Ce este de fapt SHA-256

O amprentă matematică ce începe în șaizeci și patru de caractere și care se schimbă complet dacă se mișcă o singură virgulă din textul original. De ce îl numim sigiliu de ceară digital.

Ideea simplă din spatele numelui tehnic

Imaginează-ți că există o mașină cu o singură fantă și un singur ecran. Prin fantă introduci un text: un cuvânt, o frază, un roman întreg. Pe ecran apare, clipe mai târziu, o secvență de exact șaizeci și patru de caractere. Acea secvență, pentru cititorul profesionist, o numim *hash* sau *rezumat criptografic*; pentru cititorul general, o putem numi deocamdată o amprentă matematică a textului, așa cum amprenta digitală este a unei persoane.

Dacă introduci același text de două ori, mașina arată aceeași amprentă de ambele dăți. Dacă introduci un text ușor diferit — o singură virgulă mutată, o majusculă care devine minusculă — mașina arată o amprentă complet diferită de prima. Nu asemănătoare: diferită. Aceste două proprietăți împreună — determinismul și sensibilitatea — reprezintă ideea simplă. Tot restul SHA-256 este mecanismul care le face să funcționeze bine.

Merită spus de la început ce nu face mașina. Nu criptează textul. Nu îl ascunde. Nu îl salvează. Mașina se uită la text, calculează amprenta și uită textul. Amprenta nu permite reconstruirea textului care a produs-o; permite doar, dat fiind un text candidat, să verifici dacă acesta coincide sau nu cu originalul. De aceea spunem că este un rezumat *într-o singură direcție*: pleacă, nu se mai întoarce.

Un hash nu este același lucru cu criptarea

Confuzia este frecventă și merită clarificată: criptarea și hash-uirea sunt operațiuni diferite. Criptarea constă în transformarea unui text astfel încât doar posesorul cheii să îl poată readuce la forma sa originală. Hash-uirea constă în producerea unei amprente a textului din care textul original nu mai poate fi recuperat niciodată, nici cu cheie, nici fără ea. Prima este reversibilă prin design; a doua este ireversibilă prin design.

Consecința practică contează. Când o aplicație spune „îți păstrăm parola criptată”, există cineva care are cheia pentru a o decripta — în orice caz, aplicația însăși. Când o aplicație spune „îți păstrăm parola hash-uită”, aplicația însăși nu poate citi parola originală chiar dacă ar vrea; poate doar să verifice dacă ceea ce scrii tu produce din nou aceeași amprentă. Al doilea model, făcut bine, este de preferat primului pentru stocarea parolelor. Mai târziu vom vedea de ce „făcut bine” necesită ceva mai mult decât SHA-256 simplu.

Cele patru proprietăți care fac util un hash criptografic

O funcție hash care merită adjectivul *criptografic* îndeplinește patru proprietăți:

1. **Determinism.** Aceeași intrare produce întotdeauna aceeași amprentă.
2. **Efect de avalanșă.** O schimbare mică în intrare produce o amprentă complet diferită, fără nicio asemănare vizibilă cu cea anterioară.
3. **Rezistență la inversare.** Dată fiind o amprentă, nu este fezabil din punct de vedere computațional să găsești textul care a produs-o.
4. **Rezistență la coliziuni.** Nu este fezabil din punct de vedere computațional să găsești două texte diferite care să producă aceeași amprentă.

„Nu este fezabil din punct de vedere computațional” nu înseamnă „este matematic imposibil”. Înseamnă că costul în timp, energie și bani pentru a realiza acest lucru depășește cu ordine de mărime suma întregii capacități de calcul disponibile în mod rezonabil. Pentru SHA-256, această limită se măsoară în mii de miliarde de ani chiar și pentru cele mai optimiste abordări cu hardware specializat. Ceea ce, pentru scopurile practice ale cititorului, este același lucru cu „nu se poate”.

SHA-256, în mod concret

Numele spune totul. SHA este acronimul pentru *Secure Hash Algorithm*: algoritm de hash securizat. Numărul 256 indică dimensiunea amprentei în biți: două sute cincizeci și șase de biți, adică treizeci și doi de octeți, care, afișați în hexazecimal, sunt cele șaizeci și patru de caractere pe care cititorul le recunoaște deja. Standardul a fost publicat de NIST-ul american, organismul care normalizează acest tip de funcții, în 2001, ca parte a familiei SHA-2; versiunea actuală a standardului, FIPS 180-4, este din 2015.

Pentru cine nu are încă prezent ce sunt biții și octeții:

```
1 bit     →  0 sau 1       (un întrerupător: pornit sau oprit)
1 octet   →  8 biți        (256 de combinații posibile)
32 octeți →  256 biți      (amprenta SHA-256)
```

Numărul 256 de la sfârșitul numelui indică dimensiunea amprentei în biți. În hexazecimal — un sistem de numerație cu șaisprezece simboluri în loc de zece — acești 256 de biți încap în exact 64 de caractere. Aceștia sunt cei 64 de caractere pe care îi vedeți la subsolul fiecărui Cuaderno.

Dimensiunile merită un moment de atenție. Două sute cincizeci și șase de biți permit două la puterea două sute cincizeci și șase de valori diferite: un număr cu șaptezeci și opt de cifre zecimale, cu câteva ordine de mărime mai mare decât numărul estimat de atomi din universul observabil. Fiecare text din lume — fiecare carte, fiecare e-mail, fiecare mesaj — cade pe una dintre aceste valori. Probabilitatea ca două texte diferite să coincidă din întâmplare este, practic, imposibil de distins de zero.

Cum arată în cod

În Zig, limbajul în care scriem piesele care susțin Solo2, calcularea sigiliului SHA-256 al unui text arată astfel:

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

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

Tocmai am cerut bibliotecii standard din Zig să calculeze SHA-256 al textului între ghilimele. După apel, variabila *resumen* conține cei treizeci și doi de octeți care compun sigiliul în forma sa brută; când sunt afișați pe ecran în hexazecimal, aceștia sunt cele șaizeci și patru de caractere care apar la subsolul acestui articol. Dacă am schimba *Cuadernos Lacre* în *Cuadernos lacre* — o majusculă mai puțin — sigiliul s-ar schimba complet. Aceasta este, în cinci linii, proprietatea centrală care susține restul. Pentru cine vrea să vadă cum funcționează intern, la finalul articolului am inclus o versiune lizibilă a algoritmului cu comentarii pas cu pas.

De ce îl numim sigiliu de ceară

În corespondența europeană din secolele al XV-lea până la al XIX-lea, ceara de sigiliu închidea scrisoarea. O picătură de ceară topită, un sigiliu presat deasupra, iar scrisoarea rămânea marcată într-un mod irepetabil. Nu proteja conținutul de curioșii hotărâți — hârtia putea fi citită în lumină, ceara putea fi spartă — dar îl evidenția. Orice alterare a închiderii era vizibilă destinatarului înainte chiar de a deschide hârtia. Ceara nu împiedica dauna; o declara.

SHA-256 al corpului fiecărui Cuaderno îndeplinește aceeași funcție în versiunea sa digitală. Dacă un singur cuvânt din articol s-ar schimba între momentul în care a fost publicat și momentul în care îl citești, sigiliul hexazecimal de la subsolul textului nu ar mai coincide cu SHA-256 al textului pe care îl ai în față. Orice cititor cu cinci linii de cod ar putea verifica acest lucru. Publicația nu își poate rescrie istoria fără ca sigiliul să o dea de gol. Nu protejează împotriva daunei; o face verificabilă.

Ce nu este un hash

Patru utilizări sunt cerute uneori de la SHA-256, deşi nu îi corespund:

1. **Criptarea.** Un hash rezumă; nu ascunde. Dacă vrei ca textul să nu poată fi citit, trebuie să îl criptezi, nu să îl hashuieşti.
2. **Autentificarea autorului.** Un hash nu spune cine a scris textul, ci doar ce text a fost hashuit. Pentru a asocia paternitatea, este nevoie de o semnătură criptografică peste hash, nu de hash simplu.
3. **Stocarea parolelor.** Aici există o capcană care merită înțeleasă. SHA-256 este conceput să fie foarte rapid — ceea ce este bine pentru multe lucruri, dar rău pentru acesta. Un atacator cu hardware specializat poate testa miliarde de parole pe secundă împotriva unui hash SHA-256 până când o găseşte pe a ta. Pentru a salva parole, trebuie folosite funcţii de derivare a cheii deliberat lente, precum Argon2, scrypt sau bcrypt, combinate cu un *salt* (o dată aleatorie unică per utilizator, care împiedică ca două persoane cu aceeaşi parolă să aibă acelaşi hash).
4. **Citirea hash-ului ca identificator al autorului.** Nu este aşa. Un hash identifică conținutul. Dacă două persoane hashuiesc cuvântul *salut* cu SHA-256, ambele obțin același rezumat — și aceasta este proprietatea centrală, nu un defect: dacă ar fi rezumate diferite, nu am putea verifica coincidența între ceea ce a fost publicat și ceea ce a fost primit.

Unde apare SHA-256 în viaţa de zi cu zi

Deşi nu îl vezi, SHA-256 susține o bună parte din ceea ce foloseşti zilnic pe internet. Blockchain-ul Bitcoin este construit prin înlănțuirea SHA-256 al fiecărui bloc cu următorul; alterarea unui bloc trecut obligă la recalcularea întregului lanț ulterior. Git, sistemul cu care se versionează codul a jumătate din lume, identifică fiecare commit prin SHA-256 (în versiunile recente) sau prin predecesorul său SHA-1 (în versiunile mai vechi) al conținutului său complet. Certificatele HTTPS care verifică identitatea unui site web atunci când intri au o amprentă SHA-256 asociată. Descărcările de software sunt adesea însoțite de un SHA-256 publicat de dezvoltator pentru a verifica dacă fişierul nu a fost alterat pe drum. Și, aşa cum am spus, la subsolul fiecărui Cuadernos Lacre.

Pentru cititorul profesionist

Patru memento-uri operative pentru cine decide sau auditează sisteme:

1. Hash-ul nu este criptare. Dacă un furnizor confundă cei doi termeni în documentația sa tehnică, merită întrebat ce vrea să spună exact.
2. Pentru stocarea parolelor nu trebuie folosit niciodată SHA-256 simplu. SHA-256 este prea rapid pentru această sarcină (vezi punctul 3 din *Ce nu este un hash*). Standardul actual este **Argon2id**: lent prin design, configurabil în funcție de capacitatea serverului, combinat cu un *salt* aleatoriu diferit per utilizator.
3. Pentru integritatea documentelor — contracte, dosare, fişiere — SHA-256 rămâne standardul de referință. Este cel folosit de furnizorii de servicii de marcare temporală calificată din UE.
4. Pentru conservarea pe termen lung (zeci de ani), merită calculat și arhivat și un SHA-3 sau un SHA-512 alături de SHA-256; prudența criptografică recomandă să nu te bazezi pe o singură funcție în cazul arhivelor centenare.

Tehnic, această structură iterată — unde starea intermediară este conservată între blocurile de intrare — este cunoscută sub numele de construcție **Merkle-Damgård**, modelul pe care se bazează SHA-1, SHA-2 (inclusiv SHA-256) și multe alte funcții hash clasice. SHA-3, dimpotrivă, abandonează Merkle-Damgård în favoarea unei arhitecturi diferite numite *burete*.

Cum funcţionează SHA-256, pas cu pas, în cuvinte simple

Imaginează-ți că ai montat cel mai elaborat circuit de domino din lume: mii de piese, zeci de ramificații, poduri mecanice și rampe care traversează întreaga cameră, aşezate cu grijă piesă cu piesă.

Dacă dai un impuls primei piese, lanțul cade într-o secvență precisă și repetabilă. Aceeași configurație, același impuls inițial → model final identic de piese căzute, de fiecare dată.

Iată ce este interesant: mută **o singură piesă** jumătate de centimetru într-o parte înainte de a începe și atinge-o din nou. O rampă care trebuia să se activeze rămâne inertă, un pod nu cade, o ramificație diferită se declanşează. Modelul final de piese de pe podea este complet de nerecunoscut comparativ cu primul.

SHA-256 este, din punct de vedere matematic, acest circuit. Textul pe care îl scrii este poziția inițială a pieselor. Algoritmul este impulsul care eliberează cascada. Iar rezultatul final — ceea ce numim *hash* — este fotografia fixă a podelei când totul s-a oprit. Schimbă o singură virgulă din textul original și fotografia va fi radical diferită. Atât de simplu și atât de drastic.

Pasul 1. Traducerea textului în piese binare. Calculatoarele nu înțeleg literele; le traduc mai întâi în numere (ASCII) și numerele în binar (unu și zero). Fiecare literă se transformă în 8 piese albe sau negre: la *A* este `01000001`, la *B* este `01000010`, spațiul este `00100000`. Întregul tău text — un cuvânt, un contract, un roman — devine un șir lung de piese albe și negre.

Pasul 2. Completarea până la dimensiunea standard. Circuitul procesează șirul în *blocuri* de exact 512 piese. Dacă mesajul tău nu ajunge la un multiplu de 512, se adaugă o piesă de marcare (cea cu valoarea `10000000`) imediat după text și apoi zerouri până la completarea blocului. Ultimele 64 de poziții ale fiecărui bloc sunt rezervate pentru a nota lungimea originală a textului. Astfel, circuitul știe întotdeauna unde s-a terminat conținutul real și unde a început umplutura.

Pasul 3. Plasarea celor opt piese principale. Înainte de a începe, așezăm pe masă **opt piese principale** într-o poziție inițială precisă. Aceste opt piese nu sunt un secret: valoarea lor inițială este fixată de o regulă matematică publică (rădăcinile pătrate ale primelor opt numere prime — 2, 3, 5, 7, 11, 13, 17, 19 — și primii biți ai părții zecimale a fiecărei rădăcini). Toată lumea, în orice colț al planetei, începe cu aceleași opt piese principale în aceeași poziție. Destinul lor este să fie împinse și transformate de avalanșă.

Pasul 4. Marea avalanșă: șaizeci și patru de runde de impulsuri. Aici începe spectacolul. Primul bloc de 512 piese din textul tău este lovit de cele opt piese principale. Dar acestea nu cad dintr-o dată: mecanismul execută **șaizeci și patru de runde consecutive**. În fiecare rundă, efectuează trei operații cu piesele:

- **Caruselul** (rotația). Piesele se mișcă în cerc: cele din dreapta trec în stânga. Nicio piesă nu se pierde și niciuna nu se adaugă; pur și simplu se reordonează făcând o tură completă în carusel. Este un mod ieftin și reversibil de a redistribui informația.
- **Pâlnia Logică** (XOR). Piesele trec printr-o pâlnie care le compară două câte două: dacă ambele sunt de aceeași culoare, iese una albă; dacă sunt diferite, iese una neagră. Este cea mai simplă operație de logică binară, dar combinată cu rotațiile caruselului devine extrem de puternică pentru amestecarea informațiilor fără a le pierde.
- **Depășirea** (adunarea modulară). Rezultatul este adunat cu o *piesă de impuls constantă* extrasă dintr-o listă publică de șaizeci și patru de constante (rădăcinile cubice ale primelor șaizeci și patru de numere prime). Dacă adunarea generează piese suplimentare care nu încap în spațiul de 32 de piese prevăzut, acele piese excedentare sunt eliminate. Masa are spațiu doar pentru 32 de piese, niciuna în plus.

La finalul rundei șaizeci și patru, fiecare dintre piesele din blocul textului tău a influențat poziția celor opt piese principale. Energia impulsului a călătorit prin tot circuitul.

Pasul 5. Adăugarea următorului bloc (fără resetare). Dacă textul tău a fost lung și mai rămâne un alt bloc de 512 piese de procesat, **circuitul nu se resetează**. Cele opt piese principale rămân așa cum le-a lăsat prima avalanșă, iar al doilea bloc este lansat împotriva lor pentru a activa alte șaizeci și patru de runde. Este ca și cum ai adăuga o cameră nouă plină de piese de domino la capătul celei care tocmai a căzut: dezordinea primeia condiționează în întregime cum va cădea a doua.

Pasul 6. Realizarea fotografiei finale. Când nu mai sunt blocuri de procesat, avalanșa se oprește. Ne uităm la poziția finală în care au rămas cele opt piese principale. Traducem configurația lor într-un cod de litere și numere în sistem hexazecimal. Rezultatul este un șir de exact șaizeci și patru de caractere: acesta este sigiliul tău SHA-256.

Patru proprietăți decurg de la sine din modul în care este montat circuitul:

1. **Determinism.** Același text produce întotdeauna aceeași fotografie finală, pe orice calculator din lume. Zero aleatoriu, zero surprize.
2. **Efect de avalanșă.** O virgulă adăugată, o majusculă schimbată, o tildă uitată: fotografia rezultată este complet de nerecunoscut. Aceasta este sensibilitatea extremă pe care am descris-o deja la început.
3. **O singură direcție.** Având fotografia finală, nu poți reconstrui textul original. Rotațiile, pâlniile și depășirile distrug orice informație direcțională despre *de unde venea fiecare bit* și păstrează doar *ce s-a adunat în total*.
4. **Rezistență la coliziuni.** În douăzeci și cinci de ani de criptanaliză publică, nimeni nu a reușit să găsească două texte diferite ale căror fotografii finale să coincidă. Iar dificultatea de a face acest lucru este dincolo de capacitatea de calcul a oricărei civilizații rezonabil imaginabile.

Apendicele de cod care urmează implementează exact aceşti şase paşi în Zig. Acum îl poţi citi ştiind ce înseamnă fiecare operaţie de biţi, în loc să accepţi manipulările orbeşte.

Glosar tehnic

Pentru cititorul care doreşte să înţeleagă ce face fiecare operaţie. Poţi sări peste el liber: articolul poate fi înţeles şi fără acesta.

ASCII şi Unicode — cum literele devin numere. Calculatoarele nu văd litere; ele văd numere. Un standard numit **ASCII** (*American Standard Code for Information Interchange*, din 1963) atribuie fiecărui caracter de tastatură un număr specific: *A* este 65, *B* este 66, *a* este 97, *0* este 48, spaţiul este 32, virgula este 44. Sistemele moderne îl extind cu **Unicode**, care atribuie un număr fiecărui caracter din fiecare alfabet al lumii: chirilic, arab, chinez, japonez şi chiar emoji-uri. Când scrii un caracter sau deschizi un fişier text, calculatorul citeşte numărul de bază, nu forma de pe ecran. SHA-256 lucrează pe aceste numere, tratând orice text ca pe o secvenţă lungă de cifre. De aceea poate sigila un articol în spaniolă, un poem în japoneză şi un fişier binar cu acelaşi algoritm.
XOR — comparatorul bit cu bit. XOR (pronunţat *«exor»*, din englezul *exclusive or*, „sau exclusiv") este una dintre cele mai simple operaţii pe care un calculator le poate face cu două numere binare. Compară doi biţi poziţie cu poziţie şi returnează: **1** dacă exact unul dintre cei doi este 1 (unul, dar nu amândoi), **0** dacă cei doi sunt identici (ambii 0 sau ambii 1). Exemplu: XOR între `1010` şi `1100` este `0110`. Are o proprietate remarcabilă: este reversibilă — dacă faci XOR de două ori cu aceeaşi cheie, revii la original. De aceea este calul de povară al criptografiei: amestecă biţii fără a pierde informaţie, dar rezultatul nu dezvăluie nimic despre intrări dacă nu cunoşti una dintre ele.
Hexazecimal — numărarea în baza 16. Aproape toate numerele din viaţa de zi cu zi folosesc zece cifre (0-9). Hexazecimalul foloseşte şaisprezece: obişnuitele 0-9 plus şase litere care reprezintă valorile următoare: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. De ce şaisprezece? Deoarece calculatoarele gândesc în grupuri de patru biţi, iar patru biţi pot reprezenta exact şaisprezece valori diferite — astfel, un caracter hexazecimal corespunde perfect celor patru biţi. O amprentă SHA-256 măsoară 256 de biţi, ceea ce înseamnă exact **64 de caractere hexazecimale**. Dacă am scrie-o în zecimal obişnuit, ar ocupa aproximativ 78 de cifre şi ar fi mai incomodă. Alegerea este estetică şi compactă; numărul de bază este acelaşi.
Rotaţia biţilor — caruselul binar. Imaginează-ţi un rând de şapte becuri, unele aprinse (1) şi altele stinse (0): `1 0 1 1 0 0 1`. Rotaţia la dreapta cu o poziţie constă în a lua becul din dreapta de tot, a-l duce la marginea stângă şi a le deplasa pe celelalte cu o poziţie la dreapta: `1 1 0 1 1 0 0`. Niciun bec nu se pierde şi niciunul nu se adaugă: pur şi simplu dansează în cerc. SHA-256 utilizează rotaţia biţilor de sute de ori în fiecare calcul; este un mod ieftin şi fără pierderi de a redistribui informaţia în interiorul stării.
Constante *«nothing-up-my-sleeve»* — de ce provin din numere prime. Cele opt piese principale şi cele şaizeci şi patru de constante de rundă ale SHA-256 nu au fost alese la întâmplare. Ele provin din rădăcinile pătrate şi cubice ale primelor numere prime. De ce? Deoarece designerii săi doreau constante *„fără nimic ascuns în mânecă"*: valori a căror origine oricine să o poată verifica. Dacă cineva ţi-ar spune *„ai încredere în mine: foloseşte acest număr aleatoriu pe 32 de biţi"*, ai suspecta pe bună dreptate o slăbiciune ascunsă sau o poartă din spate. Dar oricine cu un calculator poate verifica faptul că primii 32 de biţi ai rădăcinii pătrate a lui 2 sunt `0x6a09e667`. Valorile sunt matematice, publice şi reproductibile: nicio capcană ascunsă nu se poate strecura în reţetă.

Anexă: SHA-256 în cod lizibil

Această anexă este pentru cititorul care doreşte să vadă algoritmul pe interior. Este o implementare didactică în Zig care urmează specificaţia FIPS 180-4. Nu este versiunea pe care o foloseşte Solo2 — cea reală se află în `std.crypto.hash.sha2.Sha256` din biblioteca standard Zig, optimizată şi auditată. Dar algoritmul este acelaşi: ceea ce vezi aici este, pas cu pas, ceea ce se întâmplă când acel apel de cinci caractere îşi execută sarcina.

```
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
}
```

Orice rescriere într-un alt limbaj care urmează aceeași structură — constante inițiale, expansiunea programului, șaizeci și patru de runde, acumulare — produce același rezultat. Algoritmul nu are secrete: valoarea sa constă în faptul că proprietățile enumerate mai sus continuă să se susțină după două decenii de criptanaliză publică realizată de mii de ochi.

Dacă te întorci la subsolul acestui articol, vei vedea un sigiliu hexazecimal de șaizeci și patru de caractere. Este SHA-256 al textului pe care tocmai l-ai citit, în această limbă. Dacă am traduce articolul, sigiliul ar fi altul; dacă s-ar schimba un cuvânt din versiunea spaniolă, sigiliul spaniol s-ar schimba. Sigiliul nu protejează conținutul — pentru asta există alte instrumente — ci îl identifică în mod unic. Și asta, oricât de modest ar suna, este suficient pentru ca nicio etapă a lanțului editorial să nu poată altera ceea ce s-a spus fără să se observe. Restul — criptarea, semnarea, identificarea — se construiește pe această idee simplă.

Surse și lecturi suplimentare

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, august 2015. Specificația oficială a familiei SHA-2, inclusiv SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, mai 2011. Versiune normativă pentru implementatori.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Capitolele 5 și 6 acoperă funcțiile hash și utilizările lor legitime și ilegitime.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Exemplu practic de utilizare a SHA-256 pentru înlănțuirea blocurilor într-o structură imuabilă prin construcție.
- Regulamentul (UE) 910/2014 (eIDAS) — cadrul furnizorilor de servicii de marcare temporală calificată. SHA-256 este funcția de referință pentru semnăturile și sigiliile electronice calificate emise în UE.
- Implementarea de referință în Zig: `std.crypto.hash.sha2.Sha256` în depozitul oficial al limbajului (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Este versiunea optimizată și auditată pe care o folosește de fapt Solo2. Utilă pentru a contrasta cu implementarea didactică din anexă.

← PrecedentCUADERNOS_LIST_SCHREMS_TITLEUrmător →CUADERNOS_LIST_KILLSWITCH_TITLE

Lecturi recente

- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE

Luați acest articol cu dumneavoastră oriunde aveți nevoie.

↓ Markdown ↓ Text simplu ↓ PDF

Fișierul se va descărca pe dispozitivul dumneavoastră. De acolo îl puteți salva, importa în Solo2 sau partaja oriunde doriți. Cuadernos nu decide destinația în locul dumneavoastră.

Sigiliu de ceară · SHA-256 45837d9586ca02057ed42b08bffc08fe0a3e04b03b496a288e563876f787bdfe

Cuadernos Lacre · O publicație a Menzuri Gestión S.L. ·
scrisă de R.Eugenio · editată de echipa Solo2.