**AÇIK ANAHTARLI ŞİFRELEME VE MESAJ KİMLİK DOĞRULAMA**

**21.1 Güvenli Hash Fonksiyonları**

**Simple Hash Fonksiyonları**

**SHA Güvenli Hash Fonksiyonları**

**SHA-3**

**21.2 HMAC**

**HMAC Tasarım Amaçları**

**HMAC Algoritması**

**HMAC güvenliği**

**21.3 RSA Açık Anahtar Şifreleme Algoritması**

**Algoritmanın Açıklaması**

**RSA'nın Güvenliği**

**21.4 Diffie-Hellman ve Diğer Asimetrik Algoritmalar**

**Diffie-Hellman Anahtar Değişimi**

**Diğer Açık Anahtarlı Şifreleme Algoritmaları**

**21.5 Önerilen Okumalar**

**21.6 Anahtar Terimler, İnceleme Soruları veProblemler**

**Öğrenme Hedefleri**

Bu bölümü okuduktan sonra şunları yapabilmelisiniz:

◆ SHA-1 ve SHA-2'nin çalışmasını anlamak.

◆ Mesaj doğrulama için HMAC kullanımına genel bir bakış sunmak.

◆ RSA algoritmasını tanımlamak.

◆ Diffie-Hellman algoritmasını tanımlamak.

Bu bölüm, Bölüm 2.2'den 2.4'e kadar olan bölümlerde tanıtılan konulara ilişkin teknik ayrıntılar sağlar.

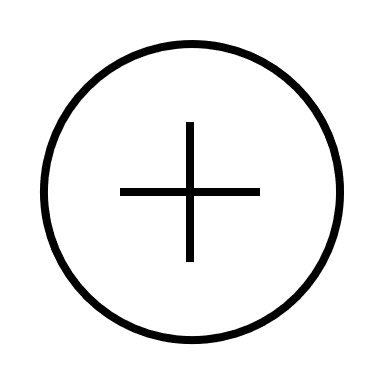
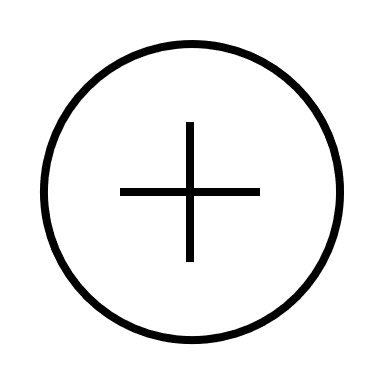
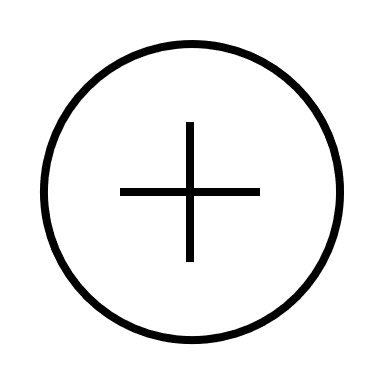
**21.1 Güvenli Hash Fonksiyonları**

Tek yönlü hash fonksiyonu veya güvenli hash fonksiyonu, yalnızca mesaj doğrulamada değil, aynı zamanda dijital imzalarda da önemlidir. Güvenli karma işlevlerin gereksinimleri ve güvenliği Bölüm 2.2'de tartışılmıştır. Burada, belki de en yaygın kullanılan özet işlevleri ailesine odaklanan birkaç özet işlevine bakıyoruz: Güvenli Karma Algoritması (SHA).

**Simple Hash Fonksiyonları**

Tüm hash fonksiyonları aşağıdaki genel prensipleri kullanarak çalışır. Girdi (mesaj, dosya vb.), n-bitlik blokların bir dizisi olarak görülür. Girdi, n-bitlik bir karma işlevi üretmek için yinelemeli bir şekilde her seferinde bir blok işlenir.

En basit hash fonksiyonlarından biri, her bloğun bit-bit exclusive-OR(XOR)'dır. Bu şu şekilde ifade edilebilir:

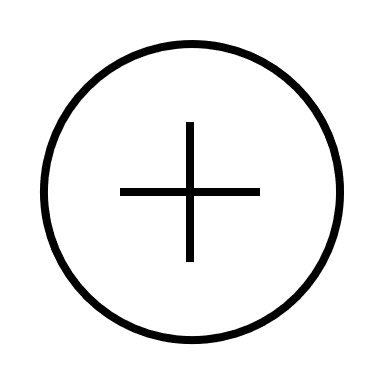
*Ci = bi1 bi2 …bim*

öyle ki;

*Ci*= 1 ≤ i ≤ n olmak üzere hash kodunun n. biti,

*m* = girilen n-bit bloklarının sayısı,

*bij* = j. bloktaki n. bit,

 = XOR işlemi

Şekil 21.1 bu işlemi göstermektedir; her bit konumu için basit bir eşlik üretir ve uzunlamasına artıklık denetimi olarak bilinir. Veri bütünlüğü kontrolü olarak rastgele veriler için oldukça etkilidir. Her n bitlik hash değeri eşit derecede olasıdır. Bu nedenle, bir veri hatasının değişmeyen bir hash değeriyle sonuçlanma olasılığı 2-n 'dir. Daha öngörülebilir biçimde biçimlendirilmiş verilerle, işlev daha az etkilidir. Örneğin, çoğu normal metin dosyasında, her sekizlinin yüksek dereceli biti her zaman sıfırdır. Dolayısıyla, 2-128 'lik bir etkinlik yerine 128 bitlik bir özet değeri kullanılırsa, bu tür verilerdeki özet işlevinin etkinliği 2-112 'dir.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Bit 1 | Bit 2 | . . . | Bit n |
| Blok 1 | *b11* | *b21* | *. . .* | *bn1* |
| Blok 2 | *b12* | *b22* | *. . .* | *bn2* |
| .  .  . | *.*  *.*  *.* | *.*  *.*  *.* | *.*  *.*  *.* | *.*  *.*  *.* |
| Blok m | *b1m* | *b2m* | *. . .* | *bnm* |
| Hash kodu | *C1* | *C2* | *. . .* | *Cn* |

Şekil 21.1 Bitsel XOR Kullanan Basit Hash Fonksiyonu

Sorunları düzeltmenin basit bir yolu, her blok işlendikten sonra karma değer üzerinde 1 bitlik dairesel kaydırma veya döndürme yapmaktır. Yöntem aşağıdaki gibi özetlenebilir:

1. Başlangıçta *n-bi*t karma değerini sıfıra ayarlayın.

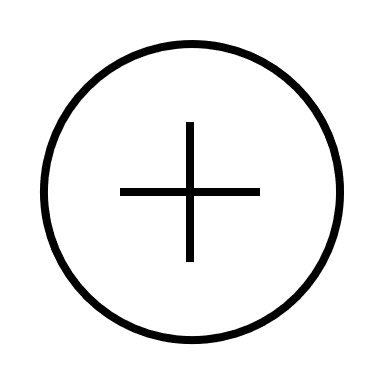
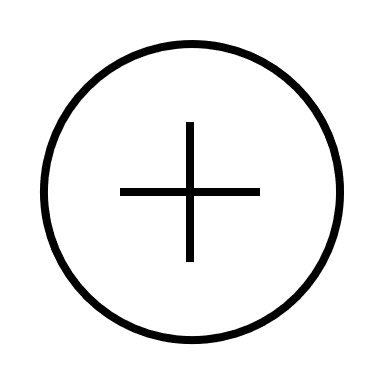
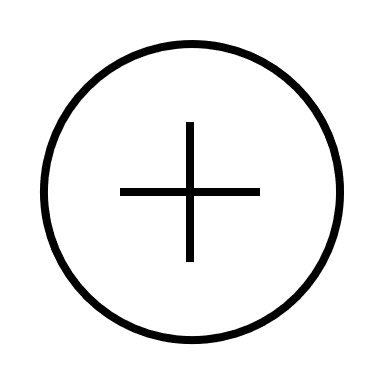
2. Her bir ardışık *n-bit* veri bloğunu aşağıdaki gibi işleyin:

a. Geçerli karma değerini 1 bit sola döndürün.

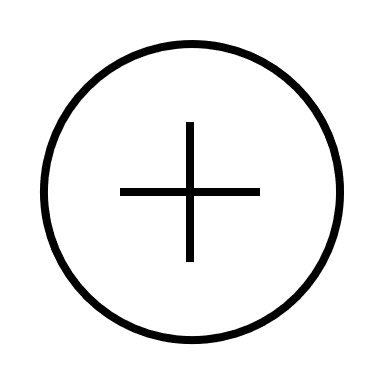
b. Bloğu hash değerine XOR ile yerleştirin.

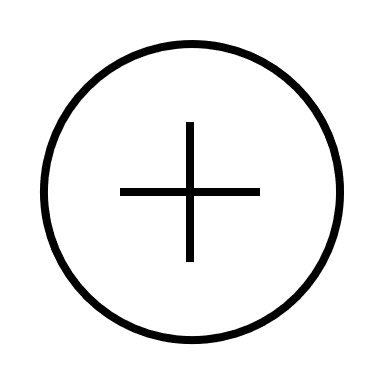
Bu, girdiyi daha eksiksiz bir şekilde "rastgeleleştirme" ve girdide görünen tüm düzenliliklerin üstesinden gelme etkisine sahiptir. İkinci yöntem veri bütünlüğünün iyi bir ölçüsünü sağlasa da, Şekil 2.6a ve b'de olduğu gibi bir düz metin mesajı ile şifrelenmiş bir karma kod kullanıldığında, veri güvenliği için neredeyse hiçbir işe yaramaz. Bir mesaj verildiğinde, o hash kodunu veren yeni bir mesaj üretmek kolay bir meseledir: Sadece istenen alternatif mesajı hazırlayın ve ardından yeni mesajı artı bloğu istenen hash kodunu vermeye zorlayan bir n-bitlik blok ekleyin.

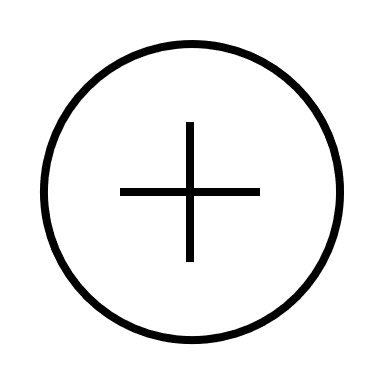
Basit bir XOR veya döndürülmüş XOR (RXOR) yalnızca hash kodu şifrelenince yetersiz olsada, mesaj ve hash kodu şifrelendiğinde böyle basit bir işlevin yararlı olabileceğini hala hissedebilirsiniz. Ama dikkatli olmalıyız. Başlangıçta Ulusal Standartlar Bürosu tarafından önerilen bir teknik, mesajın 64-bit bloklarına uygulanan basit XOR'u ve ardından şifre bloğu zincirleme (CBC) modunu kullanan tüm mesajın bir şifrelemesini kullandı. Şemayı şu şekilde tanımlayabiliriz: Bir dizi 64 bitlik bloklardan oluşan *X1, X2,…, XN*hash kodunu C blok-blok XOR veya tüm bloklar olarak tanımlayın ve hash kodunu son blok olarak ekleyin:

*C = XN+1 = X1  X2 … XN*

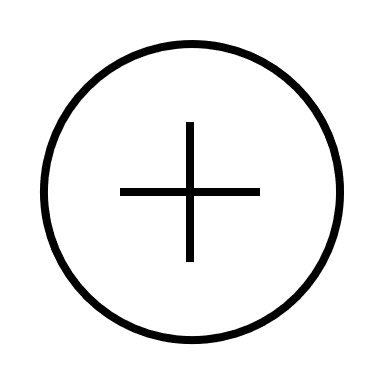
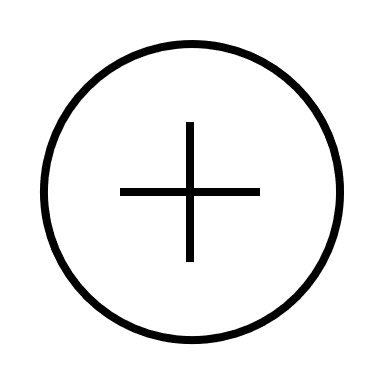
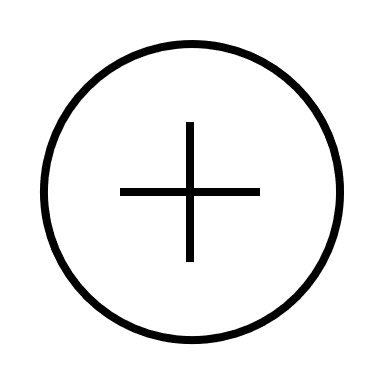
Ardından, *Y1, Y2,…, XN+1*şifreli mesajı üretmek için CBC modunu kullanarak tüm mesajı artı karma kodu şifreleyin. [JUEN85], bu mesajın şifreli metninin, hash kodu tarafından algılanamayacak şekilde değiştirilebileceği birkaç yola işaret eder. Örneğin, CBC'nin tanımına göre (Şekil 20.6):

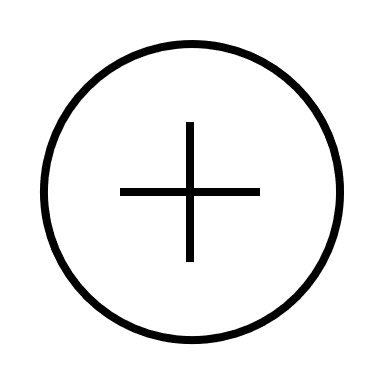
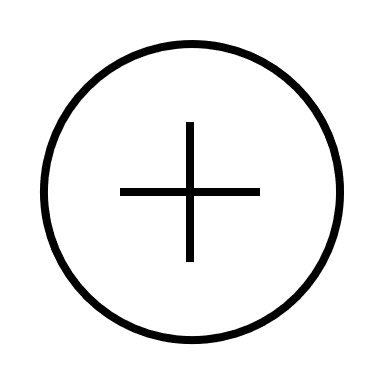
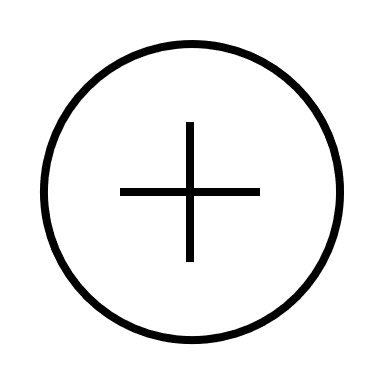
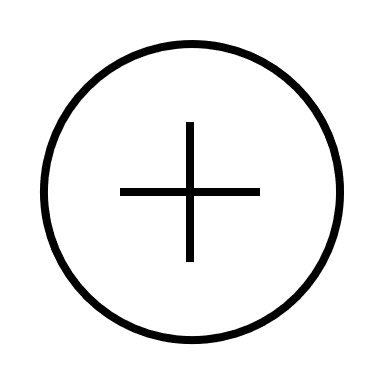
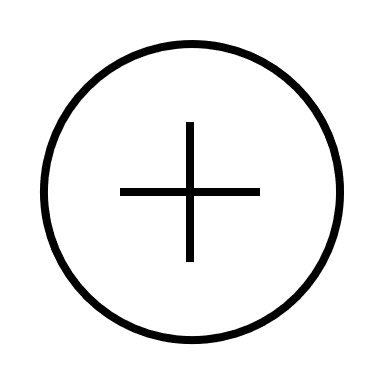
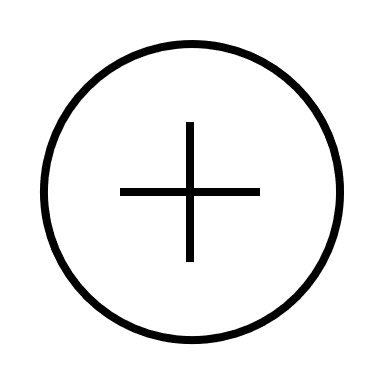
*X1 = IV  D(K,Y1)*

*Xi = Yi-1  D(K,Yi)*

*XN+1 = YN  D(K,YN+1)*

Ama *XN+1*  hash kodu:

*XN+1 = X1  X2 … XN*

*= [IV  D(K,Y1)]  [ Y1  D(K,Y2)] …  [YN-1  D(K,YN)]*

Önceki denklemdeki terimler herhangi bir sırada XOR'lanabildiğinden, şifreli metin bloklarına izin verilirse karma kodun değişmeyeceği sonucu çıkar.

**SHA Güvenli Hash Fonksiyonu**