^{(i)}$$

3. *t* = 0'dan 79'a kadar:

$$T = ROTL^1(a) + f_t(b,c,d) + e + K_t + W_t$$
$$e = d$$
$$d = c$$
$$c = ROTL^{(30)}(b)$$
$$b = a$$
$$a = T$$

4. *i*-inci $H^{(i)}$ ara hash değerleri hesaplanır.

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$

$$H_4^{(i)} = e + H_4^{(i-1)}$$

$M^{(N)}$ işlendikten sonra 160-bitlik mesaj özeti $H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)}$ ile gösterilir. "||" ifadesi değerlerin birleştirilmesini ifade etmektedir.

Tablo 3.4. ASCII Tablosu

000	NUL	033	!	066	B	099	c	132	ä	165	Ñ	198	ã	231	þ
001	Start Of Header (SOH)	034	"	067	C	100	d	133	à	166	ª	199	Ã	232	Þ
002	Start Of Text (STX)	035	#	068	D	101	e	134	å	167	º	200	╚	233	Ú
003	End Of Text (ETX)	036	$	069	E	102	f	135	ç	168	¿	201	╔	234	Û
004	End Of Transmission (EOT)	037	%	070	F	103	g	136	ê	169	®	202	╩	235	Ù
005	Enquiry	038	&	071	G	104	h	137	ë	170	¬	203	╦	236	ý
006	Acknowledge (ACK)	039		072	H	105	i	138	è	171	½	204	╠	237	Ý
007	Bell	040	(	073	I	106	j	139	ï	172	¼	205	═	238	¯
008	Backspace (BS)	041	)	074	J	107	k	140	î	173	¡	206	╬	239	´
009	Horizontal Tab	042	*	075	K	108	l	141	ì	174	«	207	¤	240	-
010	Line Feed (LF)	043	+	076	L	109	m	142	Ä	175	»	208	ð	241	±
011	Vertical Tab	044	,	077	M	110	n	143	Å	176	░	209	Ð	242	‗
012	Form Feed (FF)	045	-	078	N	111	o	144	É	177	▒	210	Ê	243	¾
013	Carriage Return (CR)	046	.	079	O	112	p	145	æ	178	▓	211	Ë	244	¶
014	Shift Out	047	/	080	P	113	q	146	Æ	179	│	212	È	245	§
015	Shift In	048	0	081	Q	114	r	147	ô	180	┤	213	ı	246	÷
016	Dataline Escape (DLE)	049	1	082	R	115	s	148	ö	181	Á	214	Í	247	¸
017	DC 1 (XON)	050	2	083	S	116	t	149	ò	182	Â	215	Î	248	°
018	DC 2	051	3	084	T	117	u	150	û	183	À	216	Ï	249	¨
019	DC 3 (XOFF)	052	4	085	U	118	v	151	ù	184	©	217	┘	250	·
020	DC 4	053	5	086	V	119	w	152	ÿ	185	╣	218	┌	251	¹
021	Negative Acknowledge (NAK)	054	6	087	W	120	x	153	Ö	186	║	219	█	252	³
022	Synchronous Idle	055	7	088	X	121	y	154	Ü	187	╗	220	▄	253	²
023	End Of Transmission Block	056	8	089	Y	122	z	155	ø	188	╝	221	¦	254	■
024	Cancel	057	9	090	Z	123	{	156	£	189	¢	222	Ì	255	
025	End Of Medium	058	:	091	[	124	|	157	Ø	190	¥	223	▀		
026	Substitude	059	;	092	\	125	}	158	×	191	┐	224	Ó		
027	Escape (ESC)	060	<	093	]	126	~	159	ƒ	192	└	225	ß		
028	File Seperator	061	=	094	^	127 (DEL)	⌂	160	á	193	┴	226	Ô		
029	Group Seperator	062	>	095	_	128	Ç	161	í	194	┬	227	Ò		
030	Record Seperator	063	?	096	`	129	ü	162	ó	195	├	228	õ		
031	Unit Seperator	064	@	097	a	130	é	163	ú	196	─	229	Õ		
032	SPACE (SP)	065	A	098	b	131	â	164	ñ	197	┼	230	µ		

E. RIPEMD-160

Ripemd-160, MD4'e dayanan bir hash fonksiyonudur. *160-bitlik* mesaj özet algoritması Hans Dobbertin, Antoon Bosselaers ve Bart Preneel tarafından Avrupa'da 1996 yılında geliştirilmiş ve yayımlanmıştır. Geniş kapsamlı Ripemd-160 sıkıştırma fonksiyonu 21 kelimelik girdileri (5 kelimelik zincir değişkeni ve 16 kelimelik mesaj) 5 kelimelik çıktılara dönüştürür. Bu program dosyaları karşılaştırmak ve değiştirilip değiştirilmediğini anlamak için ve çoğunlukla daha büyük özetler gerektiren uygulamalarda kullanılır. Ripemd algoritmasının Ripemd-128, Ripemd-256, Ripemd-320 olarak adlandırılan sırasıyla *128, 256*ve *320-bitlik* sürümleri da vardır (Preneel B., Bosselaers A. and Dobbertin H., 1997).

3.6.3. Modüler Aritmetiğe Dayanan Hash Fonksiyonları

Modüler aritmetiğe dayanan hash fonksiyonlarının ana fikri ModM'yi kullanarak bir sıkıştırma fonksiyonunun esası gibi tekrarlı hash fonksiyonu oluşturmaktır. Mod işlemlerini kullanarak sonucu küçültüp, hash fonksiyonunu biçimlendirmek için kullanılır. MASH-1 (Modular Arithmetic Secure Hash) ve MASH-2 fonksiyonları modüler aritmetiğe dayanan hash algoritmalarıdır. MASH-1 ve MASH-2 anahtarsız kriptografik hash fonksiyonları olup ön görüntü, ikinci ön görüntü ve çakışma direncini sağlarlar.

3.7. Anahtarlı Hash Fonksiyonları (MACs)

Amacı mesajın bütünlüğünü garantilemek, mesajı kanıtlamak olan anahtarlı hash fonksiyonları Message Authentication Code algoritmaları olarak da adlandırılırlar. MDC algoritmalarının sayısının fazlalığına karşın 1995'ten önce ancak birkaç tane MAC algoritması ileri sürülebilmiştir. Çoğu tekrarlı MACs tekrarlı hash fonksiyonları gibi tanımlanabilirler. Bu durumda bir MAC anahtarı genellikle *g* çıktı dönüşümünün

bir kısmı olabileceği gibi, ilk tekrarda sıkıştırma fonksiyonuna bir girdi olabilir ve *f* sıkıştırma fonksiyonuna her evrede dahil olabilir.

Sayısal bilginin kaynak, köken ve içeriğinin reddedilmez bir yolunu ortaya koymak için 1976'da Diffie ve Hellman tarafından ilk kez ortaya çıkarılan sayısal imza tasarılarının düzenlemesinin teknik problemi ve onların yerine getirilmesi iyi anlaşılmıştır fakat bazı uygulamalar için onlar çok fazla depolama ve hesaplama gerektirmektedir. Bu yüzden bazı uygulamalar halen veri bütünlüğü ve veri kaynağı belgelenmesinde MAC algoritmaları kullanır. MACs imzalardan daha zayıf garanti sağlarlar. Bu yüzden tarafların birbirine güvendiği simetrik ortamlarda tercih edilmelidir. Bankalar son yetmiş yıldır mesaj doğrulamada MACs'i kullanmaktadır. Bir MAC kullanmak için gönderen ve alıcının bir gizli *K* anahtarını paylaşması gerekir. (*k* bitlik rasgele bir bit şeridi - tipik değerleri k = 56, ..., 128) Mesajı korumak için, gönderen mesaja uygun *m-bit* uzunluklu bit şeridini MAC'i hesaplar ve bu şeridi mesaja ekler. (Tipik *m* değerleri 32, ..., 64) MAC, mesajın ve anahtarın her bitinin karmaşık bir fonksiyonudur. Mesajın alındısı üzerinde alıcı, MAC'i yeniden hesaplar ve gönderilen MAC değerine karşılık gelip gelmediğinin doğruluğunu kanıtlar.

3.7.1. Blok Şifrelere Dayanan MACs

En yaygın kullanılan MAC algoritması şifre-blok zincirinin (CBC-Cipher Block Chaining) kullanımı sağlayan bir blok şifreye dayanan CBC-MAC'dir. Genellikle blok şifrenin temelini oluşturan DES ile kullanılır. Blok şifre blok uzunluğu n = 64 olan *E* blok şifresi ve MAC anahtarı olarak da *56-bitlik* bir DES anahtarı alınır. CBC-MAC aşağıdaki sıkıştırma fonksiyonu verilen ile tekrarlı bir MAC'dir:

$$H_i = E_K(H_{i-1} \oplus x_i) \ , 1 \leq i \leq t \ .$$

Burada $E_K(x)$; k bitlik K anahtarını *n-bitlik* E blok şifresi ve $H_0 = IV = 0$ ile kullanarak x'in şifrelenmesini belirtir. İlk olarak eğer gerekli ise x doldurulur, doldurulan metin *n-bitlik* $x_1, x_2, \ldots, x_t$ bloklarına bölünür daha sonra CBC işlemleri yapılır. g çıktı dönüşümü olmak üzere MAC, $MAC_K(x) = g(H_t)$ olarak hesaplanır.

Algoritma 3.8: CBC-MAC

GİRDİ: *x* verisi; *E* blok şifrenin belirtilmesi; *E* için gizli *k* MAC anahtarı

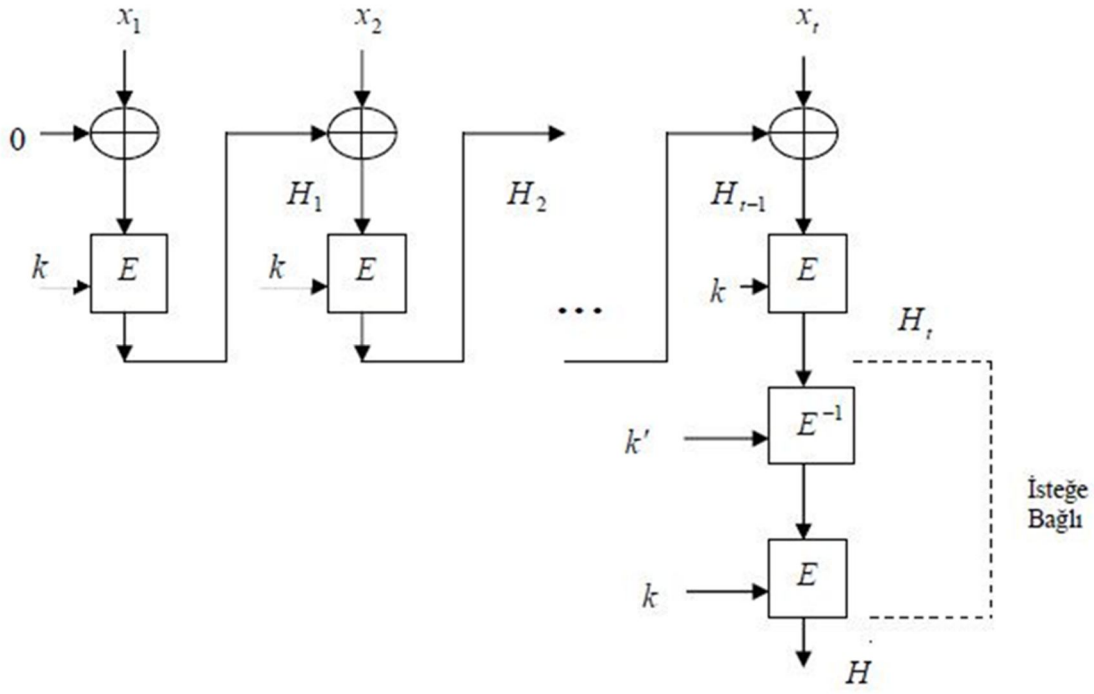
ÇIKTI: *E*'nin blok uzunluğu *n* olmak üzere *x* üzerinde *n-bitlik* MAC

1. Eğer gerekiyorsa *x* doldurulur. Doldurulmuş metni *n-bitlik* $x_{,,}$ x_2, ..., x_t bloklarına bölünür.
2. *k* anahtarı ve *E* kullanılarak E_K şifrelemeyi belirtmek üzere H_t bloğunu aşağıdaki gibi hesaplanır.
   $H_1 \leftarrow E_K(x_1);\ H_i \leftarrow E_K(H_{i-1} \oplus x_i)\ , 2 \leq i \leq t$ (Standart CBC algoritma 2.1, $IV = 0$, şifre metin blokları $C_i = H_i$ atılır)
3. MAC'in gücünü artırmak için isteğe bağlı işlemler kullanılabilir. İkinci bir $k' \neq$ k gizli anahtarı ile $H'_t \leftarrow E_{k'}^{-1}(H_t), H_t \leftarrow E_K(H'_t)$ hesaplanır.
4. MAC, *n-bitlik* H_t bloğudur.

MAC'e dayanan bu şifre-blok-zincirleme algoritması Şekil 3.4'te gösterilmiştir.

3.7.2. MDC'ye Dayanan MAC(MDCs'den MACs' in Oluşturulması)

Kriptografik hash fonksiyonlarına dayanan MAC algoritmaları elde etme çalışmaları birkaç öneri ile sonuçlanmıştır. Bir MDC algoritmasından bir MAC oluşturmak için bilinen bir öneri; gizli bir *k* anahtarını MDC girdisinin bir kısmı olarak dâhil etmektir.



Şekil 3.4. MAC Algoritmasına Dayalı CBC

3.7.3. Hash'e Dayalı MAC

HMAC'in tanımı hash ile belirttiğimiz kriptografik hash fonksiyonu ve gizli bir *K* anahtarı gerektirir. HMAC, SHA1, MD5, RIPE-MD hash fonksiyonlarından birini kullanır. Bir HMAC fonksiyonu gönderen tarafından, bir gizli anahtar ve mesajın özetlenmesi ile MAC değeri üretmek için kullanılır. MAC, mesajla birlikte alıcıya gönderilir.

3.8. Bilinen Bazı Kriptografik Hash Algoritmalarının Özellikleri

Tablo 3.5. Kriptografik Hash Algoritmalarının Özellikleri

(Tabloda gösterilen değerlerin referansları kaynaklar bölümünde verilmiştir.)

	Blok Uzunluğu	Mesaj Özeti	Kelime Boyutu	Collision Atakları	Preimage Atakları
GOST	256	256	32	VAR	VAR
MD4	512	128	32	VAR	VAR
MD5	512	128	32	VAR	VAR
RIPEMD	512	128	32	VAR	-
RIPEMD-128/256	512	128/256	32	YOK	-
RIPEMD-160/320	512	160/320	32	YOK	-
SHA0	512	160	32	VAR	-
SHA1	512	160	32	VAR	YOK
SHA256/224	512	256/224	32	YOK	YOK
SHA512/384	1024	512/384	64	YOK	YOK
TIGER(2)-192/160/128	512	192/160/128	64	VAR	VAR
WHIRLPOOL	512	512	8	VAR	-
HAVAL	1024	256/224/192/160/128	32	VAR	-

3.9. Hash Algoritmalarında Güvenlik ve Saldırı Çeşitleri

Güvenlik derecesi, algoritmadan algoritmaya değişiklik gösterir. Bu, kırılmanın ne kadar zor olduğuna bağlıdır. Bir algoritmayı kırmanın maliyeti şifrelenmiş verinin değerinden fazlaysa veya kırmak için gereken süre şifrelenmiş verinin gizli kalması gereken süreden fazlaysa büyük olasılıkla güvendesiniz demektir. Tek bir anahtar ile şifrelenmiş veri miktarı, algoritmayı kırmak için gereken veri miktarından az ise yine büyük olasılıkla güvendesiniz demektir. "Büyük olasılıkla" çünkü şifre analizinde her zaman yeni gelişmeler, buluşlar söz konusudur. Zaman içinde çoğu verinin değeri düşer. Şifrelemenin güvenliği anahtar uzunluğuna ve kırılan şifreli metnin içerdiği matematiksel işlemlerin karmaşıklığına dayanır.

Yıllardır kriptografik ilkellerin ve protokollerin üzerinde çok farklı saldırı tipleri saptanmıştır:

Bir *pasif saldırı*; düşmanın sadece iletişim kanalını dinlediği saldırıdır. Bir pasif saldırgan sadece verinin bütünlüğünü tehdit eder.

Bir *aktif saldırı*; düşmanın, kanal üzerinde gönderilen mesaja silme, ekleme veya diğer yollarla müdahale ederek değiştirmeye teşebbüs ettiği saldırıdır. Bir aktif saldırgan veri bütünlüğü ve verinin kaynağı (verinin kimliği) ile ilgili bilgileri tehdit eder.

Algoritmayı kırmaya ilişkin kategoriler ise sertlik-güç derecelerine göre şu şekilde sınıflandırılmıştır.

Genel Saldırı Çeşitleri

1. **Şifre Metin Saldırısı:** Kriptanalist, aynı şifreleme algoritmasını kullanılarak şifrelenmiş çeşitli mesajların şifre metinlerine sahiptir. Kriptanalistin görevi mümkün olduğunca çok sayıdaki mesajın açık metnini kurtarmak veya mesajları şifrelemek için kullanılan anahtarları, aynı anahtarla şifrelenen başka mesajları çözmek için anlamaktır.
2. **Açık Metin Saldırısı:** Kriptanalist sadece çeşitli mesajların şifrelenmiş haline değil, bu mesajların açık metinlerine de erişebilir. Bir yada daha fazla $(x,\ldots,h_k(x_t))$MAC metin çifti elde edebilir. Görevi mesajları şifrelemek için kullanılan anahtarları veya aynı anahtarla şifrelenmiş yeni mesajları çözebilecek bir algoritma bulmaktır.
3. **Seçilmiş Açık Metin Saldırısı:** Kriptanalist sadece çeşitli mesajların şifre metinlerine ve bunlarla ilişkilendirilmiş açık metinlere erişmekle kalmayıp, aynı zamanda da şifrelenmiş açık metinleri seçebilmektedir. Saldırgan tarafından seçilen x_i'ler için bir ya da fazla (x_i, h_k (x_i)) MAC metin çifti elde edebilir. Bu bilinen açık metin saldırısından daha güçlü bir saldırıdır, çünkü kriptanalist şifrelemek için açık metnin belirli bloklarını, anahtar hakkında daha fazla bilgi sağlayabilecek olanlarını seçebilmektedir.
4. **Uyarlanabilir Seçilmiş Açık Metin Saldırısı:** Bu seçilmiş açık metin saldırısının özel bir türüdür. Kriptanalist, şifrelenmiş açık metni seçmekle kalmayıp, aynı zamanda seçimini önceki şifrelemeleri esas alarak değiştirebilmektedir. Seçilmiş açık metin saldırısında bir kriptanalist, sadece bir büyük açık metin bloğunu seçebilirken; uyarlanabilir seçilmiş açık metin saldırısında daha küçük bir açık metin bloğu seçebilir ve ilkinin sonuçlarını esas alarak bir başkasını seçebilir.
5. **Seçilmiş Anahtar Saldırısı:** Bu saldırı kriptanalistin anahtarı seçebileceğini ifade etmez; anlamı değişik anahtarlar arasındaki ilişki hakkında biraz bilgisi olduğudur.

6. **Rubber-Hose Saldırısı:** Kriptanalist bir kişiye anahtar kendisine verilene kadar gözdağı verir, şantaj yapar. Bunlar çok güçlü saldırılardır ve bazen anahtarı kırmanın en iyi yoludur.
7. **Kaba Güç (Brute Force) Saldırısı:** Bir şifre metninin okunaklı mesaj elde edilene kadar birer birer her olası anahtarın denenmesiyle deşifre edildiği saldırılardır. Bir anahtar araştırmasını engellemek için tek yol yeterince büyük anahtarlar seçmektir.
8. **Doğum Günü Saldırısı (Birthday Attack):** Büyük bir popülasyondan iki özdeş değer elde etmek için kullanılan bir saldırı biçimidir. Kriptografik hash fonksiyonlarına uygulanan, çakışma bulmak için kullanılan bir tür kaba güç (brute force) saldırısıdır. İsmini 23 kişilik bir grupta iki veya daha fazla kişinin aynı doğum gününü paylaşma olasılığının 1/2 olması gibi şaşırtıcı sonuçtan alır. Bu olay "doğum günü paradoksu (birthday paradox)" olarak adlandırılır.

N elemanlı bir kümeden rasgele elemanlar seçildiğinde tekrarlayan bir elemanı yüksek olasılıkla $O(\sqrt{N})$ seçimden sonra ulaşılır. Böyle saldırılar kare-kök saldırıları arasında adlandırılırlar. Hash fonksiyonu ile ilgisi bir tek yol hash fonksiyonu için belli bir hash değerinin çakışmasını bulmak ön görüntü veya 2. ön direncini bulmaktan daha kolay olmasından gelir. Sonuç olarak tek yol hash fonksiyonu gerektiren imza tasarıları doğum günü saldırılarına karşı savunmasızdır. Bu saldırı *m bit* uzunluklu hash değerine bağlı olarak uygulama adımlarının sayısı $O(2^{m/2})$ olan bütün anahtarsız hash fonksiyonlarına (Teorem 3.5) uygulanabilir.

Doğum günü paradoksunu aşağıdaki gibi açıklayabiliriz:

Bir *X* rasgele tamsayı değişkeni 1 ile *n* arasında düzenli gruplara ayrılsın. $P(x_i) = 1/n$, $i = 1, ..., n$ ve bu *X* değişkeninin bir *k* örneğinin seçimi için *P(n,k)*'nın en az bir eşi olma olasılığı nedir? Örnek olarak $k = 12$ öğrencili bir sınıf verilsin. İki öğrencinin aynı doğum gününe sahip olma olasılığı nedir? Farz edelim ki $n = 365$ doğum günü bir yıla dağıtılsın. Yani olasılık 1/365 midir?

İfade kolaylığı açısından *"eş olmama"* olasılığı *Q(n,k)* olsun. Verilen bir *k* için

$$N_{eş-olmama} = n \times (n-1) \times \ldots \times (n-k+1) = \frac{n!}{(n-k)!}$$

n elemanlarının cümlesinden alınan *k* elemanları ile *eş olmamanın* elde edilme yoludur. Eğer eşe sahip olmama durumu olmadan *n*'nin dışında *k* seçmenin bütün yollarının sayısını düşünürsek;

$N_{eş-olma} = n \times n \times \ldots \times n = n^k$ ve eşi olmama olasılığı $Q(n,k)$;

$Q(n,k) = \frac{N_{eş-olmama}}{N_{eş-olma}} = \frac{n!}{(n-k)!\, n^k}$ ile verilebilir.

$$P(n,k) = 1 - Q(n,k) dır.$$

$$P(n,k) = 1 - \frac{n!}{(n-k)!\ n^k}$$

$$P(n,k) = 1 - \left[\left(\frac{n-1}{n}\right) \times \left(\frac{n-2}{n}\right) \times \ldots \times \left(\frac{n-k+1}{n}\right)\right]$$

$$P(n,k) = 1 - \left[\left(1-\frac{1}{n}\right) \times \left(1-\frac{2}{n}\right) \times \ldots \times \left(1-\frac{k-1}{n}\right)\right]$$

Her x > 0 *için* (1 - x) < e^x eşitsizliği kullanılırsa

$$P(n,k) \geq 1 - \left[e^{-1/n} \times e^{-2/n} \times \ldots \times e^{-(k-1)/n}\right]$$

$$P(n,k) \geq 1 - e^{-\left[\frac{1}{n}+\frac{2}{n}+\cdots+\frac{k-1}{n}\right]}$$

$$P(n,k) \geq 1 - e^{-\left[\frac{k\times(k-1)}{2n}\right]}.$$

Örnek 3.9.1.

Aynı doğum gününe sahip iki kişi bulma olasılığı %50 olacak şekilde bir odadaki insanların sayısını *k* ile belirtmek istiyoruz. Burada kabul edelim ki *n* = 365 gün olsun. *k* kişinin minimum sayısını *P(n, k)* = 1/2 olacak şekilde belirtmek istiyoruz. Biliyoruz ki $P(n,k) \geq 1 - e^{-\left[\frac{k\times(k-1)}{2n}\right]}$ idi.

$$\Rightarrow \frac{1}{2} = 1 - e^{-\left[\frac{k\times(k-1)}{2n}\right]}$$

$$\Rightarrow -\frac{1}{2} = -e^{-\left[\frac{k\times(k-1)}{2n}\right]}$$

$\Rightarrow \frac{1}{2} = e^{-\left[\frac{k\times(k-1)}{2n}\right]}$ her iki tarafın *e* tabanında logaritmasını alırsak;

$$\ln\left(\frac{1}{2}\right) = \ln\left(e^{-\left[\frac{k\times(k-1)}{2n}\right]}\right)$$

$$\ln 2 = \left[\frac{k \times (k-1)}{2n}\right]$$

$2n \times \ln 2 = k\,(k-1) \leq k^2 \Rightarrow \pm\sqrt{2n \times ln2} = k$'dır.

Buradan n = 365 gün, k minimum kişi sayısı olmak üzere $k \geq \sqrt{2 \text{ x } 365 \text{ x } \ln 2}$ veya $k \geq 22{,}4944$ bulunur. Bir odada k = 23 kişi varsa aynı doğun gününe sahip iki kişi bulma olasılığı %50 den büyüktür.

$$P(n,k) \geq 1 - e^{-\left[\frac{k\times(k-1)}{2n}\right]}$$

$$P(365,23) \geq 1 - e^{-\left[\frac{23\times22}{2\times365}\right]}$$

$$P(365,23) \geq 0{,}500017518271'dir.$$

Başka bir ifadeyle şöyle özetleyebiliriz. İki kişinin aynı doğum gününe sahip olmama olasılığı q,

$$q = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \ldots \times \frac{343}{365} = 0{,}492703 \text{ olur.}$$

Buradan en az iki kişinin aynı doğum gününe sahip olma olasılığı 1 - q = 0.507297'dır. Bu olasılık odadaki kişi sayısına bağlı olarak hızla artar. Örneğin 46 kişinin aynı doğum gününü paylaşma olasılığı 0,948253'tür.

Doğum günü paradoksu, çakışma bulmak için rasgele eşler seçmekten daha hızlı çakışma çiftleri bulmak için hash fonksiyonlarına uygulanabilir.

Örnek 3.9.2.

n = 2^{128} olsun. Hash değerleri birbirine eşit olacak şekilde yani H(x) = $H(y)$ olacak şekilde x, y gibi farklı iki 128-bitlik vektör bulma olasılığını düşünelim. Bu aynı hash değerinde herhangi bir *(x, y)* çiftinin güçlü çakışma problemidir. Eşitliği sağlayacak y vektörlerinin sayısı;

$k = \sqrt{2n \times ln2}$ yani

$k = \sqrt{2 \times 2^{128} \times ln2}$'dır.

$k = 2^{64}\sqrt{2 \times ln2} = 1{,}1774 \times 2^{64}$

$k = \sqrt{2n \times ln2} = 1{,}18 \times \sqrt{n} \cong \sqrt{n}$'dir.

9. Zincir Saldırıları

512-bitlik x_i mesaj bloklarını ile 128 bitlik H_i *(*H_0 = *IV)* zincir değişkenlerini girdi olarak alan ve f sıkıştırma fonksiyonu $H_{i+1} = f\ (H_i, x_i)$ olmak üzere *128-bitlik* bir hash değeri üreten çakışma dirençli tekrarlı bir hash fonksiyonu düşünelim. Sabit 10

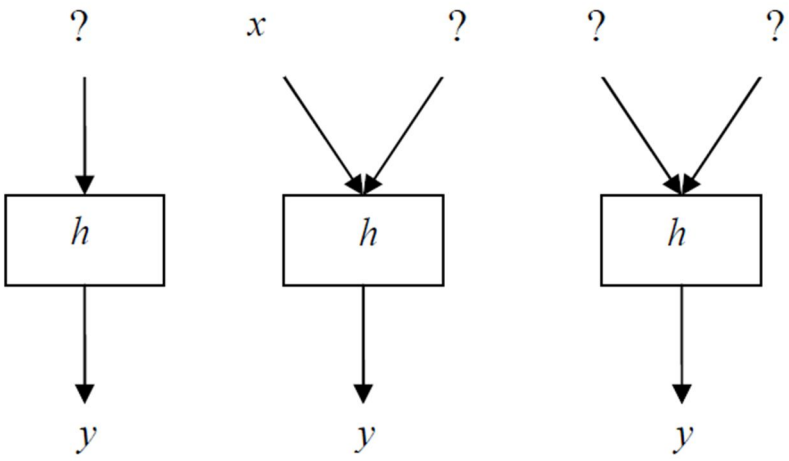
blokluk x mesajı (640 byte) için $H = h(x)$ düşünelim. Kabul edelim ki biri 10 bloktan herhangi birinin seçip yerine H fonksiyonunu etkilemeden yerine başka bir blok koymak istesin. Eğer h rasgele dönüşüm gibi davranırsa *512-bitlik* böyle blokların sayısı yaklaşık $2^{512} / 2^{128} = 2^{384}$'tür. Bu 2^{384} bloktan herhangi birini bulmanın metodu hash üzerine bir saldırı oluşturmaktan farklıdır.

10. Meet-in the-middle Zincir Saldırıları

Bu saldırılar doğum günü saldırılarına benzerler fakat çakışmalar bütün hash değeri üzerinde değil de ara zincir değişkenlerinde aranır.

Bu saldırılar altında MDC'ye saldırmak isteyen bir saldırganın amacı şunlardır:

1. Tek Yol Hash Fonksiyonlarına Saldırı: Bir y hash değeri verildiğinde $y = h(x)$ olacak şekilde bir x ön görüntüsü bulmak veya $(x, h(x))$ ikilisi verildiğinde $h(x) = h(x')$ olacak şekilde ikinci bir ($x \neq x'$ olmak üzere) x' girdisi bulmaktır.
2. Çakışma Dirençli Hash Fonksiyonlarına Saldırı: $h(x) = h(x')$ olacak şekilde herhangi iki x, x' girdileri bulmaktır. Bir çakışma dirençli hash fonksiyonu doğum günü saldırılarına karşı koyacak şekilde düzenlenmelidir.



Şekil 3.5. Saldırganların Amaçları

Hash değerinin önceden verilmediği uygulamalar da vardır. Bunlar saldırgana daha büyük özgürlük sağlar. Örneğin sayısal imzalarda bir d dokümanı $H(d)$ ile imzalanır. $H(d)$ üzerinde herhangi bir kısıtlama olmadığından saldırgan $H(d) = H(d')$ olacak şekilde bir farklı d, d' çifti belirler. $H(d)$'yi kullanarak d'yi imzalar ve orijinal metnin d' olduğunu iddia eder. $H(d) = H(d')$ olduğundan saldırganın iddiası

yalanlanamaz. Bu yüzden hash fonksiyonu çakışma direncine sahip olmalıdır; hash değeri aynı olan farklı bit şeritleri oluşturmak mümkün olmamalıdır.

MAC'e saldırmak isteyen bir saldırganın amacı ise şöyledir:

Önceden k anahtarı ile ilgili bir bilgi olmadan bir yâda daha fazla (x_i, h_k (x_i)) ikilileri verildiğinde bazı $x \neq x_i$ metinleri için $(x, h_k(x))$ ikilisi hesaplamaktır. Burada hesaplama direnci ele alınarak MACs ile eşlenen x_i metinlerinin düşman tarafından seçilen metinlerle elde edilip edilemeyeceğine bakılır. Bilinen, seçilmiş, uyarlanabilir metin saldırısı MACs için mevcuttur. Saldırıların belirli bir ortamda tırmanıp tırmanmadığına bakmaksızın MACs mutlaka seçilmiş metin saldırılarına karşı koymalıdır.

Hash fonksiyonlarının hesaplanabilir güvenliğinin değerlendirilebilmesi için hash fonksiyonunu düzenleyenin ve düşmanın her ikisinin de anlaşılması gerekir. Tablo3.6'da bu özetlenmeye çalışılmıştır.

Tablo 3.6. *t*-bitlik MAC Anahtarlı, n-bitlik Hash Fonksiyonlarının Dizayn Amacı.

TYHF	Ön Görüntü Direnci 2. Ön Görüntü Direnci	2^n 2^n	Ön görüntü üretmek; 2.girdi, aynı görüntü bulmak
ÇDHF	Çakışma Direnci	$2^{n/2}$	Herhangi bir çakışma üretmek
MAC	Anahtar geri alınamaz; hesaplama direnci	2^t $P_f = \max(2^{-t}, 2^{-n})$	MAC anahtarını ortaya çıkarmak; yeni (msj,MAC) üretmek

(P_f bir MAC'i doğru tahmin eden sahtekârlığın olasılığını belirtir.)

Temel saldırılar yapılırken algoritmalarda kullanılan bit boyutları ve algoritmanın anahtar uzayı göz önüne alınır.

3.9.1. Bir MDC'nin Bit Boyutuna Saldırı

n-bitlik h(x) ile sabit bir *x* mesajı verilsin. *x* ile çakışık bir girdi bulmak için basit bir metotla rasgele *x′* mesajı seçmek ve *h(x) = h(x′)* olup olmadığını kontrol etmektir. Hash değerinin aynı rasgele değere yaklaştığı kabul edilirse; eşlenme olasılığı 2^{-n}'dir.

Teorem 3.5.Temel Hash Saldırıları: *n-bitlik* hash fonksiyonu için *2n* hashing (özetleme) işlemi içinde bir ön görüntü veya ikinci ön görüntü bulmak için tahmini bir saldırı beklenebilir. Mesajları seçebilen bir saldırgan için bir doğum günü saldırısı, yaklaşık $2^{n/2}$ işlem içinden $h(x) = h(x')$ olacak şekilde x, x' çakışık mesaj çifti bulunmasını sağlar.

Tanım 3.5. Bir *n-bitlik* anahtarsız hash fonksiyonu *ideal güvenliğe* sahiptir. Eğer

1. Bir hash çıktısı verildiğinde ön görüntü ve ikinci ön görüntünün her birini üretmek yaklaşık olarak 2^nişlem gerektiriyorsa ve
2. Bir çakışma üretmek yaklaşık olarak $2^{n/2}$işlem gerektiriyorsa.

3.9.2. MAC Anahtar Uzayına Saldırı

Bir MAC anahtarı saptamak için eksiksiz, ayrıntılı bir araştırma yoluyla denemeler yapılabilir. Bir saldırgan bir tek bilinen MAC metin çifti ile bu metin üzerinde bütün olası anahtarların yönetiminde *n-bitlik* MAC hesaplayabilir. Sonra bilinen çift ile hesaplanan MAC değerlerinden hangilerinin uyuştuğunu kontrol eder. Bu, *t-bitlik* anahtar uzay için 2^t MAC işlemi gerektirir ve $1+2^{t-n}$ adayın kalması beklenir. Bir MAC anahtarı 2^t den daha az işlemle geri alınamaz. Bir MAC uzayına olası bir saldırı anahtarı geri almadan farklıdır. *t-bitlik* bir anahtar ve sabit bir girdi için rasgele seçilen bir anahtar *n-bitlik* doğru MAC'i $\approx 2^{-t}$, $t < n$ olasılığı ile sağlayacaktır.

3.9.3. Bir MAC'in Bit Boyutuna Saldırı

MAC sahtekârlığı (MAC Forgery), herhangi bir x girdisi ve herhangi birinden elde edilmiş anahtar bilgisine sahip olmadan doğru MAC üretmeyi içerir. *n-bitlik* bir MAC algoritmasına göre verilen bir girdi için bir MAC'i tahmin etme veya verilen bir MAC çıktısı için bir ön görüntüyü tahmin etmenin başarı olasılığı, MDC de olduğu gibi,

yaklaşık olarak 2^{-n} kadardır. Bununla birlikte buradaki bir fark bilinen MAC-metin çifti anahtar bilgisi veya verilen girdiler için MACs'i sağlayan girdi ve çıktı değerlerinden oluşan bir bileşenden (black box) her ikisi olmadan MAC değerlerinin doğruluğu kanıtlanamaz. İdeal olarak bir saldırgan olasılığı $max(2^{-t},2^{-n})$ den daha iyi olan yeni doğru bir (x, y) MAC metin çifti üretememelidir. Yani bir anahtar veya MAC değerini $max(2^{-t},2^{-n})$'den daha iyi tahmin edememelidir.

Ön görüntü ve ikinci ön görüntü saldırılarının her ikisi için hash fonksiyonunun girdi çıktı çiftlerinin büyük bir miktarını önceden hesaplayan bir rakip, sonraki saldırı zamanı için depolama ile ön hesaplamaları değiş tokuş edebilir. Örneğin; Bir 64-bitlik hash değeri için eğer rasgele 2^{40} girdi seçip ayrılırsa, sonra onların hash değerini hesaplar ve (hash değeri, çıktı) çiftlerini depolar. Böylece uzay, bir düşmanın ön görüntü bulma olasılığını 2^{-64} ten 2^{-24}'e çıkarmasını sağlar.

Farz edelim ki bir hash fonksiyonu n bitlik hash değeri üretsin. Temsilci olarak kabul edelim ki 2^{80} işlem hesaplana bilirliğinin dışında uygun olsun. 2^{40} basit işlem tamamıyla olasıdır, 2^{56}'sına da yeterli uygulamalarla tam olarak erişilebilir. Bu takdirde n hakkında şu ifadeler hazırlanabilir:

1. Bir TYHF için, $n \geq 80$ olması gerekir. Etraflı çevrimdışı saldırılar 2^n işlem gerektirir. Bu yukarıda bahsedildiği gibi hesaplamalarla azaltılabilir.
2. Bir ÇDHF için $n > 160$ olması gerekir. Doğum günü saldırıları uygulanabilir.
3. Bir MAC için *64-80 bitlik* bir MAC anahtarı ile birlikte $n > 64$ olması birçok uygulama için yeterlidir. Eğer tek bir MAC anahtarı kullanımda kalıyorsa saldırılar bir veya daha fazla MAC metin çiftine uygulanabilir. Fakat düzgün bir MAC algoritması için ön görüntü ve 2. ön görüntü direnci anahtarın bilgi eksikliğinden takip edilmelidir. Şu halde güvenlik n'den ziyade anahtarın boyutuna bağlı olmalıdır.

Bir blok şifrenin amacı güvenliği sağlamak; bir saldırganın amacı ise şifre metinden çeşitli kod anahtarları yardımıyla açık metni oluşturmaktır. Bir blok şifreye, eğer bir anahtar bulunur ise *tamamen kırılabilirdir* ve eğer saldırgan, anahtar olmaksızın şifre metinden açık metnin bir kısmını elde edebilirse *kısmen kırılabilirdir* denir.

4. HASH FONKSİYONLARININ UYGULAMA ALANLARI

Kriptografik hash fonksiyonları birçok alanda kullanılmakla beraber en bilinen belli başlı alanları şu şekildedir:

1. Sayısal İmza (Digital Signature)
2. Dosya Bütünlüğü (File Integrity)
3. Şifre Koruması (Password Protection)
4. Kimlik Doğrulama Protokolleri (Authentication Protocols)
5. Rootkit Tespiti (Rootkit Detection)vs…

Bu bölümde yukarıdaki alanlardan dosya bütünlüğü ve şifre koruması alanlarında yaptığımız uygulamalardan kısaca bahsedilmiştir.

4.1. Dosya Bütünlüğü

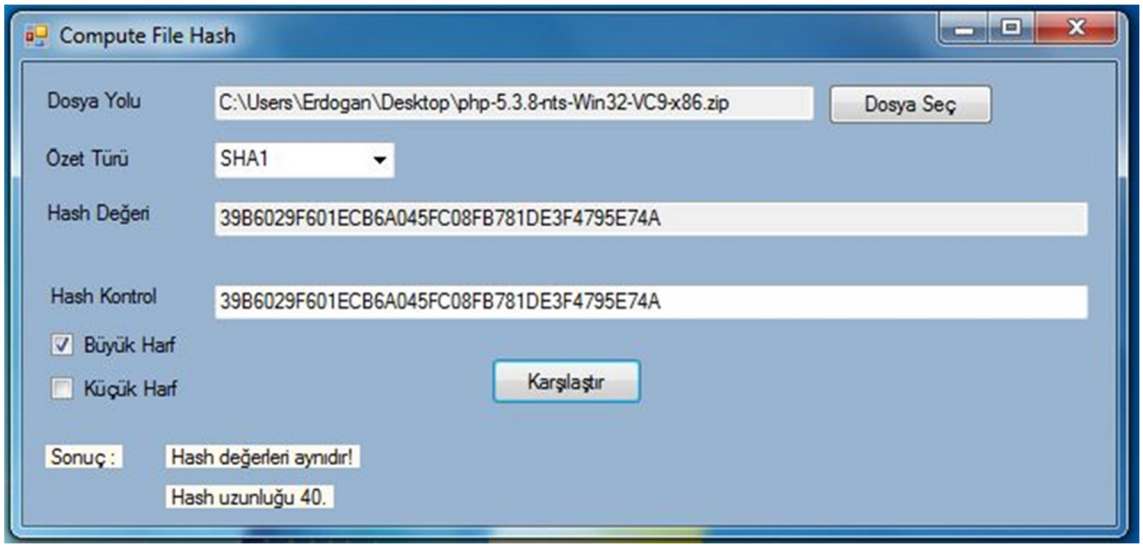
Dosya bütünlüğünün kontrolü için yapılmış olan çalışma sayesinde (hash değeri bilinen) internetten indirdiğimiz veya mesaj yoluyla aldığımız herhangi bir dosyanın hash değerlerini bulup karşılaştırabiliyoruz.

Örneğin *http://windows.php.net/download/* adresinden PHP 5.3 (5.3.8) sürümünün sıkıştırılmış (zip'li) halini indirilip SHA1 hash değerini kontrol edilmiştir.



Şekil 4.1. Dosya Bütünlüğü

"php-5.3.8-nts-Win32-VC9-x86.zip" adlı dosyamızın SHA1 özet değerinin kontrol edilmiş hali Şekil 4.2'de verilmiştir.



Şekil 4.2. Dosya Bütünlüğü Kontrol Programı

Karşılaştırma sonrasında internetten indirdiğimiz programın dosya bütünlüğü kontrol edilip dosya bütünlüğünün korunduğunu tespit edilmiştir.

Programımızda MD5, SHA1, SHA256, SHA384 ve SHA512 özet değerlerinin kontrolü gerçekleştirilebilmektedir.

4.2. Şifre Koruması

Kriptografik Özet Fonksiyonlarından MD5 kullanarak özet değerleri alınan kullanıcı şifreleri veri tabanına kaydedilmiştir. Bu işlem için kullandığımız web ara yüzü Şekil 4.3'te verilmiştir.



Şekil 4.3. Web Ara Yüzü

Kullanıcı bilgileri *MySql* veri tabanına uye tablosuna Şekil 4.4'te görüldüğü üzere kaydedilmiştir.

Ad	Soyad	Email	SifreMD5
Engin	Çalımfidan	engin_calimfidan@mynet.com	E6EF73806CD72F12E42E1340919EE782
nurdan	karaman	nursel_baby_face@hotmail.com	CE07DD96BED8EB38EE0808088CAFA5D8
İbrahim	Gündüz	ibrahimgunduz1991@gmail.com	30B899CE9AE40A142D270BB16F1D1FF4
Ali	VARLI	zuper_basketci@hotmail.com	AD42471A5BD4198406BAE22F4D932E9D
m sami	gürpınar	msamig01@gmail.com	D051533BFA2E172CA61C069634EF3275
Recep	Akyüz	recepexx@hotmail.com	96E79218965EB72C92A549DD5A330112
tolga	sakallı	tolga@trakya.edu.tr	72113B401A454EA7EBAC1F2051B9D05F
Onur	Erdoğan	onurerdogan1991@gmail.com	3255A08E672A5FA222D6E2DE9EF5B882
metin	ögtem	metinogtem@gmail.com	114BFC0DF1CC2C2C3176F09E74BBDD62
FATİH	KARA	fatihka_569@hotmail.com	E10ADC3949BA59ABBE56E057F20F883E
Semih	Çetinkaya	semihcetinkaya@fenerbahce.com.tr	51EE1C19128AC6FF92994EB2E8AF2DE3

Şekil 4.4. Kullanıcı Verileri Tablosu

Şekil 4.4'te görüldüğü üzere kullanıcı şifrelerinin sadece hash değerleri tabloya eklenmiştir. Bu sayede şifre güvenliğini sağlamıştır.

5. KÜÇÜLTÜLMÜŞ MD5 ALGORİTMASI: MİNİ-MD5

Bu bölümde, MD5 algoritmasının küçültülmüş bir benzeri olan ve Mini-MD5 olarak adlandırdığımız kriptografik hash fonksiyonu verilmiştir. MD5 ve Mini-MD5 fonksiyonlarının yapısal özelliklerinin karşılaştırılması Tablo5.1'de yapılmıştır.

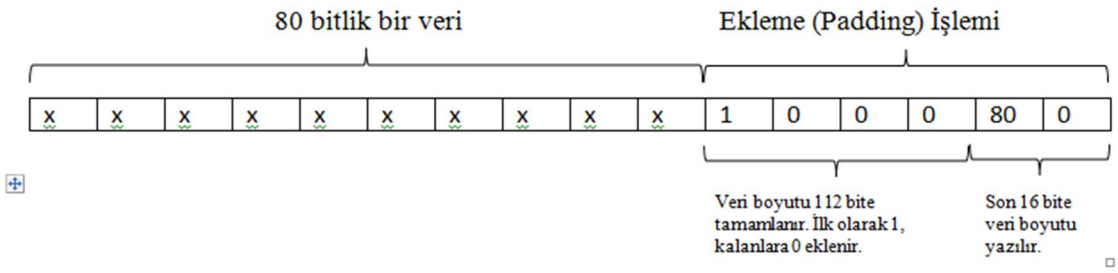
Tablo 5.1. Mini-MD5 ve MD5 Fonksiyonlarının Karşılaştırılması

	MD5	**Mini-MD5**
Blok Boyutu	512 bit (64 byte)	128 bit (16 byte)
Kelime Boyutu	32 bit (4 byte)	8 bit (1 byte)
Başlangıç Değerleri	32 * 4 = 128 bit (16 byte)	8 * 4 = 32 bit (4 byte)
Veri Boyutunun Yazıldığı Alan	64 bit (8 byte)	16 Bit (2 byte)
Döngü Sayısı	64	16

Tablo5.1'daki özellikleri bu bölümde açıklanmıştır.

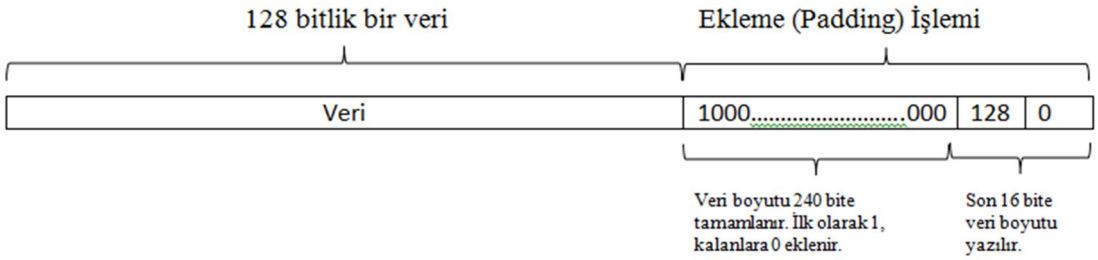
5.1. Mini-MD5'te Ekleme (Padding) İşlemi

Mini-MD5 fonksiyonu veriyi 128 bitlik bloklar halinde işler. Eğer veri boyutu 128 bitin katı değilse ekleme işlemi yapılır ve veri boyutu 128 bitin katı olacak şekilde oluşturulur. Her bir sütun 8 biti temsil etmek üzere 128 bitlik bir bloğun yapısı Şekil5.1'de verilmiştir. Veri boyutu 80 bit olarak alınmıştır.



Şekil 5.1. 128 Bitlik Bir Bloğun Yapısı ve Ekleme (Padding) İşlemi

Şekil 5.1'de 80 bitlik bir verinin blok yapısı ve nasıl 128 bitlik bir bloğa tamamlandığı yani ekleme işlemi gösterilmiştir. Burada akla gelecek ilk soru "Eğer verimizin boyutu 128 bit ve katları olursa o zaman nasıl bir ekleme işlemi yapılmalıdır?" şeklinde olabilir. Böyle bir durumda 128 bitlik verimize ek olarak 128 bitlik bir ekleme işlemi yapılır. Yani 129. bit 1, geri kalan bitler 240. bite kadar 0 eklenerek doldurulur. Son 16 bit ise veri boyutunu yani 128 verisini tutar. Bu işlemin anlatımı Şekil5.2'de verilmiştir.



Şekil 5.2. 128 Bitlik Bir Veriye Uygulanan Ekleme İşlemi

Akla gelecek diğer bir soru da "Veri boyutu en çok kaç bit olabilir?" olabilir. Veri boyutunu yazmak için128 bitlik bloğun son 16 bitlik kısmını kullandığımızdan, 2^{16} = 65536 (8192 byte) bitlik bir veri boyutu bu alana yazılabilir. Aslında bu veri boyutu, ekleme işlemi yapıldıktan sonraki gerçek veri boyutudur. Yani ekleme işleminde gerçek veriden sonra eklediğimiz 1 değeri (8 bit'lik alan kaplar) ve veri boyutunu yazdığımız kısım (16 bit'lik alan kaplar) toplamda 24 bit (3 byte) alan kaplar. Bu alanı 65536 bit'ten (8192 byte) çıkarmamız gerekmektedir. Onun için en çok (65536 – 24) 65512

bit'lik (8189 byte) veri boyutu yazılabilir.65536 bit'ten (8189byte) daha büyük verilerde 2^{16}'ya göre mod alma işlemi yapılır ve veri boyutu olarak mod işleminin sonucu yazılır. Dolayısıyla Mini-MD5 fonksiyonunda veri boyutu sınırlı değildir (MD5 fonksiyonunda da veri boyutu için sınır yoktur.).

5.2. Mini-MD5'te Başlangıç Değerleri (Initial Values)

Mini-MD5 fonksiyonun başlangıç değerleri Tablo5.2'de verilmiştir. Değerler hexadecimal olarak kullanılmıştır.

Tablo 5.2. Başlangıç Değerleri

Başlangıç Değerleri (Initial Values)			
67	EF	9D	7F

Başlangıç değerleri 8 bitlik 4 hexadecimal değerden oluşmaktadır. Bu değerler Mini-MD5 içerisinde kullandığımız ilkel fonksiyonların kullanacağı ilk değerler olduğu için hash fonksiyonunda çok önemli bir yere sahiptir.

5.3. Mini-MD5'te Kullanılan İlkel Fonksiyonlar

Mini-MD5 fonksiyonunda kullandığımız ilkel fonksiyonlar MD5 ile aynı olup Tablo 5.3'te verilmiştir. Bu fonksiyonlar MD5 fonksiyonunda 16'şar kez kullanılırken Mini-MD5'te 4'er kez kullanılmıştır. MD5 fonksiyonu içerisinde toplamda 64 olan döngü sayısı Mini-MD5'te 16 olmuştur.

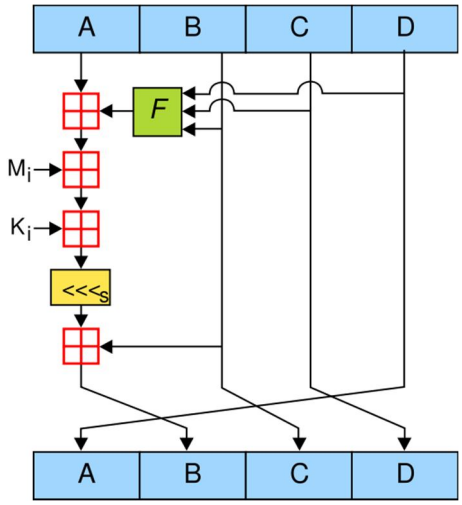
Tablo 5.3. Mini-MD5 İçerisinde Kullanılan İlkel Fonksiyonlar

Tur	İlkel Fonksiyon	Adım(64)
1	$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$	$0 \leq j \leq 3$
2	$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$	$4 \leq j \leq 7$
3	$H(X,Y,Z) = X \oplus Y \oplus Z$	$8 \leq j \leq 11$
4	$J(X,Y,Z) = Y \oplus (X \vee \neg Z)$	$12 \leq j \leq 15$

($Kullanılan\ işaretlerin\ anlamları$: $\wedge = ve$, $\vee = veya$, $\oplus = xor$, $\neg =$ değil)

5.4. Mini-MD5'in Çalışma Şekli

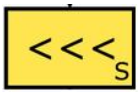
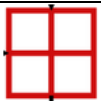
Mini-MD5 fonksiyonunda başlangıç değerleri belirlendikten sonra Tablo 5.3'te verilen ilkel fonksiyonlar kullanılarak yeni değerler üretilir ve bu değerler bir sonraki fonksiyona aktarılır. Bu işlemler Şekil 5.3'te gösterilmiştir.



Şekil 5.3. Mini-MD5 Fonksiyonun Çalışma Şekli

Şekil5.3'te A, B, C, D, değerleri ilk olarak başlangıç değerleri olup bu ilk değerler AA, BB, CC, DD gibi değişkenlere aktarıldıktan sonra ilkel fonksiyonlar tarafından işlenirler. Yeni oluşan değerler bir sonraki adıma aktarılır. İşlemler bu şekilde devam eder.16. adımdan sonra oluşan A, B, C, D değerleri ile AA, BB, CC, DD değerleri toplanarak son değerler elde edilir. Oluşan bu son A, B, C ve D değerlerinin birleşmesi sonucu özet değerimiz elde edilmiş olur. Şekil 5.3'te kullanılan ifadelerin anlamları Tablo 5.4'te verilmiştir.

Tablo 5.4. Şekil 5-3'te Kullanılan İfadelerin Anlamları

	İlkel fonksiyonları belirtir.
M_i	Özet değeri hesaplanacak verimizin ekleme işlemi yapıldıktan sonraki halinin 8 bitlik (1 byte) değerler olarak bir dizide tutulmasını belirtir.
K_i	Abs(Floor((Sin(i)*100))), i değeri 1 ile 16 arasındaki değerlerden oluşur. Sonuç olarak başvuru çizelgesini oluşturur.
	Sola kaydırma işlemini belirtir. Kaydırma miktarı ilkel fonksiyonların içinde belirlenmiştir.
	$Mod2^8$'e göre toplama işlemi yapıldığını belirtir.

T[i]=Abs(Floor((Sin(i)*100))) ifadesinin i'nin 1 ile 16 arasındaki değerleri için üretmiş olduğu değerler Tablo 5.5'te verilmiştir.

Tablo 5.5. Başvuru Çizelgesi (LookupTable)

Başvuru Çizelgesi (LookupTable)							
54	5a	0e	4c	60	1c	41	62
29	37	64	36	2a	63	41	1d

Mini-MD5 fonksiyonu oluşturan ilkel fonksiyon değerleri Tablo 5.6'da verilmiştir.

Tablo 5.6. Mini-Md5 Fonksiyonu Oluşturan İlkel Fonksiyonların Değerleri

Adım	**Fonksiyon İçeriği (Fonksiyon(A,B,C,D, m, s, k))** **m: M_i ifadesindeki indis değeri; s: sola kaydırma miktarı; k: Başvuru çizelgesindeki indis değeri**
1	`F(refA, B, C, D, 0, 1, 1);`
2	`F(ref D, A, B, C, 1, 3, 2);`
3	`F(ref C, D, A, B, 2, 5, 3);`
4	`F(ref B, C, D, A, 3, 7, 4);`
5	`G(ref A, B, C, D, 4, 2, 5);`
6	`G(ref D, A, B, C, 5, 4, 6);`
7	`G(ref C, D, A, B, 6, 6, 7);`
8	`G(ref B, C, D, A, 7, 0, 8);`
9	`H(ref A, B, C, D, 8, 3, 9);`
10	`H(ref D, A, B, C, 9, 5, 10);`
11	`H(ref C, D, A, B, 10, 7, 11);`
12	`H(ref B, C, D, A, 11, 1, 12);`
13	`I(ref A, B, C, D, 12, 4, 13);`
14	`I(ref D, A, B, C, 13, 6, 14);`
15	`I(ref C, D, A, B, 14, 0, 15);`
16	`I(ref B, C, D, A, 15, 2, 1);`

Kullanmış olduğumuz bazı fonksiyonlar ve bu fonksiyonların işlevleri sırayla açıklanmıştır.

RotateLeft Fonksiyonu

Sola kaydırma işlemi için kullanmış olduğumuz fonksiyon şu şekildedir:

publicstatic byte RotateLeft(byte uiNumber, ushortshift)

{ return ((uiNumber>> 8 - shift) | (uiNumber<<shift)); }

Başvuru Çizelgesi (LookupTable)

Abs(Floor((Sin(i)*100))) ifadesinin i'nin 1 ile 16 arasındaki değerleri için üretmiş olduğu değerler sonucu oluşan başvuru çizelgesidir ve bu değerler Tablo 5.5'te

verilmiştir. Tablo 5.5'te verilen değerle hexadecimal iken çalışma esnasında bu değerler decimal olarak kullanılır. Decimal olan değerler aşağıda verilmiştir.

T[]={84, 90, 14, 76, 96, 28, 65, 98, 41, 55, 100, 54, 42, 99, 65, 29};

Mini-MD5 fonksiyonun çalışma mantığı anlatıldıktan sonra bu işlemlerin çalışması adım adım Örnek 5.1'de ve Örnek 5.2'de gösterilmiştir.

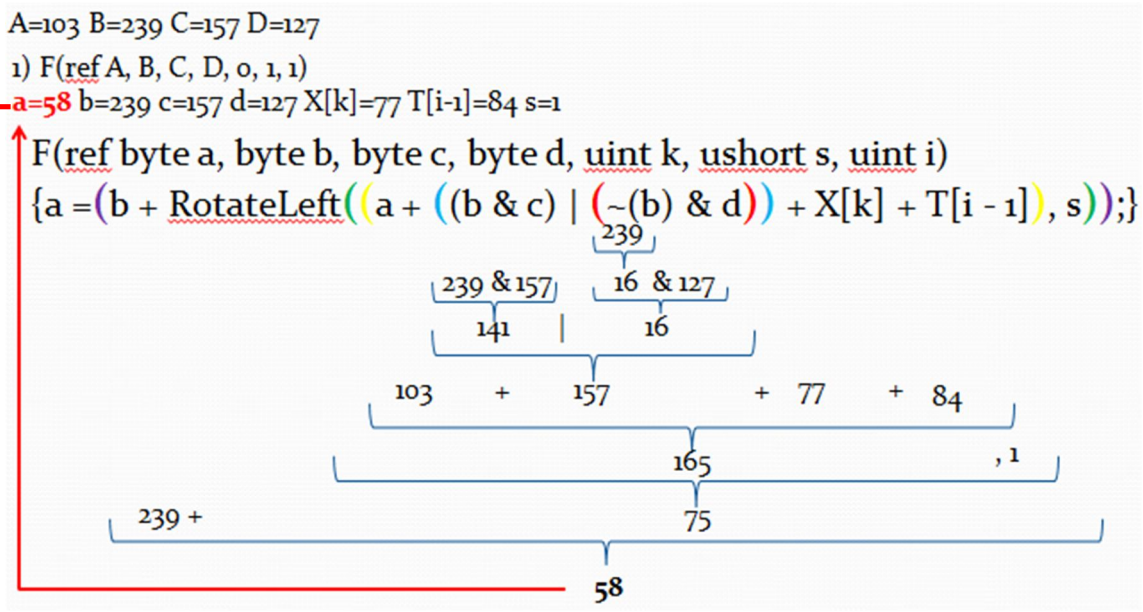
Örnek 5.1.

Aşağıda boşluk karakterinin Mini-MD5 özet değeri hesaplanmıştır.

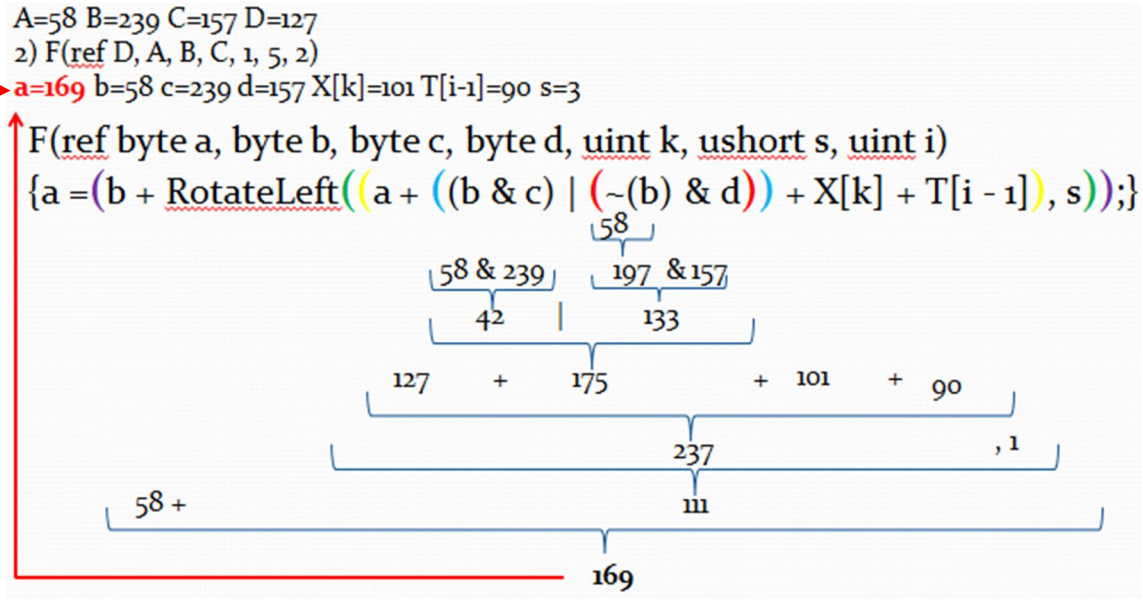
X[] dizisi veriyi okuyarak her bir karakterin ASCII değerlerini içine attığımız dizidir. Dizinin her bir elemanı 8-bit yani 1 byte'dır. Bu örneğimizde boşluk karakterinin özet değeri hesaplandığı için dizimizin birinci, ikinci ve on beşinci elemanı hariç tüm elemanları *0 (sıfır)* değeri alacaktır. Boşluk karakterinin ASCII değeri *32*'dir. Bunun için ilk elemanı *32* değeri alacaktır. İkinci eleman *128* değeri alacaktır. *128* değeri verimizin son karakterinden sonra ekleme işlemi için kullandığımız *1* değerini ifade etmektedir. Üçüncü elemandan on beşinci elemana kadar olan değerler *0 (sıfır)* olacaktır. On beşinci eleman *8* ve on altıncı eleman da *0 (sıfır)* değerini alacaktır (Bölüm 5.1 Ekleme İşlemi'nden hatırlayalım). On beşinci elemanın değerinin *8* olmasının sebebi boşluk karakterinin boyutu olan *8* bit'i ifade etmesidir. Yani on beşinci karakter veri boyutunu yazdığımız alandır (*15.* byte). On altıncı byte'ın sıfır olması da veri boyutunun ilk alana (*15.* byte) yazılabilmiş olmasıdır. Veri boyutu *256* (*32* byte) bit veya daha büyük olsaydı bu alan da (*16.* byte) kullanılacaktı.

Tüm ilkel fonksiyonlarımızın çalışma mantığı aynı olup, almış oldukları *ref* değeri üzerinde değişiklik yapmaktadırlar. Bu değişiklik için almış oldukları verileri bitsel işlemlere tabi tutarak sonuçları *ref* değişkenine atarlar. Her ilkel fonksiyonun çalışması sonucu oluşan bu yeni değer bir sonraki ilkel fonksiyona aktarılır ve işlemler bu şekilde devam eder.

Şekil 5.4'te ve Şekil 5.5'te ilkel fonksiyonun çalışma esnasındaki işlem adımları verilmiştir. İşlemlerde kullanılan tüm değerler decimal olarak alınmıştır.



Şekil 5.4. İlkel Fonksiyonun İşlem Adımları 1



Şekil 5.5.İlkel Fonksiyonun İşlem Adımları 2

Şekil 5.4 ve Şekil 5.5 incelendiğinde, ilkel fonksiyon tarafından ref olarak alınan değerin işlemler sonucu değişmiş olduğu ve bu yeni değerin bir sonraki ilkel fonksiyona aktarılacağı görülmüştür. Şekil 5.4 ve Şekil 5.5'te kullanılan değerler Örnek 5.2'den alınmıştır.

Aşağıda da ilkel fonksiyonlarımız ve bu fonksiyonların her biri için ayrı ayrı işlem adımları verilmiştir. Fonksiyonlarda kullanılan tüm değerler decimal olarak alınmıştır.

F(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)

{a = (b + RotateLeft((a + ((b & c) | (~(b) & d)) + X[k] + T[i - 1]), s));}

Not: "|": "veya", "~": "değil", "&": "ve", "^": "xor" işlemini belirtmektedir.

A=103 B=239 C=157 D=127//A,B,C,D Değerlerinin 1. F fonksiyonuna girmeden önceki hali. Bu değerler 16. adımdan sonra kullanılmak üzere AA, BB, CC, DD değişkenlerinde tutulmaktadır.

1) F(ref A, B, C, D, 0, 1, 1)

a=223 b=239 c=157 d=127 X[k]=32 T[i-1]=84 s=1

A=223 B=239 C=157 D=127 //A,B,C,D Değerlerinin 1. adımdan sonraki ve 2. adımda yer alan F fonksiyonuna girmeden önceki halidir.

2) F(ref D, A, B, C, 1, 5, 2)

a=32 b=223 c=239 d=157 X[k]=128 T[i-1]=90 s=3

A=223 B=239 C=157 D=32

3) F(ref C, D, A, B, 2, 9, 3)

a=111 b=32 c=223 d=239 X[k]=0 T[i-1]=14 s=5

A=223 B=239 C=111 D=32

4) F(ref B, C, D, A, 3, 13, 4)

a=100 b=111 c=32 d=223 X[k]=0 T[i-1]=76 s=7

A=223 B=100 C=111 D=32

G(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)

{a = (b + RotateLeft((a + ((b & d) | (c & ~d)) + X[k] + T[i - 1]), s));}

5) G(ref A, B, C, D, 4, 2, 5)

a=30 b=100 c=111 d=32 X[k]=0 T[i-1]=96 s=2

A=30 B=100 C=111 D=32

6) G(ref D, A, B, C, 5, 6, 6)
a=194 b=30 c=100 d=111 X[k]=0 T[i-1]=28 s=4
A=30 B=100 C=111 D=194

7) G(ref C, D, A, B, 6, 10, 7)
a=68 b=194 c=30 d=100 X[k]=0 T[i-1]=65 s=6
A=30 B=100 C=68 D=194

8) G(ref B, C, D, A, 7, 14, 8)
a=206 b=68 c=194 d=30 X[k]=0 T[i-1]=98 s=0
A=30 B=206 C=68 D=194

H(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (b ^ c ^ d) + X[k] + T[i - 1]), s));}

9) H(ref A, B, C, D, 8, 3, 9)
a=74 b=206 c=68 d=194 X[k]=0 T[i-1]=41 s=3
A=74 B=206 C=68 D=194

10) H(ref D, A, B, C, 9, 7, 10)
a=129 b=74 c=206 d=68 X[k]=0 T[i-1]=55 s=5
A=74 B=206 C=68 D=129

11) H(ref C, D, A, B, 10, 11, 11)
a=87 b=129 c=74 d=206 X[k]=0 T[i-1]=100 s=7
A=74 B=206 C=87 D=129

12) H(ref B, C, D, A, 11, 15, 12)
a=152 b=87 c=129 d=74 X[k]=0 T[i-1]=54 s=1
A=74 B=152 C=87 D=129

I(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (c ^ (b | ~d)) + X[k] + T[i - 1]), s));}

13) I(ref A, B, C, D, 12, 4, 13)
a=105 b=152 c=87 d=129 X[k]=0 T[i-1]=42 s=4
A=105 B=152 C=87 D=129

14) I(ref D, A, B, C, 13, 8, 14)
a=190 b=105 c=152 d=87 X[k]=0 T[i-1]=99 s=6
A=105 B=152 C=87 D=190

15) I(ref C, D, A, B, 14, 12, 15)

a=244 b=190 c=105 d=152 X[k]=8 T[i-1]=65 s=0
A=105 B=152 C=244 D=190

16) I(ref B, C, D, A, 15, 1, 1)
a=196 b=244 c=190 d=105 X[k]=0 T[i-1]=84 s=2
A=105 B=196 C=244 D=190//A,B,C,D Değerlerinin son hali

A=105 B=196 C=244 D=190//A,B,C,D Değerlerinin son hali
AA=103 BB=239 CC=157 DD=127//A,B,C,D Değerlerinin ilk hali (AA-BB-CC-DD)

(A + AA) A= 208 = d0 // Toplama işlemi sonucunda oluşan değerler
(B + BB) B= 179 = b3// Toplama işlemi (A+AA) Mod 2^8'e göre yapılır
(C + CC) C= 145 = 91
(D + DD) D= 61 = 3d

A,B,C,D Değerleri Hexadecimal formata çevrilmeden önce Reverse4Bit işlemine tabi tutulur (d0-b3-91-3d => 0d-3b-19-d3). Son olarak bu değerlerin Hexadecimal değerleri girilen verinin hash değeri olarak alınır. Boşluk karakterinin Mini-MD5 özet değeri (0d3b19d3) olarak hesaplanmıştır.

Reverse4Bit fonksiyonu

Reverse4Bit fonksiyonu almış olduğu byte değerin ilk 4 bit ile son 4 bit'inin yerlerini değiştirmektedir. Reverse4Bit fonksiyonu şu şekildedir:

```
publicstatic byte Reverse4Bit(byte uiNumber)
{
return (byte)(((uiNumber& 0x0f) << 4) | ((uiNumber& 0xf0) >> 4));
}
```

Örnek 5.2.

"Merhaba" verisinin Mini-MD5 özetinin hesaplanması bu örnekte verilmiştir.

Sabitler:

T[]={84, 90, 14, 76, 96, 28, 65, 98, 41, 55, 100, 54, 42, 99, 65, 29};

X[]={77, 101, 114, 104, 97, 98, 97, 128, 0, 0, 0, 0, 0, 0, 56, 0}

F(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)

{a = (b + RotateLeft((a + ((b & c) | (~(b) & d)) + X[k] + T[i - 1]), s));}

A=103 B=239 C=157 D=127 // Başlangıç değerleri (initial value)

1) F(ref A, B, C, D, 0, 1, 1)

a=58 b=239 c=157 d=127 X[k]=77 T[i-1]=84 s=1

A=58 B=239 C=157 D=127

2) F(ref D, A, B, C, 1, 5, 2)

a=169 b=58 c=239 d=157 X[k]=101 T[i-1]=90 s=3

A=58 B=239 C=157 D=169

3) F(ref C, D, A, B, 2, 9, 3)

a=26 b=169 c=58 d=239 X[k]=114 T[i-1]=14 s=5

A=58 B=239 C=26 D=169

4) F(ref B, C, D, A, 3, 13, 4)
a=255 b=26 c=169 d=58 X[k]=104 T[i-1]=76 s=7
A=58 B=255 C=26 D=169

G(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + ((b & d) | (c & ~d)) + X[k] + T[i - 1]), s));}

5) G(ref A, B, C, D, 4, 2, 5)
a=217 b=255 c=26 d=169 X[k]=97 T[i-1]=96 s=2
A=217 B=255 C=26 D=169

6) G(ref D, A, B, C, 5, 6, 6)
a=27 b=217 c=255 d=26 X[k]=98 T[i-1]=28 s=4
A=217 B=255 C=26 D=27

7) G(ref C, D, A, B, 6, 10, 7)
a=16 b=27 c=217 d=255 X[k]=97 T[i-1]=65 s=6
A=217 B=255 C=16 D=27

8) G(ref B, C, D, A, 7, 14, 8)
a=3 b=16 c=27 d=217 X[k]=128 T[i-1]=98 s=0
A=217 B=3 C=16 D=27

H(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (b ^ c ^ d) + X[k] + T[i - 1]), s));}

9) H(ref A, B, C, D, 8, 3, 9)
a=83 b=3 c=16 d=27 X[k]=0 T[i-1]=41 s=3
A=83 B=3 C=16 D=27

10) H(ref D, A, B, C, 9, 7, 10)
a=165 b=83 c=3 d=16 X[k]=0 T[i-1]=55 s=5
A=83 B=3 C=16 D=165

11) H(ref C, D, A, B, 10, 11, 11)
a=89 b=165 c=83 d=3 X[k]=0 T[i-1]=100 s=7
A=83 B=3 C=89 D=165

12) H(ref B, C, D, A, 11, 15, 12)
a=42 b=89 c=165 d=83 X[k]=0 T[i-1]=54 s=1
A=83 B=42 C=89 D=165

I(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (c ^ (b | ~d)) + X[k] + T[i - 1]), s));}

13) I(ref A, B, C, D, 12, 4, 13)
a=52 b=42 c=89 d=165 X[k]=0 T[i-1]=42 s=4
A=52 B=42 C=89 D=165

14) I(ref D, A, B, C, 13, 8, 14)
a=93 b=52 c=42 d=89 X[k]=0 T[i-1]=99 s=6
A=52 B=42 C=89 D=93

15) I(ref C, D, A, B, 14, 12, 15)
a=24 b=93 c=52 d=42 X[k]=56 T[i-1]=65 s=0
A=52 B=42 C=24 D=93

16) I(ref B, C, D, A, 15, 1, 1)
a=40 b=24 c=93 d=52 X[k]=0 T[i-1]=84 s=2
A=52 B=40 C=24 D=93

A=52 B=40 C=24 D=93
AA=103 BB=239 CC=157 DD=127

(A + AA) A= 155 = 9b
(B + BB) B= 23 = 17
(C + CC) C= 181 = b5
(D + DD) D= 220 = dc

Reverse4Bit işleminden sonraki sonuç:

(9b-17-b5-dc)=(b9-71-5b-cd)

"Merhaba" değerinin Mini-MD5 özet değeri "b9715bcd" olarak hesaplanmıştır.

Örnek 5.3.

"Merhaba Dünya! Mini-MD5 değerini buluyoruz." verisinin Mini-MD5 özetinin hesaplanması aşağıda verilmiştir.

T[]={84, 90, 14, 76, 96, 28, 65, 98, 41, 55, 100, 54, 42, 99, 65, 29};

X[]={77, 101, 114, 104, 97, 98, 97, 32, 68, 252, 110, 121, 97, 33,32, 77,

105, 110, 105, 45, 77, 68, 53, 32, 100, 101, 31, 101, 114, 105, 110, 105,

32, 98, 117, 108, 117, 121, 111, 114, 117, 122, 46, 128, 0, 0, 88, 1}

Verimizin boyutu 43 x 8 = 344 bit olduğu için 3 blokta işlenecektir.

**********1. Blok **********

F(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)

{a = (b + RotateLeft((a + ((b & c) | (~(b) & d)) + X[k] + T[i - 1]), s));}

A=103 B=239 C=157 D=127

1) F(ref A, B, C, D, 0, 1, 1)

a=58 b=239 c=157 d=127 X[k]=77 T[i-1]=84 s=1

A=58 B=239 C=157 D=127

2) F(ref D, A, B, C, 1, 5, 2)

a=169 b=58 c=239 d=157 X[k]=101 T[i-1]=90 s=3

A=58 B=239 C=157 D=169

3) F(ref C, D, A, B, 2, 9, 3)

a=26 b=169 c=58 d=239 X[k]=114 T[i-1]=14 s=5

A=58 B=239 C=26 D=169

4) F(ref B, C, D, A, 3, 13, 4)
a=255 b=26 c=169 d=58 X[k]=104 T[i-1]=76 s=7
A=58 B=255 C=26 D=169

G(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + ((b & d) | (c & ~d)) + X[k] + T[i - 1]), s));}

5) G(ref A, B, C, D, 4, 2, 5)
a=217 b=255 c=26 d=169 X[k]=97 T[i-1]=96 s=2
A=217 B=255 C=26 D=169

6) G(ref D, A, B, C, 5, 6, 6)
a=27 b=217 c=255 d=26 X[k]=98 T[i-1]=28 s=4
A=217 B=255 C=26 D=27

7) G(ref C, D, A, B, 6, 10, 7)
a=16 b=27 c=217 d=255 X[k]=97 T[i-1]=65 s=6
A=217 B=255 C=16 D=27

8) G(ref B, C, D, A, 7, 14, 8)
a=163 b=16 c=27 d=217 X[k]=32 T[i-1]=98 s=0
A=217 B=163 C=16 D=27

H(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (b ^ c ^ d) + X[k] + T[i - 1]), s));}

9) H(ref A, B, C, D, 8, 3, 9)
a=26 b=163 c=16 d=27 X[k]=68 T[i-1]=41 s=3
A=26 B=163 C=16 D=27

10) H(ref D, A, B, C, 9, 7, 10)
a=24 b=26 c=163 d=16 X[k]=252 T[i-1]=55 s=5
A=26 B=163 C=16 D=24

11) H(ref C, D, A, B, 10, 11, 11)
a=217 b=24 c=26 d=163 X[k]=110 T[i-1]=100 s=7
A=26 B=163 C=217 D=24

12) H(ref B, C, D, A, 11, 15, 12)
a=51 b=217 c=24 d=26 X[k]=121 T[i-1]=54 s=1
A=26 B=51 C=217 D=24

I(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (c ^ (b | ~d)) + X[k] + T[i - 1]), s));}

13) I(ref A, B, C, D, 12, 4, 13)
a=112 b=51 c=217 d=24 X[k]=97 T[i-1]=42 s=4
A=112 B=51 C=217 D=24

14) I(ref D, A, B, C, 13, 8, 14)
a=232 b=112 c=51 d=217 X[k]=33 T[i-1]=99 s=6
A=112 B=51 C=217 D=232

15) I(ref C, D, A, B, 14, 12, 15)
a=190 b=232 c=112 d=51 X[k]=32 T[i-1]=65 s=0
A=112 B=51 C=190 D=232

16) I(ref B, C, D, A, 15, 1, 1)
a=106 b=190 c=232 d=112 X[k]=77 T[i-1]=84 s=2
A=112 B=106 C=190 D=232

A=112 B=106 C=190 D=232
AA=103 BB=239 CC=157 DD=127

(A + AA) A= 215 = d7
(B + BB) B= 89 = 59
(C + CC) C= 91 = 5b
(D + DD) D= 103 = 67

********** Blok Sonu **********

**********2. Blok **********

F(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + ((b & c) | (~(b) & d)) + X[k] + T[i - 1]), s));}

A=215 B=89 C=91 D=103

1) F(ref A, B, C, D, 0, 1, 1)
a=127 b=89 c=91 d=103 X[k]=105 T[i-1]=84 s=1
A=127 B=89 C=91 D=103

2) F(ref D, A, B, C, 1, 5, 2)
a=195 b=127 c=89 d=91 X[k]=110 T[i-1]=90 s=3
A=127 B=89 C=91 D=195
3) F(ref C, D, A, B, 2, 9, 3)
a=104 b=195 c=127 d=89 X[k]=105 T[i-1]=14 s=5
A=127 B=89 C=104 D=195

4) F(ref B, C, D, A, 3, 13, 4)
a=252 b=104 c=195 d=127 X[k]=45 T[i-1]=76 s=7
A=127 B=252 C=104 D=195

G(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + ((b & d) | (c & ~d)) + X[k] + T[i - 1]), s));}

5) G(ref A, B, C, D, 4, 2, 5)
a=76 b=252 c=104 d=195 X[k]=77 T[i-1]=96 s=2
A=76 B=252 C=104 D=195

6) G(ref D, A, B, C, 5, 6, 6)
a=75 b=76 c=252 d=104 X[k]=68 T[i-1]=28 s=4
A=76 B=252 C=104 D=75

7) G(ref C, D, A, B, 6, 10, 7)
a=212 b=75 c=76 d=252 X[k]=53 T[i-1]=65 s=6
A=76 B=252 C=212 D=75

8) G(ref B, C, D, A, 7, 14, 8)
a=153 b=212 c=75 d=76 X[k]=32 T[i-1]=98 s=0
A=76 B=153 C=212 D=75

H(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (b ^ c ^ d) + X[k] + T[i - 1]), s));}

9) H(ref A, B, C, D, 8, 3, 9)
a=151 b=153 c=212 d=75 X[k]=100 T[i-1]=41 s=3
A=151 B=153 C=212 D=75

10) H(ref D, A, B, C, 9, 7, 10)
a=207 b=151 c=153 d=212 X[k]=101 T[i-1]=55 s=5
A=151 B=153 C=212 D=207

11) H(ref C, D, A, B, 10, 11, 11)
a=219 b=207 c=151 d=153 X[k]=31 T[i-1]=100 s=7
A=151 B=153 C=219 D=207

12) H(ref B, C, D, A, 11, 15, 12)
a=74 b=219 c=207 d=151 X[k]=101 T[i-1]=54 s=1
A=151 B=74 C=219 D=207

I(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (c ^ (b | ~d)) + X[k] + T[i - 1]), s));}

13) I(ref A, B, C, D, 12, 4, 13)
a=151 b=74 c=219 d=207 X[k]=114 T[i-1]=42 s=4
A=151 B=74 C=219 D=207

14) I(ref D, A, B, C, 13, 8, 14)
a=189 b=151 c=74 d=219 X[k]=105 T[i-1]=99 s=6
A=151 B=74 C=219 D=189

15) I(ref C, D, A, B, 14, 12, 15)
a=113 b=189 c=151 d=74 X[k]=110 T[i-1]=65 s=0
A=151 B=74 C=113 D=189

16) I(ref B, C, D, A, 15, 1, 1)
a=160 b=113 c=189 d=151 X[k]=105 T[i-1]=84 s=2
A=151 B=160 C=113 D=189

A=151 B=160 C=113 D=189
AA=215 BB=89 CC=91 DD=103

(A + AA) A= 110 = 6e
(B + BB) B= 249 = f9
(C + CC) C= 204 = cc
(D + DD) D= 36 = 24
****************************** Blok Sonu ******************************

******************************3. Blok ******************************

F(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + ((b & c) | (~(b) & d)) + X[k] + T[i - 1]), s));}

A=110 B=249 C=204 D=36

1) F(ref A, B, C, D, 0, 1, 1)
a=86 b=249 c=204 d=36 X[k]=32 T[i-1]=84 s=1
A=86 B=249 C=204 D=36

2) F(ref D, A, B, C, 1, 5, 2)
a=27 b=86 c=249 d=204 X[k]=98 T[i-1]=90 s=3
A=86 B=249 C=204 D=27

3) F(ref C, D, A, B, 2, 9, 3)
a=67 b=27 c=86 d=249 X[k]=117 T[i-1]=14 s=5
A=86 B=249 C=67 D=27

4) F(ref B, C, D, A, 3, 13, 4)
a=167 b=67 c=27 d=86 X[k]=108 T[i-1]=76 s=7
A=86 B=167 C=67 D=27

```
G(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + ((b & d) | (c & ~d)) + X[k] + T[i - 1]), s));}
```

5) G(ref A, B, C, D, 4, 2, 5)
a=96 b=167 c=67 d=27 X[k]=117 T[i-1]=96 s=2
A=96 B=167 C=67 D=27
6) G(ref D, A, B, C, 5, 6, 6)
a=169 b=96 c=167 d=67 X[k]=121 T[i-1]=28 s=4
A=96 B=167 C=67 D=169

7) G(ref C, D, A, B, 6, 10, 7)
a=222 b=169 c=96 d=167 X[k]=111 T[i-1]=65 s=6
A=96 B=167 C=222 D=169

8) G(ref B, C, D, A, 7, 14, 8)
a=34 b=222 c=169 d=96 X[k]=114 T[i-1]=98 s=0
A=96 B=34 C=222 D=169

```
H(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (b ^ c ^ d) + X[k] + T[i - 1]), s));}
```

9) H(ref A, B, C, D, 8, 3, 9)
a=188 b=34 c=222 d=169 X[k]=117 T[i-1]=41 s=3
A=188 B=34 C=222 D=169

10) H(ref D, A, B, C, 9, 7, 10)
a=15 b=188 c=34 d=222 X[k]=122 T[i-1]=55 s=5
A=188 B=34 C=222 D=15

11) H(ref C, D, A, B, 10, 11, 11)
a=143 b=15 c=188 d=34 X[k]=46 T[i-1]=100 s=7
A=188 B=34 C=143 D=15

12) H(ref B, C, D, A, 11, 15, 12)
a=183 b=143 c=15 d=188 X[k]=128 T[i-1]=54 s=1
A=188 B=183 C=143 D=15

I(ref byte a, byte b, byte c, byte d, uint k, ushort s, uint i)
{a = (b + RotateLeft((a + (c ^ (b | ~d)) + X[k] + T[i - 1]), s));}

13) I(ref A, B, C, D, 12, 4, 13)
a=156 b=183 c=143 d=15 X[k]=0 T[i-1]=42 s=4
A=156 B=183 C=143 D=15

14) I(ref D, A, B, C, 13, 8, 14)
a=11 b=156 c=183 d=143 X[k]=0 T[i-1]=99 s=6
A=156 B=183 C=143 D=11

15) I(ref C, D, A, B, 14, 12, 15)
a=10 b=11 c=156 d=183 X[k]=88 T[i-1]=65 s=0
A=156 B=183 C=10 D=11

16) I(ref B, C, D, A, 15, 1, 1)
a=187 b=10 c=11 d=156 X[k]=1 T[i-1]=84 s=2
A=156 B=187 C=10 D=11

A=156 B=187 C=10 D=11
AA=110 BB=249 CC=204 DD=36

(A + AA) A= 10 = a

(B + BB) B= 180 = b4

(C + CC) C= 214 = d6

(D + DD) D= 47 = 2f

Reverse4Bit işleminden sonraki sonuç:

(a-b4-d6-2f)=(a0-4b-6d-f2)

“Merhaba Dünya! Mini-MD5 değerini buluyoruz.” mesajının Mini-MD5 özet değeri “a04b6df2” olarak hesaplanmıştır.

********** Blok Sonu **********

6. SONUÇ

Yapılmış olan bu çalışmada kriptografik hash fonksiyonlarından MD5'in yapısını incelenmiş ve bu yapıdaki adımların daha iyi anlaşılması için küçültülmüş MD5 (Mini-MD5) özet fonksiyonu tasarlanmıştır.

Mini-MD5 ile özet fonksiyonların yapısının daha iyi anlaşılması için algoritmanın çalışması adım adım gösterilmiştir. Sonuç olarak şunu söyleyebiliriz ki; özet fonksiyonların Bölüm 3.2'de bahsedilmiş olan özelliklerinin oluşması için kullanılan (*ilkel fonksiyonlar*, *başlangıç değerleri (initial value)*, *toplam döngü sayısı*, *özet değerin büyüklüğü* ve *blok boyutları*) kıstaslar özet fonksiyonun temelini oluşturmaktadır. Bu adımlarda yapılacak olan değişikliklerle özet fonksiyonların iyileştirilmesinin mümkün olabileceği öngörülmüştür (Wang, Feng, Lai ve Yu, 2004).

Geliştirilen tezin, özet fonksiyonların daha iyi anlaşılması ve tasarım esnasında dikkat edilmesi gereken noktaların kavranmasını açısından faydalı olacağı düşünülmüştür.

KAYNAKLAR

Gimaldi R. P., Discrete and Combinatorial Mathematics - An Applied Introduction, 5th Edition, Pearson, 2004.

Lipschutz S. and Schiller J. J., Schaum's Outline of Introduction to Probability and Statistics, McGraw-Hill, 1998.

Kaliski B., RFC 1319: The MD2 Message-Digest Algorithm, RSA Laboratories, April 1992.

Rivest R. L., RFC 1320: The MD4 Message-Digest Algorithm, Network Working Group, 1992.

Rivest R. L., RFC 1321: The MD5 Message-Digest Algorithm, MIT Laboratory for Computer Science and RSA Data Security Inc., April 1992. Available at: *http://tools.ietf.org/html/rfc1321.*

Soyalıç S., Cryptographic Hash Functions And Its Applications, Yüksek Lisans Tezi, Erciyes Üniversitesi Fen Bilimleri Enstitüsü, Eylül 2005.

Erdoğan M. ve Yılmaz G., Çözümlü Problemlerle Soyut Cebir ve Sayılar Teorisi, Birinci Baskı, Ağustos 2008.

Yalta A. T., Anlamlı Basamaklar Konusu ve Olasılık, Türkiye Bilimler Akademisi Ulusal Açık Ders Malzemeleri, Ekim 2011.

Schneier B., Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition, ISBN: 0471128457, John Wiley & Sons Inc., 1996.

Menezes A. J., Oorschot P.C., Vanstone S.A., Handbook of Applied Cryptography, CRC Press, 1997.

Stevens M., Fast Collision Attack on MD5, International Association for Cryptologic Research, 2006. Avaible at: http://eprint.iacr.org/2006/104.pdf

Wang X., Feng D., Lai X. and Yu H., Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, International Association for Cryptologic Research, 2004. Avaible at: http://eprint.iacr.org/2004/199.pdf

Manuel S., Classification and Generation of Disturbance Vectors for Collision Attacks against SHA1, International Association for Cryptologic Research, 2008. Available at: http://eprint.iacr.org/2008/469.pdf

Preneel B., Bosselaers A. and Dobbertin H., The cryptographic hash function RIPEMD-160, Crypto Bytes, Vol. 3, No. 2, 1997, Pages 9-14.

Available at: ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto3n2.pdf

Mendel F., Pramstaller N., Rechberger C., Kontak M. and Szmidt J., Cryptanalysis of the GOST Hash Function, Lecture Notes in Computer Science, 2008, Volume 5157, Advances in Cryptology – CRYPTO 2008, Pages 162-178.
Available at: *http://www.springerlink.com/content/2514122231284103/*

Naito Y., Sasaki Y., Kunihiro N. and Ohta K., Improved Collision Attack on MD4 with Probability Almost 1, Lecture Notes in Computer Science, 2006, Volume 3935, Information Security and Cryptology - ICISC 2005, Pages 129-145.
Available at: *http://www.springerlink.com/content/v6526284mu858v37/*

Guo J., Ling S., Rechberger C. and Wang H., Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA2, International Association for Cryptologic Research, 2010. Available at: *http://eprint.iacr.org/2010/016.pdf*

Xie T. and Feng D., How to Find Weak Input Diferences for MD5 Collision Attacks, International Association for Cryptologic Research, 2009. Available at: *http://eprint.iacr.org/2009/223.pdf*

Sasaki Y. and Aoki K., Finding Preimages in Full MD5 Faster Than Exhaustive Search, Lecture Notes in Computer Science, 2009, Volume 5479, Advances in Cryptology - EUROCRYPT 2009, Pages 134-152.
Available at: *http://www.springerlink.com/content/d7pm142n58853467*

Wang X., Lai X., FengD., Chen H. and Yu X., Cryptanalysis of the Hash Functions MD4 and RIPEMD, Lecture Notes in Computer Science, 2005, Volume 3494, Advances in Cryptology - EUROCRYPT 2005, Page 551.
Available at: *http://www.springerlink.com/content/n5vrtdha97a2udkx/*

Manuel S. and Peyrin T., Collisions on SHA0 in One Hour, Lecture Notes in Computer Science, 2008, Volume 5086, Fast Software Encryption, Pages 16-35.
Available at: *http://www.springerlink.com/content/3810jp9730369045/*

Guo J., LingS., Rechberger C., and Wang H., Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA2, International Association for Cryptologic Research, 2010. Available at: *http://eprint.iacr.org/2010/016.pdf*

Lamberger M., Mendel F., Rechberger C., Rijmen V. and Schläffer M., Rebound Distinguishers: Results on the Full Whirlpool Compression Function; Lecture Notes in Computer Science, 2009, Volume 5912, Advances in Cryptology – ASIACRYPT 2009, Pages 126-143. Available at:
http://www.springerlink.com/content/9711452444u24861/

Klima V., Tunnels in Hash Functions: MD5 Collisions Within a Minute, International Association for Cryptologic Research, 2006. Available at:
http://eprint.iacr.org/2006/105

Anderson R., The Classification of Hash Functions, The National Science Foundation, 1993.

Mironov I., Hash Functions: From Merkle-Damgård to Shoup, Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology, EUROCRYPT 2001, Pages 166 – 181.

Lei D., Guozhu F., Chao L., Keqin F., and QuL., The Design Principle of Hash Function with Merkle-Damgard Construction, International Association for Cryptologic Research, 2006.
Available at:*http://eprint.iacr.org/2006/135.pdf*

Küçük Ö., Design and Analysis of Cryptographic Hash Functions, PhD Thesis, Katholieke Universiteit Leuven – Faculty of Engineering Arenbergkasteel, B-3001 Heverlee (Belgium), April 2012. Available at:
http://www.cosic.esat.kuleuven.be/publications/thesis-200.pdf

Gauravaram P., Cryptographic Hash Functions: Cryptanalysis, Design and Applications, Information Security Institute Faculty of Information Technology Queensland University of Technology, PhD Thesis, 2007. Available at:*http://eprints.qut.edu.au/16372/1/Praveen_Gauravaram_Thesis.pdf*

Grah J. S., Hash Functions In Cryptography, Department of Computer Science Faculty of Mathematics and Natural Sciences, University of Bergen, 2008.

Koçak O., Design And Analysis Of Hash Functions, Master of Science in Department of Cryptography, Middle East Technical University, Master Thesis, 2009.

ÖZGEÇMİŞ

Cihat ERDOĞAN 1982 yılında Antalya'da doğdu. İlk ve orta öğrenimini Antalya'da tamamladıktan sonra 2001 yılında girdiği Kırgızistan – Türkiye Manas Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nden 2006 yılında mezun oldu. 2006 – 2007 yıllarında vatani görevini gerçekleştirdi. 2008 – 2009 yıllarında özel sektörde çalıştı. 2009 yılının Eylül ayında N.K.Ü. Çorlu Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde Araştırma Görevlisi olarak göreve başladı. 2010 – 2011 eğitim öğretim yılında Trakya Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Ana Bilim Dalı'nda Yüksek Lisans eğitimine başladı. Cihat ERDOĞAN halen N.K.Ü. Çorlu Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde Araştırma Görevlisi olarak çalışmaktadır.