Što je zapravo SHA-256

Matematički otisak koji stane u šezdeset i četiri znaka i koji se potpuno mijenja ako se pomakne samo jedan zarez u izvornom tekstu. Zašto ga zovemo digitalnim pečatnim voskom.

Jednostavna ideja iza tehničkog naziva

Zamislite da postoji stroj s jednim utorom i jednim zaslonom. Kroz utor ubacujete tekst: riječ, rečenicu, cijeli roman. Na zaslonu se, trenutak kasnije, pojavljuje niz od točno šezdeset i četiri znaka. Taj niz stručni čitatelji zovu *hash* ili *kriptografski sažetak*; obični čitatelji to zasad mogu zvati matematičkim otiskom teksta, baš kao što je otisak prsta otisak osobe.

Ako isti tekst unesete dva puta, stroj će oba puta pokazati isti otisak. Ako unesete malo drugačiji tekst — jedan pomaknuti zarez, veliko slovo koje postane malo — stroj će pokazati potpuno drugačiji otisak od prvog. Ne sličan: drugačiji. Ta dva svojstva zajedno — determinizam i osjetljivost — čine jednostavnu ideju. Sve ostalo kod SHA-256 je mehanizam koji ih tjera da rade ispravno.

Vrijedi odmah na početku reći što stroj ne radi. On ne kriptira tekst. Ne skriva ga. Ne sprema ga. Stroj gleda tekst, izračunava otisak i zaboravlja tekst. Otisak ne dopušta rekonstrukciju teksta koji ga je proizveo; on samo dopušta, s obzirom na ponuđeni tekst, provjeru podudara li se on s izvornikom. Zato kažemo da je to *jednosmjerni* sažetak: ide se naprijed, ali se ne vraća.

Hash nije isto što i kriptiranje

Zabuna je česta i vrijedi je razjasniti: kriptiranje i hashiranje različite su operacije. Kriptiranje se sastoji od transformacije teksta tako da ga samo vlasnik ključa može vratiti u izvorni oblik. Hashiranje se sastoji od stvaranja otiska teksta iz kojeg se izvorni tekst nikada ne može vratiti, ni s ključem ni bez njega. Prvo je po dizajnu reverzibilno; drugo je po dizajnu ireverzibilno.

Praktična posljedica je važna. Kada aplikacija kaže „spremamo vašu lozinku kriptiranu“, postoji netko tko ima ključ za njezino dekriptiranje — u svakom slučaju sama aplikacija. Kada aplikacija kaže „spremamo vašu lozinku hashiranu“, sama aplikacija ne može pročitati izvornu lozinku čak i da to želi; može samo provjeriti proizvodi li ono što upišete ponovno isti otisak. Drugi model, ako se izvede ispravno, puno je bolji od prvog za pohranjivanje lozinki. Kasnije ćemo vidjeti zašto „ispravno izvedeno“ zahtijeva nešto više od samog SHA-256.

Četiri svojstva koja kriptografski hash čine korisnim

Hash funkcija koja zaslužuje pridjev *kriptografska* ispunjava četiri svojstva:

1. **Determinizam.** Isti ulaz uvijek proizvodi isti otisak.
2. **Efekt lavine.** Mala promjena u ulazu proizvodi potpuno drugačiji otisak, bez vidljive sličnosti s prethodnim.
3. **Otpornost na inverziju.** Na temelju otiska nije računalno izvedivo pronaći tekst koji ga je proizveo.
4. **Otpornost na kolizije.** Nije računalno izvedivo pronaći dva različita teksta koja proizvode isti otisak.

„Nije računalno izvedivo“ ne znači „matematički je nemoguće“. To znači da trošak vremena, energije i novca za postizanje toga premašuje redove veličine ukupnog razumno dostupnog računalnog kapaciteta. Za SHA-256 se ta granica mjeri u tisućama bilijuna godina, čak i uz najoptimističnije pristupe sa specijaliziranim hardverom. Što je za praktične svrhe čitatelja isto što i „ne može se“.

SHA-256, konkretno

Ime govori sve. SHA je kratica za *Secure Hash Algorithm*: sigurni hash algoritam. Broj 256 označava veličinu otiska u bitovima: dvjesto pedeset i šest bitova, odnosno trideset i dva bajta, koji su prikazani u heksadekadskom obliku onih šezdeset i četiri znaka koje čitatelj već prepoznaje. Standard je objavio američki NIST, tijelo koje normira ovu vrstu funkcija, 2001. godine kao dio obitelji SHA-2; trenutna verzija standarda, FIPS 180-4, datira iz 2015.

Za one koji još ne znaju što su bitovi i bajtovi:

```
1 bit    →  0 ili 1        (prekidač: uključen ili isključen)
1 bajt   →  8 bitova       (256 mogućih kombinacija)
32 bajta →  256 bitova     (SHA-256 otisak)
```

Broj 256 na kraju imena govori o veličini otiska u bitovima. U heksadekadskom sustavu — sustavu brojeva sa šesnaest simbola umjesto deset — tih 256 bitova stane u točno 64 znaka. To su 64 znaka koja vidite na dnu svakog Cuaderna.

Dimenzije zaslužuju trenutak pažnje. Dvjesto pedeset i šest bitova omogućuje dva na dvjesto pedeset i šestu različitih vrijednosti: broj sa sedamdeset i osam decimalnih znamenki, nekoliko redova veličine veći od procijenjenog broja atoma u vidljivom svemiru. Svaki tekst na svijetu — svaka knjiga, svaki e-mail, svaka poruka — pada na jednu od tih vrijednosti. Vjerojatnost da se dva različita teksta slučajno podudare u praksi se ne razlikuje od nule.

Kako to izgleda u kodu

U Zigu, jeziku na kojem pišemo dijelove koji podržavaju Solo2, izračunavanje SHA-256 pečata teksta izgleda ovako:

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

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

Upravo smo zatražili od standardne knjižnice Ziga da izračuna SHA-256 teksta pod navodnicima. Nakon poziva, varijabla *resumen* sadrži trideset i dva bajta koji čine pečat u svom sirovom obliku; kada se prikažu na zaslonu u heksadekadskom obliku, to su onih šezdeset i četiri znaka koji se pojavljuju na dnu ovog članka. Kad bismo promijenili *Cuadernos Lacre* u *Cuadernos lacre* — jedno veliko slovo manje — pečat bi se potpuno promijenio. To je, u pet redaka, središnje svojstvo koje podupire sve ostalo. Za one koji žele vidjeti kako to interno radi, na kraju članka uključujemo čitljivu verziju algoritma s komentarima korak po korak.

Zašto ga zovemo pečatnim voskom

U europskoj korespondenciji od petnaestog do devetnaestog stoljeća, pečatni vosak je zatvarao pismo. Kap rastopljenog voska, pečat pritisnut na njega i pismo bi ostalo neponovljivo označeno. To nije štitilo sadržaj od odlučnog njuškala — papir se mogao pročitati kroz svjetlo, vosak se mogao slomiti — ali ga je dokazivalo. Svaka izmjena zatvarača bila je vidljiva primatelju i prije otvaranja papira. Pečatni vosak nije sprječavao štetu; on ju je objavljivao.

SHA-256 tijela svakog Cuaderna obavlja istu funkciju u svojoj digitalnoj verziji. Ako se samo jedna riječ u članku promijeni između trenutka objave i trenutka kada ga čitate, heksadekadski pečat na dnu teksta više se ne bi podudarao s SHA-256 teksta koji imate ispred sebe. Svaki čitatelj s pet redaka koda mogao bi to provjeriti. Publikacija ne može prepisati svoju povijest, a da je pečat ne oda. On ne štiti od štete; on je čini provjerljivom.

Što hash nije

Od SHA-256 se ponekad traže četiri namjene koje mu ne pripadaju:

1. **Kriptiranje.** Hash sažima; ne skriva. Ako želite da tekst ne bude čitljiv, trebate ga kriptirati, a ne hashirati.
2. **Autentifikacija autora.** Hash ne govori tko je napisao tekst, već samo koji je tekst hashiran. Za povezivanje autorstva potreban je kriptografski potpis povrh hasha, a ne sam hash.
3. **Pohranjivanje lozinki.** Ovdje postoji zamka koju vrijedi razumjeti. SHA-256 je dizajniran da bude vrlo brz — što je dobro za mnoge stvari, ali loše za ovo. Napadač sa specijaliziranim hardverom može isprobati milijarde lozinki u

sekundi protiv SHA-256 hasha dok ne pogodi vašu. Za spremanje lozinki moraju se koristiti namjerno spore funkcije za izvođenje ključeva kao što su Argon2, scrypt ili bcrypt, u kombinaciji sa *solju* (jedinstveni nasumični podatak po korisniku, koji sprječava da dvije osobe s istom lozinkom imaju isti hash).
4. **Čitanje hasha kao identifikatora autora.** To nije to. Hash identificira sadržaj. Ako dvije osobe hashiraju riječ *bok* pomoću SHA-256, obje dobivaju isti sažetak — i to je središnje svojstvo, a ne nedostatak: kad bi to bili različiti sažeci, ne bismo mogli provjeriti podudarnost između objavljenog i primljenog.

Gdje se SHA-256 pojavljuje u vašem svakodnevnom životu

Iako to ne vidite, SHA-256 podržava dobar dio onoga što svakodnevno koristite na internetu. Bitcoin blockchain gradi se ulančavanjem SHA-256 svakog bloka sa sljedećim; izmjena prošlog bloka prisiljava na ponovno izračunavanje cijelog kasnijeg lanca. Git, sustav kojim se verzira kod polovice svijeta, identificira svaku potvrdu (commit) pomoću SHA-256 (u novijim verzijama) ili njegovog prethodnika SHA-1 (u starijim verzijama) njegovog cjelokupnog sadržaja. HTTPS certifikati koji verificiraju identitet web stranice kada uđete imaju povezani SHA-256 otisak. Preuzimanja softvera često su popraćena SHA-256 otiskom koji je objavio programer kako biste mogli provjeriti da datoteka nije izmijenjena usput. I, kao što smo rekli, na dnu svakog Cuaderno Lacre.

Za profesionalnog čitatelja

Četiri operativna podsjetnika za one koji odlučuju o sustavima ili ih revidiraju:

1. Hash nije kriptiranje. Ako dobavljač pobrka ova dva pojma u svojoj tehničkoj dokumentaciji, vrijedi pitati što točno želi reći.
2. Za pohranjivanje lozinki nikada se ne smije koristiti samo SHA-256. SHA-256 je prebrz za ovaj zadatak (vidi točku 3 u *Što hash nije*). Trenutni standard je **Argon2id**: spor po dizajnu, podesiv prema kapacitetu poslužitelja, u kombinaciji s različitom nasumičnom *solju* po korisniku.
3. Za integritet dokumenata — ugovora, spisa, arhiva — SHA-256 ostaje referentni standard. To je onaj koji koriste kvalificirani pružatelji usluga vremenskog žiga u EU.
4. Za dugoročno očuvanje (desetljećima) vrijedi izračunati i arhivirati SHA-3 ili SHA-512 uz SHA-256; kriptografski oprez preporučuje da se ne oslanjate na jednu funkciju za stoljetne arhive.

Tehnički, ova iterirana struktura — gdje se međustanje čuva između ulaznih blokova — poznata je kao **Merkle-Damgård** konstrukcija, obrazac na kojem se temelje SHA-1, SHA-2 (uključujući SHA-256) i mnoge druge klasične hash funkcije. SHA-3, nasuprot tome, napušta Merkle-Damgård u korist drugačije arhitekture nazvane *spužva (sponge)*.

Kako radi SHA-256, korak po korak, jednostavnim riječima

Zamislite da ste složili najrazrađeniji domino krug na svijetu: tisuće pločica, deseci račvanja, mehanički mostovi i rampe koji se protežu cijelom sobom, pažljivo postavljeni dio po dio.

Ako dotaknete prvu pločicu, lanac pada u preciznom i ponovljivom nizu. Isti postav, isti početni dodir → identičan završni uzorak palih pločica, iznova i iznova.

Ovdje postaje zanimljivo: pomaknite **samo jednu pločicu** pola centimetra u stranu prije početka i ponovno je dotaknite. Rampa koja se trebala aktivirati ostaje nepomična, most ne pada, aktivira se drugo račvanje. Završni uzorak pločica na podu potpuno je neprepoznatljiv u usporedbi s prvim.

SHA-256 je matematički gledano upravo ovaj krug. Tekst koji upišete početni je položaj pločica. Algoritam je dodir koji pokreće lavinu. A konačni rezultat — ono što zovemo *hash* — statična je fotografija poda kada se sve zaustavi. Promijenite samo jedan zarez u izvornom tekstu i fotografija će biti radikalno drugačija. Tako jednostavno, a tako drastično.

Korak 1. Prevođenje teksta u binarne pločice. Računala ne razumiju slova; ona ih prvo prevode u brojeve (ASCII), a brojeve u binarni sustav (jedinice i nule). Svako se slovo pretvara u 8 bijelih ili crnih pločica: *A* je `01000001`, *B* je `01000010`, razmak je `00100000`. Cijeli vaš tekst — riječ, ugovor, roman — postaje dugi red bijelih i crnih pločica.

Korak 2. Nadopunjavanje do standardne veličine. Krug obrađuje red u *dionicama* od točno 512 pločica. Ako vaša poruka nije višekratnik broja 512, odmah nakon teksta dodaje se pločica marker (vrijednosti `10000000`), a zatim nule dok

se dionica ne popuni. Posljednja 64 mjesta svake dionice rezervirana su za bilježenje izvorne duljine teksta. Tako krug uvijek zna gdje je stvarni sadržaj završio, a gdje je počela nadopuna.

Korak 3. Postavljanje osam glavnih pločica. Prije početka, na stol postavljamo **osam glavnih pločica** u točno određen početni položaj. Tih osam pločica nije nikakva tajna: njihova početna vrijednost određena je javnim matematičkim pravilom (drugi korijeni prvih osam prostih brojeva — 2, 3, 5, 7, 11, 13, 17, 19 — i prvi bitovi decimalnog dijela svakog korijena). Svatko, u bilo kojem kutku planeta, počinje s istih osam glavnih pločica u istom položaju. Njihova je sudbina da ih lavina gura i transformira.

Korak 4. Velika lavina: šezdeset i četiri kruga guranja. Ovdje počinje predstava. Prva dionica od 512 pločica vašeg teksta udara u osam glavnih pločica. Ali one ne padaju odjednom: mehanizam izvodi **šezdeset i četiri uzastopna kruga**. U svakom krugu s pločicama izvodi tri operacije:

- **Vrtuljak (Tiovivo)** (rotacija). Pločice se kreću u krug: one s desne strane prelaze na lijevu. Nijedna se pločica ne gubi niti dodaje; jednostavno se preslaguju praveći puni krug na vrtuljku. To je jeftin i reverzibilan način redistribucije informacija.
- **Logički lijevak (Embudo Lógico)** (XOR). Pločice prolaze kroz lijevak koji ih uspoređuje dvije po dvije: ako su obje iste boje, izlazi bijela; ako su različite, izlazi crna. To je najjednostavnija operacija binarne logike, ali u kombinaciji s rotacijama na vrtuljku postaje iznimno moćna za miješanje informacija bez njihova gubitka.
- **Prelijevanje (Desborde)** (modularno zbrajanje). Rezultat se zbraja s *pločicom konstantnog potiska* preuzetom s javnog popisa od šezdeset i četiri konstante (treći korijeni prvih šezdeset i četiri prosta broja). Ako zbroj generira dodatne pločice koje ne stanu u predviđeni prostor od 32 pločice, te se suvišne pločice odbacuju. Stol ima mjesta za samo 32 pločice, niti jednu više.

Na kraju šezdeset i četvrtog kruga, svaka od pločica iz dionice vašeg teksta utjecala je na položaj osam glavnih pločica. Energija potiska proputovala je cijelim krugom.

Korak 5. Dodavanje sljedeće dionice (bez resetiranja). Ako je vaš tekst bio dug i preostala je još jedna dionica od 512 pločica za obradu, **krug se ne resetira**. Osam glavnih pločica ostaje onakvima kakvima ih je ostavila prva lavina, a druga dionica udara u njih kako bi pokrenula još šezdeset i četiri kruga. To je kao da dodajete novu sobu punu domina na kraj one koja je upravo pala: nered prve sobe u potpunosti uvjetuje kako će pasti druga.

Korak 6. Snimanje završne fotografije. Kada više nema dionica za obradu, lavina se zaustavlja. Gledamo završni položaj u kojem je ostalo osam glavnih pločica. Njihovu konfiguraciju prevodimo u kod od slova i brojeva u heksadecimalnom sustavu. Rezultat je niz od točno šezdeset i četiri znaka: to je vaš SHA-256 pečat.

Četiri svojstva proizlaze sama po sebi iz načina na koji je krug postavljen:

1. **Determinizam.** Isti tekst uvijek proizvodi istu završnu fotografiju, na bilo kojem računalu na svijetu. Nula nasumičnosti, nula iznenađenja.
2. **Efekt lavine.** Jedan dodani zarez, promijenjeno veliko slovo, zaboravljeni naglasak: fotografija postaje potpuno neprepoznatljiva. To je ekstremna osjetljivost koju smo već opisali na početku.
3. **Jednosmjernost.** S obzirom na završnu fotografiju, ne možete rekonstruirati izvorni tekst. Rotacije, lijevci i prelijevanja uništavaju sve smjerne informacije o tome *odakle je svaki bit došao* i čuvaju samo *što je ukupno zbrojeno*.
4. **Otpornost na kolizije.** U dvadeset i pet godina javne kriptoanalize, nitko nije uspio pronaći dva različita teksta čije se završne fotografije podudaraju. A poteškoća u tome je izvan računalnog dosega bilo koje razumno zamislive civilizacije.

Dodatak s kodom koji slijedi implementira upravo ovih šest koraka u Zigu. Sada ga možete čitati znajući što znači svaka operacija nad bitovima, umjesto da slijepo prihvaćate manipulacije.

Tehnički pojmovnik

Za čitatelja koji želi razumjeti što svaka operacija radi. Slobodno preskočite: članak je razumljiv i bez toga.

ASCII i Unicode — kako slova postaju brojevi. Računala ne vide slova; ona vide brojeve. Standard nazvan **ASCII** (*American Standard Code for Information Interchange*, iz 1963.) dodjeljuje svakom znaku na tipkovnici određeni broj: *A* je 65, *B* je 66, *a* je 97, *0* je 48, razmak je 32, zarez je 44. Moderni sustavi to proširuju s **Unicodeom**, koji dodjeljuje broj

svakom znaku svakog pisma na svijetu: ćirilici, arapskom, kineskom, japanskom, pa čak i emojijima. Kada utipkate znak ili otvorite tekstualnu datoteku, računalo čita pozadinski broj, a ne oblik na zaslonu. SHA-256 radi na tim brojevima, tretirajući bilo koji tekst kao dugačak niz znamenki. Zato istim algoritmom može zapečatiti članak na španjolskom, pjesmu na japanskom i binarnu datoteku.

XOR — usporednik bit po bit. XOR (izgovara se *„eksor”*, od engleskog *exclusive or*, „ekskluzivno ili”) jedna je od najjednostavnijih operacija koje računalo može izvesti s dva binarna broja. Uspoređuje dva bita položaj po položaj i vraća: **1** ako je točno jedan od njih 1 (jedan, ali ne oba), **0** ako su oba ista (oba 0 ili oba 1). Primjer: XOR brojeva `1010` i `1100` je `0110`. Ima značajno svojstvo: reverzibilan je — ako napravite XOR dva puta s istim ključem, vraćate se na izvornik. Zato je on „radni konj” (caballo de batalla) kriptografije: miješa bitove bez gubitka informacija, ali rezultat ne otkriva ništa o unosima ako ne poznajete jedan od njih.

Heksadecimalni sustav — brojanje u bazi 16. Gotovo svi svakodnevni brojevi koriste deset znamenki (0-9). Heksadecimalni sustav koristi šesnaest: uobičajenih 0-9 plus šest slova koja predstavljaju sljedeće vrijednosti: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Zašto šesnaest? Zato što računala razmišljaju u grupama od četiri bita, a četiri bita mogu predstavljati točno šesnaest različitih vrijednosti — tako jedan heksadecimalni znak čisto odgovara četirima bitovima. SHA-256 otisak (huella) mjeri 256 bitova, što je točno **64 heksadecimalna znaka**. Da ga pišemo uobičajenim dekadskim sustavom, zauzimao bi oko 78 znamenki i bio bi nepraktičniji. Izbor je estetski i kompaktan; pozadinski broj je isti.

Rotacija bitova — binarni vrtuljak (tiovivo binario). Zamislite red od sedam žarulja, neke su upaljene (1), a neke ugašene (0): `1 0 1 1 0 0 1`. Rotacija udesno za jedno mjesto sastoji se u tome da uzmete žarulju s krajnje desne strane, prebacite je na krajnju lijevu stranu i ostale pomaknete za jedno mjesto udesno: `1 1 0 1 1 0 0`. Nijedna se žarulja ne gubi niti dodaje: one jednostavno plešu u krug. SHA-256 koristi rotaciju bitova stotine puta u svakom izračunu; to je jeftin način preraspodjele informacija unutar stanja bez gubitaka.

Konstante *„nothing-up-my-sleeve”* — zašto dolaze iz prostih brojeva. Osam glavnih pločica i šezdeset i četiri konstante kruga u SHA-256 nisu odabrani slučajno. Dolaze iz drugih i trećih korijena prvih prostih brojeva. Zašto? Zato što su njihovi dizajneri željeli konstante *„bez ičega skrivenog u rukavu”* (*„nothing-up-my-sleeve”*): vrijednosti čije podrijetlo svatko može provjeriti. Da vam netko kaže *„vjeruj mi na riječ: koristi ovaj nasumični 32-bitni broj”*, s pravom biste posumnjali na skrivenu slabost ili „stražnja vrata”. Ali svatko s kalkulatorom može provjeriti da su prva 32 bita drugog korijena broja 2 upravo `0x6a09e667`. Vrijednosti su matematičke, javne i ponovljive: nikakva se skrivena zamka ne može uvući u recept.

Dodatak: SHA-256 u čitljivom kodu

Ovaj dodatak je za čitatelja koji želi vidjeti algoritam iznutra. To je didaktička implementacija u Zigu koja slijedi specifikaciju FIPS 180-4. To nije verzija koju koristi Solo2 — stvarna je u `std.crypto.hash.sha2.Sha256` standardne knjižnice Ziga, optimizirana i revidirana. Ali algoritam je isti: ono što vidite ovdje je, korak po korak, ono što se događa kada taj poziv od pet znakova izvrši svoj posao.

```
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
}
```

Svako ponovno pisanje u drugom jeziku koje slijedi istu strukturu — početne konstante, proširenje rasporeda (schedule expansion), šezdeset i četiri runde, akumulacija — proizvodi isti rezultat. Algoritam nema tajni: njegova vrijednost leži u činjenici da se gore navedena svojstva nastavljaju održavati nakon dva desetljeća javne kriptoanalize pred tisućama očiju.

Ako se vratite na dno ovog članka, vidjet ćete heksadekadski pečat od šezdeset i četiri znaka. To je SHA-256 teksta koji ste upravo pročitali, na ovom jeziku. Da smo preveli članak, pečat bi bio drugačiji; da se promijeni jedna riječ u hrvatskoj verziji, hrvatski bi se pečat promijenio. Pečat ne štiti sadržaj — za to služe drugi alati — već ga jedinstveno identificira. I to je, koliko god skromno zvučalo, dovoljno da nijedan korak u uredničkom lancu ne može promijeniti rečeno, a da se to ne primijeti. Sve ostalo — kriptiranje, potpisivanje, identificiranje — gradi se na ovoj jednostavnoj ideji.

Izvori i dodatno štivo

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, kolovoz 2015. Službena specifikacija obitelji SHA-2, uključujući SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, svibanj 2011. Normativna verzija za implementatore.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Poglavlja 5 i 6 pokrivaju hash funkcije i njihovu legitimnu i nelegitimnu upotrebu.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Praktičan primjer korištenja SHA-256 za ulančavanje blokova u strukturi koja je po konstrukciji nepromjenjiva.
- Uredba (EU) 910/2014 (eIDAS) — okvir kvalificiranih pružatelja usluga vremenskog žiga. SHA-256 je referentna funkcija za kvalificirane elektroničke potpise i pečate izdane u EU.

- Referentna implementacija u Zigu: `std.crypto.hash.sha2.Sha256` u službenom repozitoriju jezika (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). To je optimizirana i revidirana verzija koju zapravo koristi Solo2. Korisno za usporedbu s didaktičkom implementacijom iz dodatka.

← PrethodnoCUADERNOS_LIST_SCHREMS_TITLESljedeće →CUADERNOS_LIST_KILLSWITCH_TITLE

Nedavna čitanja

- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE

Ponesite ovaj članak sa sobom gdje god vam zatreba.

↓ Markdown ↓ Obični tekst ↓ PDF

Datoteka će se preuzeti na vaš uređaj. Od tamo je možete spremiti, uvesti u Solo2 ili dijeliti gdje god želite. Cuadernos ne odlučuje o odredištu umjesto vas.

Voštani pečat · SHA-256 d57b3c5d7ce63a253d4262aa34ca0edb106eeced833472f888a390e22df9e080

Cuadernos Lacre · Publikacija Menzuri Gestión S.L. ·
napisao R.Eugenio · uredio tim Solo2.