Kavram · 19 Mayıs 2026

← Önceki·Sonraki →

SHA-256 Gerçekte Nedir?

Altmış dört karaktere sığan ve orijinal metindeki tek bir virgül bile hareket etse tamamen değişen matematiksel bir parmak izi. Neden ona dijital mühür mumu damgası diyoruz.

Teknik ismin ardındaki basit fikir

Tek bir yuvası ve tek bir ekranı olan bir makine hayal edin. Yuvadan bir metin giriyorsunuz: bir kelime, bir cümle, koca bir roman. Ekranda birkaç an sonra tam olarak altmış dört karakterlik bir dizi beliriyor. Biz profesyonel okuyucular buna *hash* veya *kriptografik özet* diyoruz; genel okuyucu içinse buna şimdilik metnin matematiksel parmak izi diyebiliriz, tıpkı parmak izinin bir insan için olduğu gibi.

Aynı metni iki kez girerseniz, makine her iki seferde de aynı parmak izini gösterir. Biraz farklı bir metin girerseniz —yer değiştirmiş tek bir virgül, küçük harfe dönüşen bir büyük harf— makine ilkinden tamamen farklı bir parmak izi gösterir. Benzer değil: bambaşka. Bu iki özellik bir arada —belirlenimcilik (determinism) ve hassasiyet— basit fikri oluşturur. SHA-256 hakkındaki diğer her şey, bunları düzgünce yerine getiren mekanizmadır.

Makinenin ne yapmadığını en baştan söylemekte fayda var. Metni şifrelemez. Onu gizlemez. Onu saklamaz. Makine metne bakar, parmak izini hesaplar ve metni unutur. Parmak izi, onu üreten metnin yeniden oluşturulmasına izin vermez; sadece aday bir metin verildiğinde, orijinaliyle eşleşip eşleşmediğini kontrol etmeye yarar. Bu yüzden buna *tek yönlü* bir özet diyoruz: gidilir ama dönülmez.

Bir hash şifreleme ile aynı şey değildir

Kafa karışıklığı yaygındır ve bunu gidermek gerekir: şifrelemek (encrypt) ve hash'lemek farklı işlemlerdir. Şifrelemek, bir metni sadece anahtar sahibinin orijinal formuna döndürebileceği şekilde dönüştürmekten ibarettir. Hash'lemek ise metnin bir parmak izini üretmektir ki orijinal metin bu izden asla geri kazanılamaz, anahtarlı veya anahtarsız. İlki tasarım gereği geri döndürülebilirdir (reversible); ikincisi ise tasarım gereği geri döndürülemezdir (irreversible).

Pratik sonucu önemlidir. Bir uygulama "şifrenizi şifreli olarak saklıyoruz" dediğinde, o şifreyi çözecek anahtara sahip biri vardır — her durumda uygulamanın kendisi. Bir uygulama "şifrenizi hash'lenmiş olarak saklıyoruz" dediğinde, uygulamanın kendisi istese bile orijinal şifreyi okuyamaz; sadece sizin yazdığınız şifrenin tekrar aynı parmak izini üretip üretmediğini kontrol edebilir. İkinci model, doğru yapıldığında, şifre saklamak için ilkine göre çok daha tercih edilebilirdir. "Doğru yapıldığında" ifadesinin neden sadece SHA-256'dan fazlasını gerektirdiğini ileride göreceğiz.

Bir kriptografik hash'i yararlı kılan dört özellik

Kriptografik sıfatını hak eden bir hash fonksiyonu şu dört özelliği karşılar:

1. **Belirlenimcilik (Determinism).** Aynı girdi her zaman aynı parmak izini üretir.
2. **Çığ etkisi (Avalanche effect).** Girdideki küçük bir değişiklik, öncekine görünür bir benzerliği olmayan, tamamen farklı bir parmak izi üretir.
3. **Tersine çevrilmeye karşı direnç.** Bir parmak izi verildiğinde, onu üreten metni bulmak hesaplama açısından mümkün değildir.
4. **Çakışma direnci (Collision resistance).** Aynı parmak izini üreten iki farklı metin bulmak hesaplama açısından mümkün değildir.

«Hesaplama açısından mümkün değildir» ifadesi «matematiksel olarak imkansızdır» anlamına gelmez. Bunu başarmanın zaman, enerji ve para maliyetinin, makul ölçüde mevcut olan tüm hesaplama kapasitesinin toplamını katbekat aştığı anlamına gelir. SHA-256 için bu sınır, özel donanımlarla yapılan en iyimser yaklaşımlarda bile binlerce trilyon yıl ile ölçülür. Bu da okuyucu için pratik anlamda «mümkün değil» ile aynı şeydir.

Özel olarak SHA-256

İsmi her şeyi anlatıyor. SHA, *Secure Hash Algorithm* (Güvenli Hash Algoritması) kelimelerinin kısaltmasıdır. 256 sayısı parmak izinin bit cinsinden boyutunu gösterir: iki yüz elli altı bit, yani otuz iki bayt; onaltılık (hexadecimal) sistemde gösterildiğinde okuyucunun artık tanıdığı o altmış dört karakterdir. Standart, bu tür fonksiyonları normalleştiren ABD kuruluşu NIST tarafından 2001 yılında SHA-2 ailesinin bir parçası olarak yayınlanmıştır; standardın mevcut sürümü olan FIPS 180-4, 2015 tarihlidir.

Henüz bitlerin ve baytların ne olduğunu bilmeyenler için:

```
1 bit    →  0 veya 1      (bir anahtar: açık veya kapalı)
1 byte   →  8 bit        (256 olası kombinasyon)
32 byte  →  256 bit      (SHA-256 parmak izi)
```

İsmin sonundaki 256 sayısı parmak izinin bit boyutunu söyler. Onaltılık sistemde —on yerine on altı sembollü bir sayma sistemi— bu 256 bit tam olarak 64 karaktere sığar. Bunlar, her Cuaderno'nun altında gördüğünüz o 64 karakterdir.

Boyutlar bir anlık durup düşünmeyi hak ediyor. İki yüz elli altı bit, iki üzeri iki yüz elli altı farklı değere izin verir: yetmiş sekiz ondalık basamaklı bir sayı, gözlemlenebilir evrendeki tahmini atom sayısından birkaç kat daha büyüktür. Dünyadaki her metin —her kitap, her e-posta, her mesaj— bu değerlerden birine düşer. İki farklı metnin tesadüfen çakışma olasılığı pratik amaçlar için sıfırdan farksızdır.

Kodda nasıl görünür?

Solo2'yi taşıyan parçaları yazdığımız dil olan Zig'de, bir metnin SHA-256 damgasını hesaplamak şöyle görünür:

```
const std = @import("std");

const texto = "Cuadernos Lacre";
var resumen: [32]u8 = undefined;
std.crypto.hash.sha2.Sha256.hash(texto, &resumen, .{});
```

Az önce Zig standart kütüphanesinden tırnak içindeki metnin SHA-256'sını hesaplamasını istedik. Çağrıdan sonra *resumen* değişkeni, damgayı ham haliyle oluşturan otuz iki baytı içerir; ekranda onaltılık sistemde gösterildiklerinde bu makalenin altındaki altmış dört karakterdir. Eğer *Cuadernos Lacre* ifadesini *Cuadernos lacre* olarak değiştirseydik —bir büyük harf eksik— damga tamamen değişirdi. Beş satırda, geri kalan her şeyi ayakta tutan temel özellik budur. Dahili olarak nasıl çalıştığını görmek isteyenler için makalenin sonunda algoritmanın adım adım yorumlanmış, okunabilir bir versiyonunu ekledik.

Neden ona mühür mumu damgası diyoruz?

On beşinci yüzyıldan on dokuzuncu yüzyıla kadar Avrupa yazışmalarında mektubu mühür mumu (lacre) kapatırdı. Erimiş bir balmumu damlası, üzerine bastırılan bir mühür ve mektup benzersiz bir şekilde işaretlenmiş olurdu. İçeriği kararlı bir gözetlemeciden korumazdı —kağıt ışığa tutularak okunabilir, mühür mumu kırılabilirdi— ama kanıtlardı. Kapanıştaki herhangi bir değişiklik, alıcıya kağıdı açmadan önce bile görünür olurdu. Mühür mumu hasarı engellemezdi; onu ifşa ederdi.

Her Cuaderno gövdesinin SHA-256'sı dijital versiyonda aynı işlevi görür. Makalenin yayınlandığı an ile okuduğunuz an arasında tek bir kelime bile değişse, metnin altındaki onaltılık damga artık elinizdeki metnin SHA-256'sı ile eşleşmez. Beş satırlık kodu olan her okuyucu bunu kontrol edebilir. Yayın, damga onu ele vermeden tarihini yeniden yazamaz. Hasara karşı korumaz; onu doğrulanabilir kılar.

Bir hash ne değildir?

Bazen SHA-256'dan ona ait olmayan dört kullanım beklenir:

1. **Şifreleme.** Bir hash özetler; gizlemez. Eğer metnin okunamamasını istiyorsanız onu hash'lemeniz değil, şifrelemeniz gerekir.
2. **Yazarı doğrulamak.** Bir hash metni kimin yazdığını söylemez, sadece hangi metnin hash'lendiğini söyler. Yazarlıkla ilişkilendirmek için hash'in üzerinde sadece hash'in kendisi değil, kriptografik bir imza gerekir.
3. **Şifreleri saklamak.** Burada anlaşılması gereken bir tuzak var. SHA-256 çok hızlı olacak şekilde tasarlanmıştır —bu birçok şey için iyidir ama bunun için kötüdür. Özel donanıma sahip bir saldırgan, sizinkini bulana kadar bir SHA-256 hash'ine karşı saniyede milyarlarca şifre deneyebilir. Şifreleri saklamak için, *tuz* (her kullanıcıya özel rastgele bir veri, aynı şifreye sahip iki kişinin aynı hash'e sahip olmasını engeller) ile birleştirilmiş Argon2, scrypt veya bcrypt gibi kasıtlı olarak yavaşlatılmış anahtar türetme fonksiyonları kullanılmalıdır.
4. **Hash'i yazar kimliği olarak okumak.** Öyle değildir. Bir hash içeriği tanımlar. Eğer iki kişi SHA-256 ile *merhaba* kelimesini hash'lerse, ikisi de aynı özeti elde eder — ve bu bir kusur değil, temel özelliktir: eğer özetler farklı olsaydı, yayınlanan ile alınan arasındaki eşleşmeyi kontrol edemezdik.

Günlük hayatınızda SHA-256 nerede karşınıza çıkar?

Siz görmeseniz de SHA-256, internette her gün kullandığınız şeylerin büyük bir kısmını ayakta tutar. Bitcoin blok zinciri, her bloğun SHA-256'sını bir sonrakine zincirleyerek inşa edilir; geçmiş bir bloğu değiştirmek sonraki tüm zincirin yeniden hesaplanmasını zorunlu kılar. Dünyanın yarısının kodlarını versiyonladığı sistem olan Git, her onayı (commit) içeriğinin tamamının SHA-256'sı (yeni sürümlerde) veya selefi SHA-1 (eski sürümlerde) ile tanımlar. Bir web sitesine girdiğinizde kimliğini doğrulayan HTTPS sertifikaları, ilişkili bir SHA-256 parmak izi taşır. Yazılım indirmeleri, dosyanın yolda değiştirilmediğini doğrulamanız için genellikle geliştirici tarafından yayınlanan bir SHA-256 ile birlikte gelir. Ve dediğimiz gibi, her Cuadernos Lacre'nin altında.

Profesyonel okuyucu için

Sistemlere karar veren veya onları denetleyenler için dört operasyonel hatırlatma:

1. Hash şifreleme değildir. Eğer bir tedarikçi teknik dokümantasyonunda bu iki terimi karıştırıyorsa, tam olarak ne demek istediğini sormakta fayda vardır.
2. Şifreleri saklamak için asla tek başına SHA-256 kullanılmamalıdır. SHA-256 bu iş için fazla hızlıdır (*Bir hash ne değildir* bölümündeki 3. maddeye bakınız). Mevcut standart **Argon2id**'dir: tasarım gereği yavaştır, sunucu kapasitesine göre yapılandırılabilir, kullanıcı başına farklı rastgele bir *tuz* ile birleştirilir.
3. Belgelerin —sözleşmeler, dosyalar, arşivler— bütünlüğü için SHA-256 referans standart olmaya devam etmektedir. AB'deki nitelikli zaman damgası hizmetlerinin kullandığı algoritmadır.
4. Uzun süreli koruma (on yıllar) için SHA-256'nın yanına bir SHA-3 veya SHA-512 hesaplayıp arşivlemek de faydalıdır; kriptografik ihtiyat, asırlık arşivlerde tek bir fonksiyona dayanmamanızı önerir.

Teknik olarak, ara durumun girdi blokları arasında korunduğu bu yinelenen yapı, SHA-1, SHA-2 (SHA-256 dahil) ve diğer birçok klasik hash fonksiyonunun dayandığı model olan **Merkle-Damgård** yapısı olarak bilinir. SHA-3 ise aksine, Merkle-Damgård yapısını terk ederek *sünger (sponge)* adı verilen farklı bir mimari lehine hareket eder.

SHA-256 nasıl çalışır, adım adım, yalın kelimelerle

Dünyanın en ayrıntılı domino devresini kurduğunuzu hayal edin: binlerce taş, onlarca çatal, mekanik köprüler ve tüm odayı geçen rampalar, parça parça özenle yerleştirilmiş.

İlk taşa bir dokunuş yaparsanız, zincir hassas ve tekrarlanabilir bir sıra ile düşer. Aynı kurulum, aynı başlangıç dokunuşu → tekrar tekrar düşen taşların aynı final deseni.

İşte ilginç olan şudur: Başlamadan önce **tek bir taşı** yarım santimetre yana kaydırın ve tekrar dokunun. Etkinleşmesi gereken bir rampa hareketsiz kalır, bir köprü düşmez, farklı bir çatal tetiklenir. Yerdeki taşların son deseni, ilkiyle karşılaştırıldığında tamamen tanınmaz haldedir.

SHA-256 matematiksel olarak bu devredir. Yazdığınız metin, taşların başlangıç konumudur. Algoritma, şelaleyi serbest bırakan dokunuştur. Ve sonuç — *hash* dediğimiz şey — her şey durduğundaki zeminin donmuş fotoğrafıdır. Orijinal

metinden tek bir virgülü değiştirin, fotoğraf radikal bir şekilde farklı olacaktır. İşte bu kadar basit ve bu kadar çarpıcı.

1. Adım: Metni ikili taşlara çevirmek. Bilgisayarlar harflerden anlamaz; onları önce sayılara (ASCII), sayıları da ikiliye (birler ve sıfırlar) çevirirler. Her harf 8 beyaz veya siyah taşa dönüşür: *A* `01000001`'dir, *B* `01000010`'dur, boşluk `00100000`'dır. Tüm metniniz — bir kelime, bir sözleşme, bir roman — uzun bir beyaz ve siyah taş dizisi haline gelir.

2. Adım: Standart boyuta kadar doldurmak. Devre, diziyi tam olarak 512 taşlık *bölümler* halinde işler. Mesajınız 512'nin katına ulaşmıyorsa, metinden hemen sonra bir işaretleyici taş (`10000000` değerindeki taş) ve ardından bölümü tamamlamak için sıfırlar eklenir. Her bölümün son 64 pozisyonu, metnin orijinal uzunluğunu kaydetmek için ayrılmıştır. Böylece devre, gerçek içeriğin nerede bittiğini ve dolgunun nerede başladığını her zaman bilir.

3. Adım: Sekiz usta taşı yerleştirmek. Başlamadan önce, masaya tam bir başlangıç konumunda **sekiz usta taş** yerleştiririz. Bu sekiz taş sır değildir: başlangıç değerleri halka açık bir matematiksel kural (ilk sekiz asal sayının karekökleri — 2, 3, 5, 7, 11, 13, 17, 19 — ve her karekökün ondalık kısmının ilk bitleri) tarafından belirlenmiştir. Dünyanın her köşesindeki herkes, aynı pozisyondaki aynı sekiz usta taşla başlar. Kaderleri, çığ tarafından itilmek ve dönüştürülmektir.

4. Adım: Büyük çığ: altmış dört tur itiş. İşte gösteri burada başlıyor. Metninizin 512 taşlık ilk bölümü, sekiz usta taşa çarptırılır. Ancak birden düşmezler: mekanizma **altmış dört ardışık tur** yürütür. Her turda taşlarla üç işlem yapar:

- **Atlıkarınca** (döndürme). Taşlar bir daire içinde hareket eder: sağdakiler sola geçer. Hiçbir taş kaybolmaz veya eklenmez; atlıkarıncada tam bir tur atarak yeniden düzenlenirler. Bu, bilgiyi yeniden dağıtmanın ucuz ve geri döndürülebilir bir yoludur.
- **Mantıksal Huni** (XOR). Taşlar, onları ikişer ikişer karşılaştıran bir huniden geçer: her ikisi de aynı renkse beyaz bir taş çıkar; farklılarsa siyah bir taş çıkar. İkili mantığın en basit işlemidir, ancak atlıkarıncanın döndürmeleriyle birleştiğinde, bilgiyi kaybetmeden karıştırmak için çok güçlü hale gelir.
- **Taşma** (modüler toplama). Sonuç, altmış dört sabitten oluşan halka açık bir listeden (ilk altmış dört asal sayının küp kökleri) getirilen bir *sabit itme taşı* ile toplanır. Toplam, öngörülen 32 taşlık alana sığmayan ekstra taşlar üretirse, bu fazla taşlar atılır. Masada sadece 32 taşlık yer vardır, bir tane fazla değil.

Altmış dördüncü turun sonunda, metin bölümünüzdeki taşların her biri sekiz usta taşın konumunu etkilemiştir. İtiş enerjisi tüm devre boyunca seyahat etmiştir.

5. Adım: Sonraki bölümü eklemek (yeniden başlatmadan). Metniniz uzunsa ve işlenecek başka bir 512 taşlık bölüm varsa, **devre yeniden başlatılmaz**. Sekiz usta taş ilk çığın bıraktığı gibi kalır ve ikinci bölüm diğer altmış dört turu etkinleştirmek için onlara fırlatılır. Bu, az önce düşen odanın sonuna dominolarla dolu yeni bir oda eklemek gibidir: ilkinin düzensizliği, ikincisinin nasıl düşeceğini tamamen şartlandırır.

6. Adım: Final fotoğrafını çekmek. Artık işlenecek bölüm kalmadığında çığ durur. Sekiz usta taşın kaldığı final konumuna bakarız. Yapılandırmalarını onaltılık sistemde bir harf ve sayı koduna çevirириz. Sonuç, tam olarak altmış dört karakterlik bir dizidir: bu sizin SHA-256 mührünüzdür.

Devrenin nasıl kurulduğundan dört özellik kendiliğinden ortaya çıkar:

1. **Belirlenimcilik.** Aynı metin, dünyanın her yerindeki herhangi bir bilgisayarda her zaman aynı final fotoğrafını üretir. Sıfır rastgelelik, sıfır sürpriz.
2. **Çığ etkisi.** Eklenen bir virgül, değiştirilen bir büyük harf, unutulan bir aksan: fotoğraf tamamen tanınmaz hale gelir. Bu, başlangıçta tarif ettiğimiz aşırı hassasiyettir.
3. **Tek yönlü.** Final fotoğrafı verildiğinde, orijinal metni yeniden oluşturamazsınız. Döndürmeler, huniler ve taşmalar, *her bir bitin nereden geldiğine* dair tüm yönsel bilgiyi yok eder ve sadece *toplamda neyin eklendiğini* korur.
4. **Çakışma direnci.** Yirmi beş yıllık halka açık kripto analizinde, kimse final fotoğrafları çakışan iki farklı metin bulmayı başaramadı. Ve bunu yapmanın zorluğu, makul olarak hayal edilebilecek herhangi bir medeniyetin hesaplama yeteneğinin ötesindedir.

Aşağıdaki kod eki, Zig'de tam olarak bu altı adımı uygular. Artık bit işlemlerini körü körüne kabul etmek yerine, her birinin ne anlama geldiğini bilerek okuyabilirsiniz.

Teknik sözlük

Her bir işlemin ne yaptığını anlamak isteyen okuyucu içindir. Özgürce atlayabilirsiniz: makale onsuz da anlaşılmaya devam eder.

ASCII ve Unicode — harfler nasıl sayıya dönüşür. Bilgisayarlar harfleri görmez; sayıları görürler. 1963 tarihli **ASCII** (*American Standard Code for Information Interchange*) adlı bir standart, her klavye karakterine belirli bir sayı atar: *A* 65, *B* 66, *a* 97, *0* 48, boşluk 32, virgül 44'tür. Modern sistemler bunu, dünyadaki her alfabenin her karakterine bir sayı atayan **Unicode** ile genişletir: Kiril, Arapça, Çince, Japonca ve hatta emojiler. Bir karakter yazdığınızda veya bir metin dosyası açtığınızda, bilgisayar ekrandaki şekli değil, arka plandaki sayıyı okur. SHA-256 bu sayılar üzerinde çalışır ve herhangi bir metni uzun bir rakam dizisi olarak ele alır. Bu nedenle bir İspanyolca makaleyi, bir Japonca şiiri ve bir ikili dosyayı aynı algoritma ile mühürleyebilir.
XOR — bit bit karşılaştırıcı. XOR (İngilizce *exclusive or*, «özel veya» ifadesinden gelir), bir bilgisayarın iki ikili sayı ile yapabileceği en basit işlemlerden biridir. İki biti pozisyon pozisyon karşılaştırır ve şunu döndürür: **1** eğer ikisinden tam olarak biri 1 ise (biri ama ikisi birden değil), **0** eğer ikisi aynıysa (ikisi de 0 veya ikisi de 1). Örnek: `1010` ve `1100`'ün XOR'u `0110`'dur. Önemli bir özelliği vardır: geri döndürülebilirdir —aynı anahtarla iki kez XOR yaparsanız, orijinaline dönersiniz—. Bu nedenle kriptografinin can damarıdır: bilgiyi kaybetmeden bitleri karıştırır, ancak girdilerden birini bilmiyorsanız sonuç girdiler hakkında hiçbir şey ortaya çıkarmaz.
Onaltılık — 16 tabanında saymak. Günlük hayattaki neredeyse tüm sayılar on basamak (0-9) kullanır. Onaltılık (hexadecimal) on altı basamak kullanır: her zamanki 0-9 artı şu değerleri temsil eden altı harf: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Neden on altı? Çünkü bilgisayarlar dört bitlik gruplar halinde düşünür ve dört bit tam olarak on altı farklı değeri temsil edebilir —böylece, bir onaltılık karakter temiz bir şekilde dört bite karşılık gelir—. Bir SHA-256 mührü 256 bit ölçüsündedir, bu da tam olarak **64 onaltılık karakterdir**. Onu normal ondalık sistemde yazsaydık, yaklaşık 78 basamak yer kaplardı ve daha zahmetli olurdu. Seçim estetik ve kompakttır; arka plandaki sayı aynıdır.
Bit döndürme — ikili atlıkarınca. Bazıları yanık (1) ve bazıları sönük (0) yedi ampullük bir sıra hayal edin: `1 0 1 1 0 0 1`. Bir pozisyon sağa döndürmek, en sağdaki ampulü alıp en sol uca getirmek ve diğerlerini bir yer sağa kaydırmaktan ibarettir: `1 1 0 1 1 0 0`. Hiçbir ampul kaybolmaz veya eklenmez: sadece bir daire içinde dans ederler. SHA-256, her hesaplamada yüzlerce kez bit döndürmeyi kullanır; bu, durum içindeki bilgiyi yeniden dağıtmanın ucuz ve kayıpsız bir yoludur.
"Nothing-up-my-sleeve" sabitleri — neden asal sayılardan gelirler. SHA-256'nın sekiz usta taşı ve altmış dört tur sabiti rastgele seçilmemiştir. İlk asal sayıların kareköklerinden ve küp köklerinden gelirler. Neden? Çünkü tasarımcıları *«yeninde hiçbir şey saklamayan»* (nothing up my sleeve) sabitler istiyorlardı: kökenini herkesin doğrulayabileceği değerler. Birisi size *«bana güven: bu 32 bitlik rastgele sayıyı kullan»* deseydi, haklı olarak gizli bir zayıflıktan veya bir arka kapıdan şüphelenirdiniz. Ancak hesap makinesi olan herkes 2'nin karekökünün ilk 32 bitinin `0x6a09e667` olduğunu doğrulayabilir. Değerler matematikseldir, halka açıktır ve tekrarlanabilir: tarife hiçbir gizli tuzak sızamaz.

Ek: Okunabilir kodda SHA-256

Bu ek, algoritmayı içeriden görmek isteyen okuyucular içindir. FIPS 180-4 spesifikasyonunu izleyen öğretici bir Zig uygulamasıdır. Solo2'nin kullandığı sürüm değildir —gerçek olan Zig standart kütüphanesindeki `std.crypto.hash.sha2.Sha256` içindedir, optimize edilmiş ve denetlenmiştir—. Ancak algoritma aynıdır: Burada gördüğünüz şey, o beş karakterlik çağrı işini yaparken adım adım gerçekleşen şeydir.

```
const std = @import("std");

// SHA-256 — implementación didáctica.
// Sigue la especificación FIPS 180-4. Prioriza la claridad sobre la
// velocidad y la robustez frente a entradas hostiles. Para producción,
// usa std.crypto.hash.sha2.Sha256, que está optimizada y auditada.

// H0: las ocho palabras del estado inicial. Primeros 32 bits de la parte
// fraccionaria de las raíces cuadradas de los primeros ocho primos
// (2, 3, 5, 7, 11, 13, 17, 19).
const H0 = [_]u32{
    0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a,
    0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19,
};

// K: 64 constantes de ronda. Primeros 32 bits de la parte fraccionaria
// de las raíces cúbicas de los primeros 64 primos.
const K = [_]u32{
    0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
```

```
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2,
};

// Rotación circular a la derecha de un u32.
inline fn rotr(x: u32, n: u5) u32 {
    return std.math.rotr(u32, x, n);
}

// Lee 4 bytes consecutivos como un u32 big-endian.
inline fn readU32(b: []const u8) u32 {
    return @as(u32, b[0]) << 24 | @as(u32, b[1]) << 16 | @as(u32, b[2]) << 8 | @as(u32, b[3]);
}

// Escribe un u32 como 4 bytes consecutivos big-endian.
inline fn writeU32(b: []u8, v: u32) void {
    b[0] = @truncate(v >> 24);
    b[1] = @truncate(v >> 16);
    b[2] = @truncate(v >> 8);
    b[3] = @truncate(v);
}

// Compresión de un bloque de 64 bytes sobre el estado del hash. Sigue §6.2.2 de FIPS 180-4.
fn compress(state: *[8]u32, block: [16]u32) void {

    // 1. Expansión del schedule: 16 palabras → 64. Las nuevas se obtienen
    //    combinando cuatro anteriores con dos funciones de mezcla (s0 y s1)
    //    que usan rotación, XOR y desplazamiento. El "+%" es suma con
    //    truncado u32 (overflow-wrap), tal como exige el estándar.
    var w: [64]u32 = undefined;
    for (0..16) |i| w[i] = block[i];
    for (16..64) |i| {
        const s0 = rotr(w[i-15], 7) ^ rotr(w[i-15], 18) ^ (w[i-15] >> 3);
        const s1 = rotr(w[i-2], 17) ^ rotr(w[i-2], 19) ^ (w[i-2] >> 10);
        w[i] = w[i-16] +% s0 +% w[i-7] +% s1;
    }

    // 2. Variables de trabajo: copia del estado actual.
    var a = state[0]; var b = state[1]; var c = state[2]; var d = state[3];
    var e = state[4]; var f = state[5]; var g = state[6]; var h = state[7];

    // 3. 64 rondas de mezcla no lineal.
    //    S1, S0  : combinaciones rotacionales de 'e' y 'a'.
    //    ch      : "choose" — multiplexor bit a bit, elige entre f y g según e.
    //    maj     : "majority" — bit mayoritario entre a, b, c.
    //    t1 + t2 : se inyecta al top de la cascada cada ronda.
    for (0..64) |i| {
        const S1 = rotr(e, 6) ^ rotr(e, 11) ^ rotr(e, 25);
        const ch = (e & f) ^ (~e & g);
        const t1 = h +% S1 +% ch +% K[i] +% w[i];
        const S0 = rotr(a, 2) ^ rotr(a, 13) ^ rotr(a, 22);
        const maj = (a & b) ^ (a & c) ^ (b & c);
        const t2 = S0 +% maj;
        h = g; g = f; f = e; e = d +% t1;
        d = c; c = b; b = a; a = t1 +% t2;
    }

    // 4. Acumular las variables de trabajo en el estado.
    state[0] +%= a; state[1] +%= b; state[2] +%= c; state[3] +%= d;
    state[4] +%= e; state[5] +%= f; state[6] +%= g; state[7] +%= h;
}

// Hash completo: procesa el mensaje en bloques, padea el último, escribe el resumen.
pub fn sha256(msg: []const u8, out: *[32]u8) void {
    var state = H0;
    var block: [64]u8 = undefined;
    var block_w: [16]u32 = undefined;
```

```
    // Procesar bloques completos del mensaje original.
    var i: usize = 0;
    while (i + 64 <= msg.len) : (i += 64) {
        @memcpy(block[0..64], msg[i..i+64]);
        for (0..16) |j| block_w[j] = readU32(block[j*4..j*4+4]);
        compress(&state, block_w);
    }

    // Padding del último bloque: byte 0x80, después ceros, después la
    // longitud original (en bits) como u64 big-endian en los 8 últimos bytes.
    const remaining = msg.len - i;
    @memcpy(block[0..remaining], msg[i..]);
    block[remaining] = 0x80;
    const bit_len: u64 = @as(u64, msg.len) * 8;

    if (remaining + 1 + 8 <= 64) {
        // El padding cabe en el mismo bloque.
        for (remaining + 1..56) |k| block[k] = 0;
        var k: usize = 0;
        while (k < 8) : (k += 1) block[56 + k] = @truncate(bit_len >> @as(u6, @intCast((7 - k) * 8)));
        for (0..16) |j| block_w[j] = readU32(block[j*4..j*4+4]);
        compress(&state, block_w);
    } else {
        // El padding requiere un bloque adicional.
        for (remaining + 1..64) |k| block[k] = 0;
        for (0..16) |j| block_w[j] = readU32(block[j*4..j*4+4]);
        compress(&state, block_w);
        for (0..56) |k| block[k] = 0;
        var k: usize = 0;
        while (k < 8) : (k += 1) block[56 + k] = @truncate(bit_len >> @as(u6, @intCast((7 - k) * 8)));
        for (0..16) |j| block_w[j] = readU32(block[j*4..j*4+4]);
        compress(&state, block_w);
    }

    // Escribir el estado final como 32 bytes big-endian.
    for (0..8) |j| writeU32(out[j*4..j*4+4], state[j]);
}

// Ejemplo de uso.
pub fn main() void {
    var resumen: [32]u8 = undefined;
    sha256("Cuadernos Lacre", &resumen);
    for (resumen) |byte| std.debug.print("{x:0>2}", .{byte});
    std.debug.print("\n", .{});
    // Imprime: ae6bdea6bbf5476889e0651a31f3dc1612fc61497477e21a95cabae2a6886c3e
}
```

Aynı yapıyı —başlangıç sabitleri, schedule genişletme, altmış dört tur, biriktirme— izleyen başka bir dildeki herhangi bir yeniden yazım aynı sonucu verir. Algoritmanın sırrı yoktur: değeri, yukarıda sayılan özelliklerin yirmi yıllık kamuya açık kriptanaliz ve binlerce gözden sonra hala geçerliliğini korumasındadır.

Bu makalenin altına geri dönerseniz, altmış dört karakterlik onaltılık bir damga göreceksiniz. Bu, az önce okuduğunuz metnin bu dildeki SHA-256'sıdır. Eğer makaleyi çevirseydik damga başka olurdu; İspanyolca versiyonundaki bir kelime değişseydi İspanyolca damga değişirdi. Damga içeriği korumaz —bunun için başka araçlar vardır— ancak onu benzersiz bir şekilde tanımlar. Ve bu, kulağa ne kadar mütevazı gelse de, yayın zincirinin hiçbir adımının söylenenleri fark edilmeden değiştirememesi için yeterlidir. Geri kalan her şey —şifreleme, imzalama, kimlik belirleme— bu basit fikrin üzerine inşa edilir.

Kaynaklar ve ek okumalar

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, Ağustos 2015. SHA-256 dahil SHA-2 ailesinin resmi spesifikasyonu.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, Mayıs 2011. Uygulayıcılar için normatif versiyon.

- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). 5. ve 6. bölümler hash fonksiyonlarını ve bunların meşru ve gayrimeşru kullanımlarını kapsar.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Blokları yapı gereği değişmez bir yapıda birbirine zincirlemek için SHA-256 kullanımının pratik örneği.
- 910/2014 (eIDAS) sayılı AB Tüzüğü — nitelikli zaman damgalayıcılarının çerçevesi. SHA-256, AB'de düzenlenen nitelikli elektronik imzalar ve mühürler için referans fonksiyondur.
- Zig'deki referans uygulama: Dilin resmi deposunda `std.crypto.hash.sha2.Sha256` (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Solo2'nin aslında kullandığı optimize edilmiş ve denetlenmiş versiyondur. Ekin öğretici uygulamasıyla karşılaştırmak için yararlıdır.

← ÖncekiCUADERNOS_LIST_SCHREMS_TITLESonraki →CUADERNOS_LIST_KILLSWITCH_TITLE

Son okumalar

- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE

Bu makaleyi ihtiyacınız olan her yere yanınızda götürün.

↓ Markdown ↓ Düz metin ↓ PDF

Dosya cihazınıza indirilecektir. Oradan kaydedebilir, Solo2'ye aktarabilir veya istediğiniz yerde paylaşabilirsiniz. Cuadernos hedef noktaya sizin yerinize karar vermez.

Mühür mumu · SHA-256 a524db28735f4753c5a124da8174914d3d55f256875d85c3ff7488160d2b21d3

Cuadernos Lacre · Menzuri Gestión S.L. yayını ·
R.Eugenio tarafından yazıldı · Solo2 ekibi tarafından düzenlendi.