Fırat Üniversitesi Fen Bil. Dergisi
33(1), 81-91, 2021

Fen ve Mühendislik Çalışmalarındaki Rastgelelik Gereksinimleri için Orta Kare ve SHA3 Algoritmasını Temel Alan Bir Üreteç Tasarımı

Seda Nur YAŞAR[1*], Fatma Ceren DİKİCİ[2], Erkan TANYILDIZI[3], Ebru KARAKÖSE[4]
[1]Ekobilişim, Mühendislik Fakültesi, Fırat Üniversitesi, Elâzığ, Türkiye
[2]Yazılım Mühendisliği, Teknoloji Fakültesi, Fırat Üniversitesi, Elâzığ, Türkiye
[3]Yazılım Mühendisliği, Teknoloji Fakültesi, Fırat Üniversitesi, Elâzığ, Türkiye
[4]Ekobilişim, Mühendislik Fakültesi, Fırat Üniversitesi, Elâzığ, Türkiye
[*1]sedanur.yasar@hotmail.com, [2]cerendikici96@gmail.com, [3]etanyildizi@gmail.com, [4]eozbay@firat.edu.tr

(Geliş/Received: 02/02/2021; Kabul/Accepted: 22/02/2021)

Öz: Bu makalede Hash (Özet) Fonksiyonuna yer verilmiştir. Popüler kriptografik hash algoritmalarından olan SHA-3 Algoritması ile ilgili detaylı bir literatür araştırması yapılmıştır. Makalenin pratik uygulama kısmında ise Middle-Square (orta-kare) metodu ile rastgele sayı üretme işlemi yapılmıştır. C dili kullanılarak yazılan rastgele sayı üreteci komutlarıyla istenilen basamakta rassal sayılar elde edilmiştir. Elde edilen rassal sayılar, Python diliyle rastgele sayı üretecinden alınan giriş değerleri olup SHA-3 uygulaması gerçekleştirilmiştir. Makalenin son kısmında ise SHA3-256 hash fonksiyonu ile elde edilen beş farklı çıktı Ki-Kare Testi ile analiz edilip güvenilirliği hesaplanmıştır. Bu makalenin amacı hash (özet) fonksiyonlarını anlaşılır bir şekilde ortaya koyarak popüler kriptografik hash algoritmalarından olan SHA-3 Algoritmasının güvenilirliğini kanıtlamaktır.

Anahtar kelimeler: bilgi güvenliği, kriptoloji, özet fonksiyonu, Sha-3 algoritması.

A Generator Design Based on the Middle Square and SHA3 Algorithm for the Requirements of Randomness in Science and Engineering Studies

Abstract: The Hash function is included in this article. A detailed literature study was conducted on the SHA-3 Algorithm, one of the popular cryptographic hash algorithms. In the practical application part of the article, the process of generating random numbers using the Middle-Square method was performed. Random numbers were obtained in the desired digit using random number generator commands written using the C language. The resulting random numbers are input values taken from the random number generator in Python language, and SHA-3 was implemented. In the last part of the article, five different outputs obtained by SHA3-256 hash function were analyzed by Chi-Square Test and their reliability was calculated.The aim of this article is to prove the reliability of the SHA-3 algorithm, one of the popular cryptographic hash algorithms, by comprehensively outlining hash functions.

Key words: information security, cryptology, summary function, SHA-3 algorithm

1.Giriş

Bilgi, çağımızın en önemli ögesi haline geldiğinden bu bilgiyi güvende tutma çabaları da giderek daha önem kazanmıştır. Bu sürece bilgi güvenliği ya da *infosec* denilir. Bilgi güvenliği, dijital veya dijital olmayan her türlü bilgiye izin dışında erişimi engellemek, bilgiyi korumak, tehditleri önlemek, yaşanabilecek her türlü problemlerde kayıpları azaltma amacı güden bir dizi uygulamadır. Bilgi güvenliği CIA üçlüsü denen bütünlük, gizlilik ve erişilebilirlik olarak üç temel unsurdan oluşur. Bu üç unsurdan yola çıkarak bilgi güvenliğinin önemini yadsıyabiliriz. Bilginin başkaları tarafından ele geçirilmesi sonucu; kişi-şirket ya da kurumların toplum içindeki itibarlarının zedelenmesi, temel işlev ve programlarının zarar görmesi, hizmette aksamalar, maddi-manevi kayıplar ve zaman kayıplarına uğrama gibi oluşacak çoğaltılabilir daha birçok sebepten ötürü bilgi güvenliği tüm kullanıcılar için gereklidir.

Gizli bilgilerimizi yetkisi olmayan kişilerden korumak amacıyla onu o kişiler için anlamsız olacak şekilde gizlemek isteriz ve bunu bir şifreleme sistemi ile sağlarız. Bu şifreleme yöntemlerinden biri de kriptografidir.

Kriptografi; iletişimde, bilgisayar sistemlerinde, elektronik ticarette ve ortaya çıkan bilgi toplumunda bilgi güvenliğini sağlamak için anahtar bir teknolojidir.[1] Aslında buna sanal kilit de diyebiliriz.Yüzyıllardan beri

* Sorumlu yazar: sedanur.yasar@hotmail.com. Yazarların ORCID Numarası: [1] 0000-0003-3900-3340, [2]0000-0003-1344-4404, [3]0000-0003-2973-9389, [4]0000-0003-1191-6375

gerek ilkel yöntemlerle gerekse teknolojik yöntemlerle yani bilgilerin kriptoloji aracılığıyla gizliliği sağlanıyor.Bu sayede gönderdiğimiz veriye sadece alıcı ulaşabiliyor ancak teknolojinin daha ilkel olduğu İkinci Dünya Savaşı zamanlarında İngiltere ile Almanya arasında olan savaşın seyri, savaş taktiklerini içeren mesajların deşifre edilmesiyle değişmiştir. Bilginin neden daha güvende olması gerektiğinin sebebi de budur. Bu yüzden zaman ilerledikçe gelişen teknolojiye ayak uydurabilecek kritptoanaliz çalışmaları gün geçtikçe önem kazanmakta ve daha güvenli hale gelmektedir. Buna istinaden bilgi güvenliği için kriptoloji bilimi en iyi şifreleme sistemlerinden biridir.

Kriptoloji bilimi bilgi güvenliğini sağlayabilmek için bazı özet fonksiyonu da denen hash fonksiyonlarından yararlanmaktadır. Bu kriptografik hash(özet) fonksiyonlar modern kriptografide oldukça önemli bir yere sahiptir. Bu fonksiyonlar bir girdiyi alır ve aldıkları girdinin boyutundan bağımsız bir çıktı üretirler. Yani her girdinin özel bir çıktısı vardır. Öyleyse kriptografik hash fonksiyonu; bilgi güvenliği ve uygulamalarını daha kullanışlı hale getirmek için hash fonksiyonları dahilinde ilave güvenlik özellikleri olan bir özet fonksiyonudur diyebiliriz.

Bu makalede amacımız; kritografik hash(özet) fonksiyonlarından olan SHA-3 algoritmasının kapsamlı şekilde incelenmesi ve çalışma prensibinin anlatılıp uygulama üzerinde gösterilmesidir.

2.Hash (Özet) Fonksiyonu

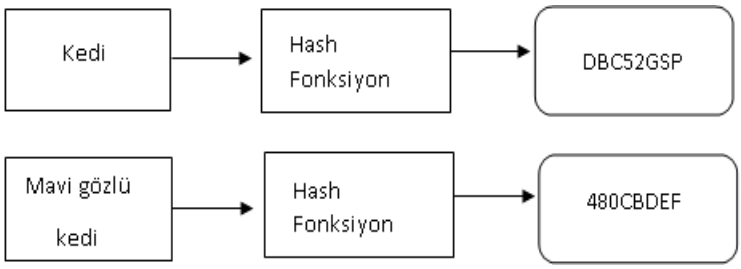
Hash(özet)fonksiyonu her boyutta olabilecek girdiyi (örn. resim ya da metin vb.) boyutlarından bağımsız olarak sabit boyutta çıktılara dönüştürür. Teorik olarak ifadesi şu şekildedir: *x* girdi/mesaj, *h* özet(hash) değeri ve *H* özet fonksiyonudur. Bu durumda elde edilen sabit boyutlu çıktı *h=H(x)*'tir.

Özet fonksiyonlarının çıktı boyutları 160 ile 512 bit arasında değişir. Girdi boyutu ise ne kadar büyük olursa olsun sabit bir çıktı boyutu elde edildiği için göndermek istediğimiz verinin boyutu da tasarruflu olmuş olur.Özet fonksiyonunun en önemli özelliklerinden biri aynı girdi değerine karşılık aynı çıktıyı -bu çıktılara mesaj özeti denir.- oluşturur ancak iki farklı girdi için aynı çıktı sonucunu asla vermemesidir.Yani bir çıktı sadece bir girdi değerine özeldir. Eğer aynı çıktı değerine sahip iki farklı girdi değeri varsa bu hash fonksiyonu kırılmış diyerek basitçe ifade edebiliriz. Bu, kırılmış bir özet fonksiyonu kriptografik olarak yeterli değil anlamına gelir. Daha bilimsel bir dille iki farklı girdiye aynı çıktı üretilmesine "collision/çarpışma" denilir. Yani korumaya çalıştığımız iki girdi değerimize collision saldırısı ile ulaşılabilir.

Karma işlev tarafından döndürülen değerlere karma değerler/işlevler veya karmalar denir. İşlevler, karma tabloyu indekslemek için kullanılır. Karmanın temelinde, bir karma kod oluşturmak amacıyla iki sabit boyutlu veri bloğu ve matematiksel bir işlev vardır. Bu karma işlev, karma algoritmanın bir parçasıdır. Verilerin boyutu, algoritmaya bağlıdır. Genelde 128-512 bit arasındadır.

Karma işlevler, etkinlikleri için uygun olasılık dağılımları oluşturmaya dayanır ve erişim süresini neredeyse sabit hale getirir. Yüksek tablo yükleme faktörleri, patolojik anahtar kümeleri ve kötü tasarlanmış karma işlevler, erişim sürelerinin tablodaki öğe sayısında doğrusal yaklaşmasına neden olabilir. Karma işlevler, en kötü durum performansını, yüksek tablo yükleme faktörleri altında iyi performansı ve özel durumlarda, anahtarların karma kodlara mükemmel (çarpışmasız) eşlenmesini sağlayacak şekilde tasarlanabilir. Uygulama, pariteyi koruyan bit işlemlerine (XOR ve ADD), çarpmaya veya bölmeye dayanır. Hızlı arama fonksiyonuna gerekli bir ek, bağlantılı listeler gibi yardımcı bir veri yapısını veya boş bir yuvayı bulmak için tablonun sistematik incelemesini kullanan eski bir çarpışma çözme yöntemidir. [2]

Hash fonksiyonu tek yönlüdür. Şekil 1'de hash fonksiyonunun açıklamalarla nasıl tek yönlü olduğu anlatılmaya çalışılmıştır.



Şekil 1. Hash (Özet) Fonksiyonu.

Yukarıdaki şekil temsili ve örnek amaçlıdır. Bu şekilde ''Kedi'' kelimesini girdi olarak verdiğimizde alınan çıktı DBC52GSP olmuştur. Diğer girdimiz ise ''Mavi gözlü kedi''dir ve görüldüğü üzere üç kelime olmasına

rağmen çıktı boyutu diğeriyle aynıdır. Buradan hash fonksiyonunun her ne uzunlukta veri olursa olsun sabit boyutta çıktı ürettiğini de görmüş olduk. Yine şekildeki örnek çıktılardan birini ele alacak olursak, DBC52GSP çıktısından asla girdideki ‘’Kedi’’ verimizi elde edemeyiz. Çünkü bir hash fonksiyonundan veriye ulaşmak gibi bir durum söz konusu değildir. Tek yönlü algoritma olmasından ötürü veriye dönüş sağlanamaz.

2.1.Hash (Özet) Fonksiyonu’nun Özellikleri

Bu fonksiyonun özelliklerini beş madde ile listeyebiliriz:

- Özet fonksiyona giren bir veri her zaman aynı sonucu verir.
- Özet fonksiyonları hızlı bir şekilde verinin özetini hesaplar. Girdi üzerindeki yapılan en ufak bir değer değişimi bile (örn. tek bir bitlik değişim) tamamen farklı çıktıya sahip özetle sonuçlanacaktır.
- Özet fonksiyonun çıktısından girdi verinin ne olduğunu bulmak polinom zamanda mümkün olmamalıdır. (tek yönlü fonksiyon/ön görümtü direnci).
- Özet algoritmaları da çarpışmaya dirençli olarak (ikincil ön-imaj (second preimage)) tasarlanmalıdır: aynı çıktıyı üreten iki veya daha fazla girdi bulmak polinom zamanda bulmak mümkün olmamalıdır. [3] İdeal bir hash fonksiyonu bu özelliklerin tümünü sağlamalıdır.

Bu kısımda kriptografi, kriptoloji ve kriptoanaliz kavramlarından kısaca bahsedilecektir.

Kriptografi: Kodunu yalnız alıcısının açabileceği şekilde düzenlenen mesajların içeriğini gizleme ve mesajı tekrar eski haline dönüştürme prensipleri ve yöntemlerini içeren gizli dönüşümler bilimidir.[4]

Kriptoloji: Şifre bilimidir. Gönderilmek istenen verinin şifreleme algoritmalarıyla şifrelenip alıcıya ulaştırılmasıdır.

Kriptoanaliz: Kod kırma olarak da adlandırılır. Bilgi, veri, anahtar vb. kaynakları olmadan şifrelenmiş veriye yasa dışı ulaşmayı, deşifre etmeyi amaçlayan analizdir.

2.2.Kriptografik Hash Fonksiyonu

Kriptografik hash (Kriptografik Özet) fonksiyonları, şifrelemede kullanılan matematiksel işlev olarak tanımlanır. Özellikleri hash fonksiyonları ile aynıdır bu fonksiyonlara güvenlik özelliklerini dahil ederek iletilecek verinin çözümünün daha zor olmasını sağlar.

2.2.1.Popüler Kriptografik Hash Algoritmaları

A) Mesaj Özütü (MD)

128 bitlik bir hash fonksiyonu olan MD ailesi, MD2, MD4, MD5 ve MD6 karma işlevlerinden ibarettir. MD5, son yıllarda kullanılan en popüler hash işlevi olarak bilinmektedir. Profesör Ron Rivest tarafından 1991 yılında geliştirilmiştir. MD5 algoritması, herhangi bir uzunlukta metni, şifreyi, mesajı ya da dosyayı 128 bit parmak izine kodlar ve bunu hash fonksiyonlarına dayalı olarak yapar. Yani verilerin kendine has “parmak izi” nin oluşturan bir “hash” algoritmasıdır.

MD5 Hash Fonksiyonunun Özellikleri

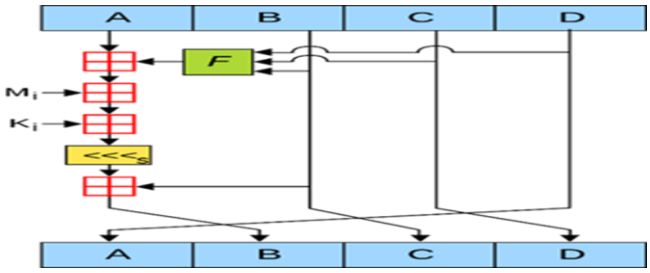
- MD5 algoritması da tek yönlüdür. Şifreleme yapar ama şifre çözüm işlemi yapılmasına olanak sağlamaz.
- MD5 algoritması, üzerinde işlem yaptığı dosyada herhangi bir değişiklik varsa bunu tespit eder. Bir değişiklik varsa yeni dosyanın çıkışı ilkinden farklı olacaktır.
- MD5, MD4’ten daha yavaştır. Bununla birlikte MD4’e göre şifreleme sistemi daha karmaşık ve çözülmesi daha zordur.
- Boyutu ne olursa olsun girişi yapılan dosyanın çıkışı 128-bit, 32 karakter,16'lık sayı sisteminde bir dizi olarak elde edilir.

Algoritması

- MD5 değişken uzunluktaki bir mesajı 128 bitlik sabit uzunlukta bir çıktıya dönüştürür.

- Daha sonra girişteki veri, 512 bitlik bloklara ayrılır (16 tane 32 bit). Eğer veri 512 ve katlarından biri olmasaydı ekleme yapılırdı(padding). Ekleme yapılırken; verinin sonuna bir bit 1 eklenir sonra 512'nin en yakın katından 64 bit eksik olma kaydıyla geri kalan kısma 0 eklenir.
- Geriye kalan 64 bite de orijinal mesajın uzunluğu mod 264 'de yazılır.
- Ana MD5 algoritması, A, B, C ve D olarak adlandırılan dört adet 32 bitlik kelimeye ayrılmış 128 bitlik parçalar üzerinde çalışır. Bunlar belirli sabit değerlerle başlatılır.Daha sonra ana algoritma, her 512-bit ileti bloğunu durumunu(128 bit) değiştirmek için kullanır.Bir mesaj bloğunun işlenmesi, tur denilen dört benzer aşamadan oluşur. Her tur, doğrusal olmayan bir fonksiyon, modüler toplama işlemi ve bit bazında sola kaydırma işlemlerinden oluşur. Toplamda 16 tur vardır. 4 olası F fonksiyonu vardır; her turda farklı bir fonksiyon kullanılır. [5]

Aşağıdaki şekil çalışma sistemini açıklamaktadır:



Şekil 2. MD5 çalışma diyagramı.

MD5 Kullanım Alanları

A. Bir verinin doğru transfer edilip edilmediğini sorgulamada
B. Bir verinin değiştirilip değiştirilmediğini kontrol etmede
C. Asimetrik şifrelemede(public-key)
D. Kimlik doğrulamasında
E. VPN, SSL gibi uygulamalarda MD5 algoritması tercih edilir.

Md5 Hash Algoritmasının Desteklediği Veri Formatları

a) **Dosya:** Binary/ikili mod (müzik/music, audio, ses/sound, video, resim/image, ikon/icon, metin/text, sıkıştırma/compression, vs., ile uzantılar/extensions: .mp3, .wav, .avi, .mpg, midi, .mov, .dvd, .ram, .zip, .rar, .ico, .pif, .pic, .tif, .tiff, .txt, .doc, .docx, .pdf, .wps, .dat, .dll, .hex, .bin, .iso, .cpp, .dss, .par, .pps, .cue, .ram, .md5, .sfv vs.) Genelde 15 GB'a kadar sorunsuz çalışabilirler.
b) **Metin:** Herhangi bir metin, karakter, çift veya tek girilebilir.
c) **Hex:** 0-9 ve a-f arasında çift olarak girilmesi gerekir.[6]

Saldırı Metotları

a) Straight/Doğru
b) Combination/Kombinasyon
c) Toggle-Case/Geçişli Durum
d) Brute-Force/Kaba Kuvvet
e) Permutation/Permitasyon
f) Table-Lookup/Tablo Arama
g) Rainbow Table/Gökkuşağı Tablosu
h) Dictionary/Sözlük [6]

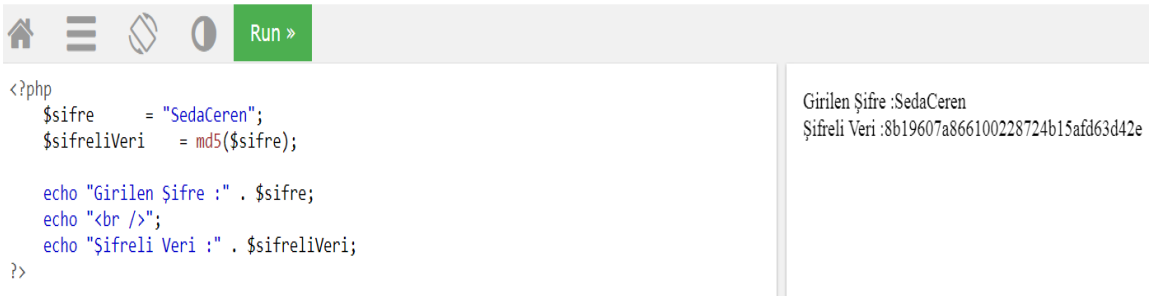
MD5 Saldırısı

MD5 ile şifrelenmiş bir bilgiyi ya da veriyi çözümlemek imkânsız denecek kadar zordur. Ancak Brute-Force(Kaba-Kuvvet) saldırıları ile şifreyi çözümleme ihtimali vardır.Brute-Force saldırısında deneme yanılma

yöntemi kullanıldığı için çözümleme çok uzun zaman almaktadır.Buna rağmen 2004'te MD5CRK Projesi gerçekleştirilmiş ve MD5'e saldırı düzenlenmiştir.Bu saldırı yalnızca bir saat başarılı olabilmiştir.

PHP MD5 Fonksiyonu

Tek taraflı şifreleme algoritmasıdır. Şifrelenecek metin, MD5 fonksiyonu tarafından 32 haneli şifrelenmiş bir veriye dönüşecektir. Aşağıdaki görselde php editöründe uyguladığımız basit bir MD5 şifreleme örneği görmektesiniz.



Şekil 3. MD5 Uygulaması.

Md5 şifreleme mantığı

- Kullanıcının kayıt sırasında şifresi alınır.
- Bu şifre MD5 ile hashlenip veri tabanına yazılır.
- Kullanıcı sisteme girmek istediğinde şifresini girer.
- Girilen şifre MD5'e çevrilip ve veri tabanındaki şifre karşılaştırılır.
- Eğer MD5 karşılaştırma kodları aynı ise sisteme kullanıcı giriş yapabilir.

B) RIPEMD

Kriptografik hash fonksiyonlarından olan RIPEMD, 1992 yılında geliştirilmiş bir algoritmadır. RIPEMD'inin RIPEMD, RIPEMD-128, RIPEMD-160, RIPEMD-256 ve RIPEMD-320 olmak üzere beş çeşidi vardır. İlk geliştirilen RIPEMD ve sonrasında 1996 yıllarında kullanıma sunulan RIPEMD-128 hash algoritmaları güvenli kabul edilmemiştir.Çünkü tasarımlarında bazı zayıflıklar tespit edilmiştir. RIPEMD 256 ve 320 bit sürümleri ise RIPEMD-128 ve 160 bit ile güvenlik düzeyleri aynı olsa da daha uzun hash sonuçları için tasarlanmıştır. Bahsi geçen RIPEMD ailesinin daha çok kripto para birimlerinde kullanımı yaygındır.

Ripemd-160

RIPEMD ailesi içerisinde en popüler olanı RIPEMD-160 fonksiyonudur. RIPEMD-160 hash algoritmasının tasarlanma amaçlarından biri de MD4 ve MD5'in yerine kullanılabilir olmasıdır.Geniş kapsamlı olan Ripemd-160 hash fonksiyonu 21 kelimelik girdiyi alır.Bu girdinin 5 kelimesi zincir değişkenidir ve kalan 16 kelimesi mesajdır. RIPEMD-160 hash fonksiyonu, 21 kelimelik girdiyi 5 kelimelik çıktı haline dönüştürür.

C) SHA-3 ALGORİTMASI

SHA-3, kriptografik Keccak ailesinin bir alt kümesi olup Guido Bertoni, Joan Daemen, Michaël Peeters ve Gilles Van Assche tarafından tasarlanan güvenli bir hash algoritmasıdır. 5 Ağustos 2015 tarihinde NIST tarafından yayınlanmıştır. SHA-3 algoritması, bit değerlerine göre 128, 224, 256, 384 veya 512 hash değerlerine sahip altı hash fonksiyonundan meydana gelmiştir.

- SHA3-224
- SHA3-256
- SHA3-384

- SHA3-512
- SHAKE128
- SHAKE256

SHA-3 algoritması tasarlanmadan önce kendisinden daha önce tasarlanmış olan SHA-2, SHA-1 ve SHA-0 algoritmalarının kullanımı söz konusudur. İlk olarak SHA-0 algoritması 1993 yılında tasarlanmıştır. Bulunan güvenlik açığı nedeniyle 1995 yılında SHA-1 algoritması olarak geliştirilmiştir. SHA-1 algoritması güvenlik amacıyla uygulamalarda yaygın olarak kullanılmıştır. Bunlardan en çok bilineni e-imza uygulamalarıdır. Ülkemizde 2013 yılına kadar elektronik imzalar SHA-1 algoritmasına göre şifrelenmekteydi. Bu durum 30.01.2013 tarihinde çıkan bir tebliğ ile yerini SHA-2 algoritmasına bıraktı.

SHA-1 algoritmaları yapı olarak MD5 özet fonksiyonuna benzerdir. Güvenli olarak kabul edilmesine rağmen 2001 yılında bulunan güvenlik açığıyla birlikte NIST tarafından SHA-2 algoritması yayınlanmıştır. SHA-2 algoritması SHA-1 algoritmasından farklı olarak çeşitli boyutlarda özet fonksiyonlarının oluşturulmasına izin vermiştir. SHA-2 algoritmaları için önemli bir güvenlik açığı bulunmadığından saldırı düzenlenmemiştir. Ancak NIST 2006 yılında yeni bir hash algoritmasına ihtiyaç duymuştur. NIST yeni bir hash algoritmasına ihtiyaç duyarak SHA-2 algoritmasının yerini alması amaçlanmamış olup alternatif ve farklı bir hash algoritmasının olmasını istemiştir. Bu nedenle NIST tarafından, 2007'nin ilk çeyreğinde başlayan halka açık bir yarışma düzenlendi. Yarışma üç tur olup her turda aday algoritmalar bazı değerlendirme kriterlerine tabi tutuldu.Bu kriterler; güvenlik,maliyet ve performans,algoritma ve uygulama özellikleri olmak üzere üç geniş kategoriye ayrıldı.Son turda bu kriterlere ek olarak finalistlerin donanım özellikleri de dikkate alınmış ve eleme süreci başlamıştır.Yarışmanın birinci turunda 51 adaydan 14'ü ikinci tura kalmış ve ikinci turdaki bazı adaylar final turu için seçilirse birkaç ince ayar ve iyileştirmeler yapacaklarını duyurmuşlardır. Ancak NIST, finalistlerin seçiminde bu değişiklikleri dikkate almamıştır. Derinlemesine değerlendirilen adaylar içerisinden 5 aday algoritma finale kalmıştır.

Finalistler

- Keccak
- Skein
- JH
- Blake
- Grostl

Yarışmanın kazananı Keccak algoritmasıdır. Keccak bütün analiz ve değerlendirme kriterlerinden rakiplerine karşı üstünlük sağlamış ve yeni SHA-3 unvanını almıştır.

KECCAK(SHA-3) Algoritmasının Çalışması

Keccak (SHA-3) algoritmasının çalışması dört adımda incelenir:

a) Padding-Dolgu
b) State-Durum Boyutu
c) The Absorb Function-Emme İşlevi
d) The Squeeze Function-Sıkıştırma İşlevi

a) Padding-Dolgu

Mesaj özetlenmeden önce giriş verisinin standart boyutu incelenir. Keccak algoritması durum boyutunun nasıl hesaplanacağını şu şekilde belirler:

- $b = 25 \times 2^l$; b = durum boyutu
- l değeri = {0, 1, 2, 3, 4, 5, 6}
- b'nin değeri = {25, 50, 100, 200, 400, 800, 1600} [7]

Sha-3, 1 değerini 6 kabul eder ve 1600 bitlik durum boyutu elde edilir. Durum boyutunun en büyük değeri almasının sebebi güvenliğin değerle orantılı olarak artmasıdır. Sonraki adımda kaç tur hesaplama yapılacağı aşağıdaki gibi bulunur:

- Tur= 12 + 2 xl
- Tur= 12 + 12; l = 6
- Tur= 24 olarak; Toplamda 24 tur olacaktır. [7]

Bu adımın sonunda durum boyutu 1600 bit ve hesaplama tur sayısının değeri de 24 olmuştur. Dolgu işleminde giriş verisinin (karma işlev) uzunluğuna bağlı olarak mesaja 1,0 değerlikli bitler eklenmesi gerekir. Dolgu işlemi r'nin tam katı olacak şekilde yapılır. Dolgu bitleri $P_{0...n+1}$ olarak adlandırılır.

Type	**Output Length**	**Rate (r)**	**Capacity (c)**
SHA3-224	224	1152	448
SHA3-256	256	1088	512
SHA3-384	384	832	768
SHA3-512	512	576	1024 [7]

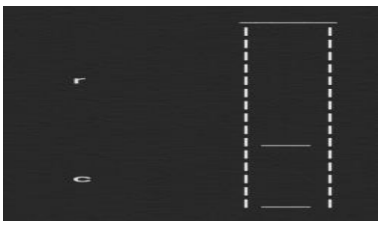
Elde edilen dolgunun ilk ve son bitlerinin değeri 1, aradaki bitler ise 0 olur. Dolgu işleminden sonra mesaj n parçalara ayrılmış olur.

Matematiksel ifadesi şu şekildedir:

- p = nxr
- p = doldurma sonrası mesaj uzunluğu
- n = 'p' yi böldüğümüz medyanın sayısı
- r = oran uzunluğu [7]

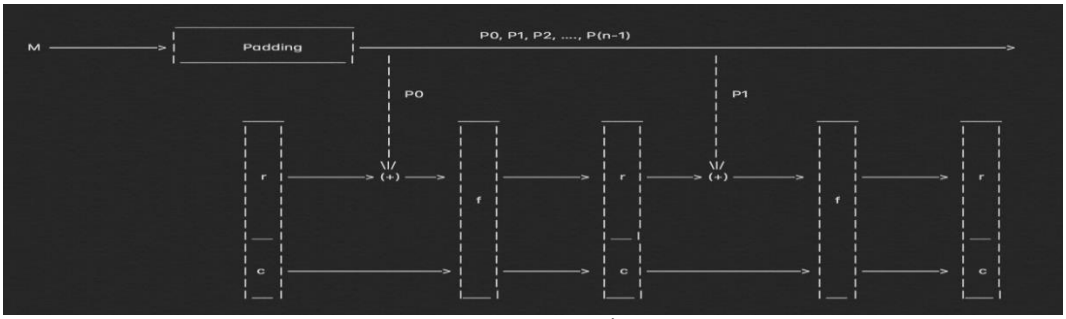
b) State-Durum Boyutu

Aşağıdaki görseldeki 'r' karma algoritma oranı, 'c' ise karma algoritmanın kapasitesidir. r ve c'nin toplamı durum boyutur. Bu toplam 1600'ü aşmamalıdır. Karma algoritmanın değeri değiştikçe bu değer de değişecektir.



Şekil 4. Durum Boyutu.

c) The Absorb Function-Emme İşlevi

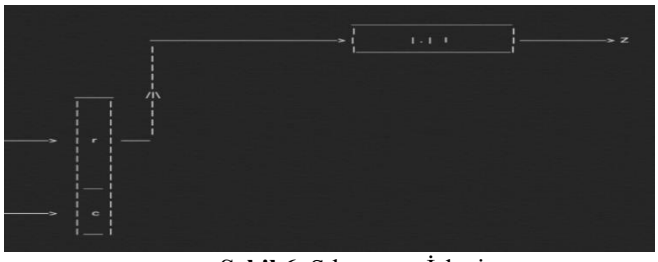


Şekil 5. Emme İşlevi.

SHA-3 algoritmasının en temel aşamaları yukarıda sıraladığımız son iki adım olan absorbe ve sıkıştırma işlevidir.Bu çalışma adımında n parçaya ayrılan mesajın tüm değerleri çıktıyı oluşturmak için tüketme işleminden

geçer.Bu tüketme işleminde amaç r uzunluğunda doldurulan mesajları beslemektir.Bu işleme P_0 ve r arasında mod hesabı ile başlanılır.”r”nin başlangıç değerinin hepsi 0 bit olarak alınır.Mod işlemi gerçekleştikten sonra oluşan değer, gerçek absorbe işlevinin başladığı noktaya aktarılır.Bu işlevin içinde bulunan 5 işlem seti tur sayısı (24) kadar tekrarlanır.1600 bit elde edilir. Tüm tur işlemleri sonlandırıldığında r ve c bitleri ayrılır ve tekrar mod işleminden geçerek fonksiyon baştan başlatılır.

d) The Squeeze Function-Sıkıştırma İşlevi



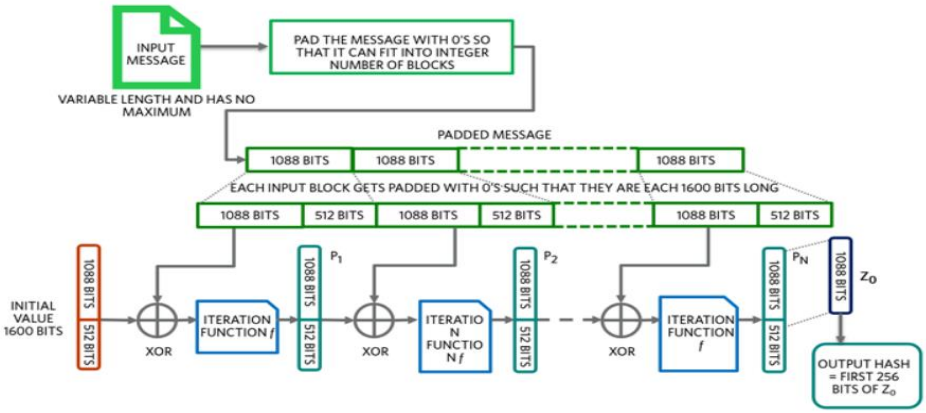
Şekil 6. Sıkıştırma İşlevi.

Absorbe işleminde öğrenimi gerçekleştirilen karma işlevin çıktısının oluşturulduğu adımdır. Hash hesaplaması yapılır. Absorbe sonucu 1600 bit uzunlukta çıktı elde edilmiştir. Bu çıktıyı hash değerine bağlı olarak r ve c bitlerine ayırma işlemi yapılır. Bu adımın sonunda çıktıya ulaşılır.

Çıktı için r değerinden ilk birkaç bit çıkarılır. Bu çıkarma işleminden elde edilen değer tüm mesajın karması olur. Buna örnek olarak:

SHA-3 algoritması 256 bit değerinde olsaydı 1088 bitten ilk 256 biti çıkarılırdı. Bütün bu işlemlerden sonra çıktı elde edilmiş olur. SHA3 algoritmasının çalışma prensibini şu şekilde özetleyebiliriz:

Aşağıdaki diyagram SHA3-256 işlevinin aşamalarını göstermektedir. Diyagramdaki yineleme işlevi, 1600 bitlik veriyi alır, onu belirli bir algoritma kullanarak 24 devir permütasyondan geçirir ve ardından bunu 1600 bitlik bir blok haline getirir. Sonra emme aşamasına geçilir. Bu, emme aşaması tamamlanana kadar devam eder. Emme aşaması tamamlandığında, son 1600 bitlik blok, sıkıştırma işleminden geçirilir. Bu durumda, SHA3-256 çıktı karma uzunluğu 1088 bitten az olduğundan, sıkıştırma fazının herhangi bir yineleme fonksiyonuna ihtiyacı yoktur. Son aşamadaki ilk 256 bit, çıktı hash'idir. Örneğin gerekli karma uzunluğu 2500 bit olsaydı, istenen uzunluk karmasını elde etmek için yineleme işlevinin üç örneğine daha ihtiyaç duyulacaktı.



Şekil 7. Sha3 Çalışma Diyagramı.

3. Pratik uygulama

Bu kısımda C dilinde yazmış olduğumuz rastgele sayı üreteci komutlarıyla istenilen basamakta rassal sayılar elde edilmiş ve Python diliyle rastgele sayı üretecinden alınan giriş değerlerinin SHA-3 uygulaması gerçekleştirilmiştir. Rastgele sayı üretme işleminde benzersiz anahtarların üretildiği bir hashing metodu da denilen Middle-square(orta-kare) metotu kullanıldı. Bu metodun kullanılma amacı; rastgele görünen sayılar dizisi oluşturmaktır. Bu metotda; n basamaklı sözde rassal sayılar oluşturmak için n basamaklı bir değer baz alınır, buna

tohum değerleri de denilir. Tohumun kare sayısının orta terimini alarak çalışır. 2n basamaklı bir sayı ürettikten sonra sayının karesinden belirli basamakta sayılar çıkarılır. Yeniden yeni sayı elde edilir. Neticede 2n'den az basamak oluştuysa, bunu dengelemek için başına sıfırlar eklenir. İşlem tekrarlanır ve daha fazla değer elde etmek için döndürülür. Bu şekilde kendi kendini tekrar eden bir zincirle karşılaşana kadar birden fazla rastgele sayı elde edilmiş olur. Metot, çift uzunluktaki tohumlar için doğru çalıştığından buna dikkat edilerek tohum değerleri belirlenmiştir.

C++ kodu ve rassal sayılar çıktı ekranı:

```cpp
#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;
int a[] = { 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000 };
int middleSquareNumber(int numb, int dig)
{
    int sqn = numb * numb, sonraki_sayi = 0;
    int trim = (dig / 2);
    sqn = sqn / a[trim];
    for (int i = 0; i < dig; i++)
    {
        sonraki_sayi += (sqn % (a[trim])) * (a[i]);
        sqn = sqn / 10;
    }
    return sonraki_sayi ;
}
int main(int argc, char **argv)
{
    cout << "Kac basamakli sayi istiyorsaniz degerini giriniz: ";
    int n;
    cin >> n;
    int start = 1, end = 1;

    start = a[n - 1];
    end = a[n];
    int number = ((rand()) % (end - start)) + start;
    cout << "Rastgele Sayilar:\n" << number << ", ";
    for (int i = 1; i < n; i++)
    {
        number = middleSquareNumber(number, n);
        cout << number << ", ";
    }
    cout << "...";
}
```

```
Kac basamakli sayi istiyorsaniz degerini giriniz: 6
Rastgele Sayilar:
100041, 3254727, 4575382, 1349485, 130861, -165727, ...
--------------------------------
```

Sonraki adımda, SHA ailesine en son kabul edilen SHA3 hash algoritmasının güvenilirliğini test etmek amacıyla makalede üzerinde çalışılmıştır. Hash işlemi için üretilen rassal sayılar Python uygulamasına dahil edilmiş ve SHA3 uygulaması gerçekleştirilmiştir.

```python
# -*- coding: utf-8 -*-

import hashlib


str = "100041, 3254727, 4575382, 1349485, 130861, -165727"

# encode the string
encoded_str = str.encode()

# create sha3 hash objects initialized with the encoded string
obj_sha3_224 = hashlib.sha3_224(encoded_str)    # SHA3-224
obj_sha3_256 = hashlib.sha3_256(encoded_str)    # SHA3-256
obj_sha3_384 = hashlib.sha3_384(encoded_str)    # SHA3-384
obj_sha3_512 = hashlib.sha3_512(encoded_str)    # SHA3-512

# print in hexadecimal
print("\nSHA3-224 Hash: ", obj_sha3_224.hexdigest())
print("\nSHA3-256 Hash: ", obj_sha3_256.hexdigest())
print("\nSHA3-384 Hash: ", obj_sha3_384.hexdigest())
print("\nSHA3-512 Hash: ", obj_sha3_512.hexdigest())
```

SHA3 hash algoritmasının çalışma kodu şu şekilde açıklanabilir:

- Import komutu ile hashlib kütüphanesi içe aktarıldı.
- Bu sayede nesnenin karma değerinin döndürülmesi için ön hazırlık yapılmış oldu. Karma değer, sözlük anahtarlarını hızlı bir şekilde karşılaştırmak için kullanılan tam sayılardır.
- Bilgisayardaki bütün girdi ve çıktılar birer strgingtir. Yani metindir. Str komutu ile çağrıldı ve rastgele sayılar alındı.
- Str.encode komutu ile str stringi byte karakterlerine dönüştürüldü.
- Obj_sha3 satırlarında ise kodlanmış dizeyle başlatılan sha3 karma nesneleri oluşturuldu.
- Print satırlarındaki hexdigest komutu ile şifrelenen rastgele sayılar heksadecimal cinsinden ekrana yazdırıldı.

Pyhton çalışma ekranında elde edilen hash sonuçları:

```
SHA3-224 Hash:  842bfeadbd2ef9f1660d8e3eb026466e1f5f62a859f0b7b041f6b402

SHA3-256 Hash:  28f05ccb9ed4e25d3519547d1293a270aad02740424caba7f97ad91bb11f9fe6

SHA3-384 Hash:
1e8e7a1126d4da9d4955483d1a56fa1335ae0c0fcc45c367fa7e3d2f3520dfc9d51eb6d3629dbe29cf
01b46aafaca5ed

SHA3-512 Hash:
3f0a33ec5532f142dd8920a0a056a57c4bf934bd148e0ed6f80c4433c7ee1b53f0c165e69c845dfe96
3333755e2be0cd149223087023cbe64d59a2d62a6239f9
```

4.Analiz Sonuçları

Makalenin bu kısmında SHA3-256 hash fonksiyonu ile elde edilen beş farklı çıktı Ki-Kare Testi ile analiz edilip güvenilirliği hesaplanmıştır. Başlangıçta p=0,05 kabul edilerek analiz sonucunda bulunan p aralığı başlangıçtaki p değeriyle karşılaştırılmıştır.

Ki-Kare Yöntemi adım adım şu şekilde uygulanmıştır:

- Gözlenen değerler tabloya kaydedildi. Tabloda her satıra r, her kolona c denilir.
- Beklenen değer:$\frac{Satır\ toplamı * Kolon(sütun) toplamı}{Tüm\ toplam}$ formülü ile her hücre için tek tek hesaplandı.
- $\frac{Gözlenen - Beklenen^2}{Beklenen}$ = X^2 'dir ve tüm satırlara bu işlem uygulanıp bulunan her X^2 değeri tek tek toplandı. X^2_{toplam}=16,608 bulunmuştur.
- Serbestlik derecesi u=k-1 işlemiyle bulunur. Buradaki k (16), rassal değerlerdir.
- Ki-Kare tablosundan serbestlik derecesinin olduğu satırdan toplam X^2 değerinden ilk büyük olan hücreye gelindi ve oradaki p aralığı gözden geçirildi. Gerekli karşılaştırmalar yapıldı.

Gözlenen değerler, bit aralıklarına göre tabloya yerleştirilirmiştir. Aşağıdaki tabloda bir bitin o çıktıda kaçar kez bulunduğu belirtilmiştir.

Tablo 1. 0 ile 15 bit arasında gözlenen rassal değerler.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	4	5	6	2	5	4	1	5	1	6	6	4	3	5	3	4
2	6	2	3	3	1	10	2	5	7	2	3	4	4	3	2	7
3	2	5	4	3	3	7	7	4	3	5	2	6	5	5	3	0
4	4	4	4	7	6	3	7	2	3	2	1	7	2	4	3	5
5	3	9	7	2	2	5	6	1	6	3	2	6	1	5	3	3

Tablo 2. Ki-Kare Testi verileri.

	Güven Değerleri										
DF	0.995	0.975	0.20	0.10	0.05	0.025	0.02	0.01	0.005	0.002	0.001
16	5.142	6.908	20.465	23.542	26.296	28.845	29.633	32.000	34.267	37.146	39.252

İncelenen Ki-kare tablosunda p=0.2 olduğu gözlemlenmiştir. Bu p değeri, aldığımız sonuçların güvenilirliğini kanıtlar niteliktedir.

5. Sonuçlar

Bu makalede Hash (Özet) Fonksiyonlarına yer verilmiş olup popüler kriptografik hash algoritmaları açıklanmıştır. SHA-3 Algoritması ile ilgili detaylı bir literatür araştırması yapılmış olup SHA-3 Algoritmasının çalışma mimarisine yer verilmiştir. Makalenin pratik uygulama kısmında ise Middle-Square (orta-kare) metodu ile rastgele sayı üretme işlemi yapılmıştır. C dili kullanılarak yazılan rastgele sayı üreteci komutlarıyla istenilen basamakta rassal sayılar elde edilmiştir. Elde edilen rassal sayılar, Python diliyle rastgele sayı üretecinden alınan giriş değerleri olup SHA-3 uygulaması gerçekleştirilmiştir. Makalenin son kısmında ise SHA3-256 hash fonksiyonu ile elde edilen beş farklı çıktı Ki-Kare Testi ile analiz edilip güvenilirliği hesaplanmıştır.

Ki-Kare Testi ile SHA3-256 algoritmasından elde edilen beş farklı çıktı analiz edilmiş ve p=0.2 bulunmuş olup küçük bir kümeden alınan giriş verilerine rağmen iyi bir sonuç elde edilmiştir. Bu sonuç, SHA3 algoritmasının güvenilirliğini kanıtlar niteliktedir. Makalenin amacına ulaşılmış ve popüler kriptografik hash algoritmalarından olan SHA-3 Algoritmasının güvenilirliği kanıtlanmıştır.

Bu çalışma; bilgi güvenliği sistemlerinde, modellemelerde, şifrelemelerde kullanılması amacıyla bu konuda çalışmayı planlayan araştırmacılar için bir rehber olma özelliği taşımaktadır. Bundan sonraki çalışmamızda SHA3 ile SHA2 algoritmalarının FPGA ve Rasperry-pi uygulamaları gerçekleştirilecek daha sonra bu algoritmaların performans ve güvenilirlikleri karşılaştırılacaktır.

Kaynaklar

[1] Liu, Zhe, Yung, Moti (Eds.). Information Security and Cryptology
[2] https://tr.qaz.wiki/wiki/Hash_function
[3] Süleyman Kardaş, Mehmet Sabır Kiraz. Bitcoin'de Mahremiyeti Sağlama Yöntemleri. Batman Üniversitesi, Mühendislik ve Mimarlık Fakültesi, Batman, Türkiye
[4] Saniye Soyalıç.Kriptografik Hash Fonksiyonları ve Uygulamaları, Yüksek Lisans Tezi
[5] https://tr.wikipedia.org/MD5
[6] Mehmet Keçeci.MD5 Nedir?
[7] Aditya Anand.Sha-3 Algorithm.Dec 9,2019.
[8] Özkaynak F. The Effects on Performance of Using Chaotic Systems in Entropy Source of Deterministic Random Number Generators. 11th CHAOS 2018 International Conference; 5 - 8 June 2018; Sapienza University of Rome, Italy.pp.415-420
[9] Stipčević M, Koç ÇK. True Random Number Generators. In: Koç ÇK (eds) Open Problems in Mathematics and Computational Science. Cham: Springer, 2014
[10] Knuth D. The Art of Computer Programming, Vol. 2, Seminumerical Algorithms. 2nd ed. Reading, Massachusetts: Addison-Wesley,1981.
[11] Ripley B. Computer Generation of Random Variables: A Tutorial. Int Stat Rev 1983; 51: 301-319.
[12] L'Ecuyer P. Random Numbers for Simulation. Commun ACM 1990; 33(1): 85-97.
[13] James F. A review of pseudorandom number generators. Compu Phys Commun 1990. 60: 329-344.
[14] Lagarias JC. Pseudorandom Number Generators in Cryptography and Number Theory. Proc Symp Appl Math 1990; 42:115–143.
[15] Ritter T. The Efficient Generation of Cryptographic Confusion Sequences. Cryptologia 1991; 15(2): 81-139.
[16] Schindler W. Random Number Generators for Cryptographic Applications. Koç ÇK (ed.): Cryptographic Engineering. Signals and Communication Theory, Berlin: Springer, 2009