# Čo je SHA-256 v skutočnosti

Matematický odtlačok, ktorý sa zmestí do šesťdesiatich štyroch znakov a úplne sa zmení, ak sa v pôvodnom texte posunie čo i len jedna čiarka. Prečo to nazývame digitálna vosková pečať.

## Jednoduchá myšlienka za technickým názvom

Predstavte si, že existuje stroj s jediným otvorom a jedinou obrazovkou. Cez otvor vložíte text: slovo, vetu, celý román. Na obrazovke sa o chvíľu neskôr objaví sekvencia presne šesťdesiatich štyroch znakov. Túto sekvenciu odborný čitateľ nazýva *hash* alebo *kryptografický odtlačok*; pre bežného čitateľa ju nateraz môžeme nazvať matematickým odtlačkom textu, rovnako ako je odtlačok prsta odtlačkom človeka.

Ak vložíte rovnaký text dvakrát, stroj ukáže obakrát rovnaký odtlačok. Ak vložíte mierne odlišný text — posunutá jedna čiarka, veľké písmeno zmenené na malé — stroj ukáže úplne iný odtlačok ako ten prvý. Nie podobný: iný. Tieto dve vlastnosti spoločne — determinizmus a citlivosť — sú tá jednoduchá myšlienka. Všetko ostatné na SHA-256 je mechanizmus, vďaka ktorému dobre fungujú.

Hneď na začiatku treba povedať, čo stroj nerobí. Nešifruje text. Neskrýva ho. Neukladá ho. Stroj sa pozrie na text, vypočíta odtlačok a text zabudne. Odtlačok neumožňuje rekonštruovať text, ktorý ho vytvoril; umožňuje iba to, že ak máme kandidátsky text, môžeme overiť, či sa zhoduje s originálom. Preto hovoríme, že ide o odtlačok *jedným smerom*: tam to ide, späť nie.

## Hash nie je to isté ako šifrovanie

Toto nedorozumenie je časté a treba ho objasniť: šifrovanie a hashovanie sú odlišné operácie. Šifrovanie spočíva v transformácii textu tak, aby ho do pôvodnej podoby mohol vrátiť iba držiteľ kľúča. Hashovanie spočíva vo vytvorení odtlačku textu, z ktorého už pôvodný text nemožno nikdy získať, a to ani s kľúčom, ani bez neho. Prvé je zámerne reverzibilné; druhé zámerne ireverzibilné.

Praktický dôsledok je dôležitý. Keď aplikácia povie „vaše heslo uchovávame zašifrované“, existuje niekto, kto má kľúč na jeho dešifrovanie — v každom prípade je to samotná aplikácia. Keď aplikácia povie „vaše heslo uchovávame zahashované“, samotná aplikácia nemôže prečítať pôvodné heslo, ani keby chcela; môže len skontrolovať, či to, ktoré napíšete, znova vytvorí rovnaký odtlačok. Druhý model, ak je urobený správne, je na ukladanie hesiel oveľa vhodnejší ako prvý. Neskôr uvidíme, prečo „urobený správne“ vyžaduje niečo viac ako len samotný SHA-256.

## Štyri vlastnosti, vďaka ktorým je kryptografický hash užitočný

Hashovacia funkcia, ktorá si zaslúži prívlastok *kryptografická*, spĺňa štyri vlastnosti:

1. **Determinizmus.** Rovnaký vstup vždy vytvorí rovnaký odtlačok.
2. **Lavínový efekt.** Malá zmena na vstupe vytvorí úplne iný odtlačok bez akejkoľvek viditeľnej podobnosti s predchádzajúcim.
3. **Odolnosť voči inverzii.** Majúc odtlačok, nie je výpočtovo možné nájsť text, ktorý ho vytvoril.
4. **Odolnosť voči kolíziám.** Nie je výpočtovo možné nájsť dva rôzne texty, ktoré vytvoria rovnaký odtlačok.

„Nie je výpočtovo možné“ neznamená „je to matematicky nemožné“. Znamená to, že náklady v podobe času, energie a peňazí na dosiahnutie tohto cieľa prevyšujú rádovo súčet celej rozumne dostupnej výpočtovej kapacity. V prípade SHA-

256 sa táto hranica meria v tisíckach miliárd rokov aj pri tých najoptimistickejších prístupoch so špecializovaným hardvérom. Čo je pre praktické účely čitateľa to isté ako „nedá sa to“.

## Konkrétne SHA-256

Názov hovorí za všetko. SHA je skratka pre *Secure Hash Algorithm*: bezpečný hashovací algoritmus. Číslo 256 označuje veľkosť odtlačku v bitoch: dvestopäťdesiatšesť bitov, teda tridsaťdva bajtov, ktoré pri zobrazení v hexadecimálnej sústave predstavujú tých šesťdesiatštyri znakov, ktoré čitateľ už pozná. Štandard zverejnil americký inštitút NIST, orgán, ktorý tieto typy funkcií štandardizuje, v roku 2001 ako súčasť rodiny SHA-2; súčasná verzia štandardu FIPS 180-4 je z roku 2015.

**Pre tých, ktorí ešte nevedia, čo sú bity a bajty:**

```
1 bit    →  0 alebo 1    (prepínač: zapnutý alebo vypnutý)
1 bajt   →  8 bitov      (256 možných kombinácií)
32 bajtov→  256 bitov    (odtlačok SHA-256)
```

Číslo 256 na konci názvu vyjadruje veľkosť odtlačku v bitoch. V hexadecimálnej sústave — číselnej sústave so šestnástimi symbolmi namiesto desiatich — sa týchto 256 bitov zmestí presne do 64 znakov. To je tých 64 znakov, ktoré vidíte na konci každého vydania Cuadernos Lacre.

Rozmery si zaslúžia chvíľu pozornosti. Dvestopäťdesiatšesť bitov umožňuje dva na dvestopäťdesiatšiestu rôznych hodnôt: číslo so sedemdesiatimi ôsmimi desatinnými miestami, čo je o niekoľko rádov viac, než je odhadovaný počet atómov v pozorovateľnom vesmíre. Každý text na svete — každá kniha, každý e-mail, každá správa — padne na jednu z týchto hodnôt. Pravdepodobnosť, že sa dva rôzne texty náhodou zhodujú, je pre praktické účely na nerozoznanie od nuly.

## Ako to vyzerá v kóde

V jazyku Zig, v ktorom píšeme komponenty podporujúce Solo2, vyzerá výpočet pečate SHA-256 textu takto:

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

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

Práve sme požiadali štandardnú knižnicu jazyka Zig, aby vypočítala SHA-256 textu v úvodzovkách. Po volaní premenná *resumen* obsahuje tridsaťdva bajtov, ktoré tvoria pečať v jej surovej podobe; keď sa zobrazia na obrazovke v hexadecimálnej sústave, je to tých šesťdesiatštyri znakov, ktoré sa objavujú na konci tohto článku. Keby sme zmenili *Cuadernos Lacre* na *Cuadernos lacre* — o jedno veľké písmeno menej — celá pečať by sa zmenila. To je v piatich riadkoch tá ústredná vlastnosť, ktorá drží pohromade všetko ostatné. Pre tých, ktorí chcú vidieť, ako to funguje zvnútra, uvádzame na konci článku čitateľnú verziu algoritmu s podrobnými komentármi.

## Prečo to nazývame vosková pečať

V európskej korešpondencii od pätnásteho do devätnásteho storočia uzatváral list pečatný vosk. Kvapka roztaveného vosku, pritlačená pečať a list bol neopakovateľne označený. Nechránilo to obsah pred odhodlaným sliedičom — papier sa dal čítať proti svetlu, vosk sa dal zlomiť — ale bolo to zjavné. Akékoľvek narušenie uzáveru bolo pre adresáta viditeľné ešte pred otvorením listu. Vosková pečať nezabránila poškodeniu; deklarovala ho.

SHA-256 jadra každého vydania Cuadernos plní v digitálnej verzii rovnakú funkciu. Ak by sa čo i len jedno slovo článku zmenilo od okamihu vydania po moment, keď si ho prečítate, hexadecimálna pečať na konci textu by sa už nezhodovala s SHA-256 textu, ktorý máte pred sebou. Mohol by to skontrolovať ktorýkoľvek čitateľ s piatimi riadkami kódu. Publikácia nemôže prepísať svoju históriu bez toho, aby ju pečať neprezradila. Nechráni pred poškodením; robí ho overiteľným.

## Čo hash nie je

Od SHA-256 sa niekedy vyžadujú štyri spôsoby použitia, ktoré mu neprislúchajú:

1. **Šifrovať.** Hash zhrnie, neskrýva. Ak chcete, aby bol text nečitateľný, musíte ho zašifrovať, nie zahashovať.
2. **Overovať autora.** Hash nehovorí, kto text napísal, iba to, aký text bol zahashovaný. Na priradenie autorstva je potrebný kryptografický podpis nad hashom, nie iba samotný hash.
3. **Uchovávať heslá.** Tu je pasca, ktorej je dobré porozumieť. SHA-256 je navrhnutý tak, aby bol veľmi rýchly — čo je dobré pre mnohé veci, ale zlé pre túto. Útočník so špecializovaným hardvérom dokáže vyskúšať miliardy hesiel za sekundu proti hashu SHA-256, kým nenájde to vaše. Na ukladanie hesiel by sa mali používať zámerne pomalé funkcie odvodenia kľúča, ako sú Argon2, scrypt alebo bcrypt, v kombinácii so *soľou* (jedinečným náhodným údajom pre každého používateľa, ktorý bráni dvom ľuďom s rovnakým heslom mať rovnaký hash).
4. **Čítať hash ako identifikátor autora.** Tak to nie je. Hash identifikuje obsah. Ak dvaja ľudia zahashujú slovo *ahoj* pomocou SHA-256, obaja získajú rovnaký odtlačok — a to je kľúčová vlastnosť, nie chyba: keby boli odtlačky odlišné, nemohli by sme skontrolovať zhodu medzi tým, čo bolo uverejnené, a tým, čo bolo prijaté.

## Kde sa SHA-256 objavuje vo vašom každodennom živote

Hoci ho nevidíte, SHA-256 podopiera veľkú časť toho, čo denne používate na internete. Blockchain siete Bitcoin sa vytvára reťazením SHA-256 každého bloku s ďalším; zmena starého bloku núti prepočítať celý nasledujúci reťazec. Git, systém na správu verzií kódu používaný polovicou sveta, identifikuje každý commit podľa SHA-256 (v novších verziách) alebo jeho predchodcu SHA-1 (v starších) jeho celého obsahu. Certifikáty HTTPS, ktoré pri vstupe overujú identitu webovej stránky, majú priradený odtlačok SHA-256. Sťahovanie softvéru je často sprevádzané hashom SHA-256 zverejneným vývojárom, aby ste si mohli overiť, že súbor nebol cestou zmenený. A ako sme už povedali, na konci každého vydania Cuadernos Lacre.

## Pre odborného čitateľa

Štyri operatívne pripomienky pre tých, ktorí rozhodujú o systémoch alebo ich auditujú:

1. Hash nie je šifrovanie. Ak poskytovateľ vo svojej technickej dokumentácii zamieňa tieto dva pojmy, je dobré sa opýtať, čo presne tým myslí.
2. Na uchovávanie hesiel by sa nikdy nemal používať samotný SHA-256. SHA-256 je na túto úlohu príliš rýchly (pozri bod 3 v *Čo hash nie je*). Súčasným štandardom je **Argon2id**: zo svojej podstaty pomalý, konfigurovateľný podľa výkonu servera, v kombinácii s inou náhodnou *soľou* pre každého používateľa.
3. Pre integritu dokumentov — zmlúv, záznamov, súborov — zostáva SHA-256 referenčným štandardom. Je to štandard, ktorý používajú kvalifikované elektronické časové pečiatky v EÚ.
4. Pri dlhodobom uchovávaní (desaťročia) sa odporúča popri SHA-256 vypočítať a archivovať aj SHA-3 alebo SHA-512; kryptografická obozretnosť velí nespoliehať sa na jednu funkciu pri archívoch uchovávaných celé storočie.

Technicky je táto iterovaná štruktúra – kde sa medzystav uchováva medzi vstupnými blokmi – známa ako **Merkleova-Damgårdova** konštrukcia, model, na ktorom sú založené SHA-1, SHA-2 (vrátane SHA-256) a mnohé ďalšie klasické hashovacie funkcie. SHA-3 naopak opúšťa Merkleovu-Damgårdovu konštrukciu v prospech inej architektúry nazývanej *špongia* (sponge).

## Ako funguje SHA-256, krok za krokom, v jednoduchých slovách

Predstavte si, že ste zostavili najprepracovanejší domino okruh na svete: tisíce kameňov, desiatky odbočiek, mechanické mostíky a rampy cez celú miestnosť, starostlivo rozložené kúsok po kúsku.

Ak cvaknete do prvého kameňa, reťaz padá v presnom a opakovateľnom poradí. Rovnaká zostava, rovnaký počiatočný impulz → identický výsledný vzor spadnutých kameňov, znova a znova.

Tu je to zaujímavé: posuňte **jediný kameň** o pol centimetra na stranu pred začiatkom a znova doň cvaknite. Rampa, ktorá sa mala aktivovať, ostane nehybná, mostík nespadne, spustí sa iná odbočka. Výsledný vzor kameňov na podlahe je úplne nerozoznateľný v porovnaní s tým prvým.

SHA-256 je matematicky tento okruh. Text, ktorý píšete, je počiatočná poloha kameňov. Algoritmus je impulz, ktorý uvoľní kaskádu. A konečný výsledok – to, čo nazývame *hash* – je nehybná fotografia podlahy, keď sa všetko zastavilo. Zmeňte jednu čiarku v pôvodnom texte a fotografia bude radikálne iná. Také jednoduché a také drastické.

**Krok 1. Preklad textu na binárne kamene.** Počítače nerozumejú písmenám; prekladajú ich najprv na čísla (ASCII) a čísla na binárne (jednotky a nuly). Každé písmeno sa premení na 8 bielych alebo čiernych kameňov: *A* je `01000001`, *B* je `01000010`, medzera je `00100000`. Celý váš text – slovo, zmluva, román – sa stane dlhým radom bielych a čiernych kameňov.

**Krok 2. Doplnenie do štandardnej veľkosti.** Okruh spracováva rad v *blokoch* s presne 512 kameňmi. Ak vaša správa nedosiahne násobok 512, hneď za text sa pridá označovací kameň (ten s hodnotou `10000000`) a potom nuly až do doplnenia bloku. Posledných 64 pozícií každého bloku je vyhradených na poznačenie pôvodnej dĺžky textu. Tak okruh vždy vie, kde skončil skutočný obsah a kde začala výplň.

**Krok 3. Umiestnenie ôsmich hlavných kameňov.** Pred začiatkom položíme na stôl **osem hlavných kameňov** do presnej počiatočnej polohy. Týchto osem kameňov nie je žiadnym tajomstvom: ich počiatočná hodnota je určená verejným matematickým pravidlom (odmocniny prvých ôsmich prvočísiel – 2, 3, 5, 7, 11, 13, 17, 19 – a prvé bity desatinnej časti každej odmocniny). Každý na svete, v ktoromkoľvek kúte planéty, začína s tými istými ôsmimi hlavnými kameňmi v rovnakej polohe. Ich osudom je byť posunuté a transformované lavínou.

**Krok 4. Veľká lavína: šesťdesiatštyri kôl postrčení.** Tu začína divadlo. Prvý blok 512 kameňov vášho textu narazí do ôsmich hlavných kameňov. Ale tie nespadnú naraz: mechanizmus vykoná **šesťdesiatštyri po sebe idúcich kôl**. V každom kole vykoná s kameňmi tri operácie:

- **Kolotoč** (rotácia). Kamene sa pohybujú v kruhu: tie sprava prechádzajú doľava. Žiadny kameň sa nestratí ani nepridá; jednoducho sa preusporiadajú tak, že urobia jedno celé kolo na kolotoči. Je to lacný a vratný spôsob prerozdelenia informácií.
- **Logický lievik** (XOR). Kamene prechádzajú lievikom, ktorý ich porovnáva po dvoch: ak sú obe rovnakej farby, vyjde biely; ak sú rôzne, vyjde čierny. Je to najjednoduchšia operácia binárnej logiky, ale v kombinácii s rotáciami kolotoča sa stáva nesmierne mocnou pri miešaní informácií bez ich straty.
- **Pretečenie** (modulárne sčítanie). Výsledok sa sčíta s *kameňom konštantného postrčenia* vybraným z verejného zoznamu šesťdesiatich štyroch konštánt (tretie odmocniny prvých šesťdesiatich štyroch prvočísiel). Ak sčítanie vygeneruje kamene navyše, ktoré sa nezmestia do určeného priestoru pre 32 kameňov, tie prebytočné sa odstránia. Stôl má miesto len pre 32 kameňov, ani o jeden viac.

Na konci šesťdesiateho štvrtého kola každý z kameňov v bloku vášho textu ovplyvnil polohu ôsmich hlavných kameňov. Energia impulzu prešla celým okruhom.

**Krok 5. Pridanie ďalšieho bloku (bez resetovania).** Ak bol váš text dlhý a zostáva ďalší blok 512 kameňov na spracovanie, **okruh sa neresetuje**. Osem hlavných kameňov zostane tak, ako ich nechala prvá lavína, a druhý blok sa spustí proti nim, aby aktivoval ďalších šesťdesiatštyri kôl. Es je ako pridať novú miestnosť plnú domina na koniec tej, ktorá práve spadla: neporiadok prvej úplne určuje, ako padne tá druhá.

**Krok 6. Vyhotovenie konečnej fotografie.** Keď už nezostávajú žiadne ďalšie bloky na spracovanie, lavína sa zastaví. Pozrieme sa na konečnú polohu, v ktorej zostalo osem hlavných kameňov. Preložíme ich konfiguráciu do kódu písmen a čísiel v hexadecimálnej sústave. Výsledkom je reťazec presne šesťdesiatich štyroch znakov: to je vaša pečať SHA-256.

Štyri vlastnosti vyplývajú samy o sebe z toho, ako je okruh zostavený:

1. **Determinizmus.** Rovnaký text vyprodukuje vždy rovnakú konečnú fotografiu na akomkoľvek počítači na svete. Nulová náhodnosť, nulové prekvapenia.
2. **Lavínový efekt.** Pridaná čiarka, zmenené veľké písmeno, zabudnutý dĺžeň: výsledná fotografia bude úplne nerozoznateľná. To je tá extrémna citlivosť, ktorú sme opísali už na začiatku.
3. **Jednosmernosť.** Z konečnej fotografie nemôžete zrekonštruovať pôvodný text. Rotácie, lieviky a pretečenia zničia všetku smerovú informáciu o tom, *odkiaľ prišiel každý bit* a zachovajú len to, *čo sa sčítalo celkovo*.
4. **Odolnosť voči kolíziám.** Za dvadsaťpäť rokov verejnej kryptografie sa nikomu nepodarilo nájsť dva rôzne texty, ktorých konečné fotografie by sa zhodovali. A náročnosť tejto úlohy je mimo výpočtovej kapacity akejkoľvek rozumne predstaviteľnej civilizácie.

Príloha s kódom, ktorá nasleduje, implementuje presne týchto šesť krokov v jazyku Zig. Teraz ho môžete čítať s vedomím, čo znamená každá bitová operácia, namiesto toho, aby ste slepo prijímali tieto manipulácie.

# Technický glosár

*Pre čitateľa, ktorý chce pochopiť, čo robí každá operácia. Môžete to pokojne preskočiť: článku porozumiete aj bez toho.*

**ASCII a Unicode — ako sa písmená menia na čísla.** Počítače nevidia písmená; vidia čísla. Štandard s názvom **ASCII** (*American Standard Code for Information Interchange*, z roku 1963) priraďuje každému znaku na klávesnici konkrétne číslo: *A* je 65, *B* je 66, *a* je 97, *0* je 48, medzera je 32, čiarka je 44. Moderné systémy ho rozširujú o **Unicode**, ktorý priraďuje číslo každému znaku každého písma na svete: cyriliky, arabčiny, čínštiny, japončiny a dokonca aj emodži. Keď píšete znak alebo otvoríte textový súbor, počítač číta číslo v pozadí, nie tvar na obrazovke. SHA-256 pracuje s týmito číslami a s akýmkoľvek textom narába ako s dlhou sekvenciou číslic. Preto môže rovnakým algoritmom zapečatiť článok v španielčine, báseň v japončine aj binárny súbor.
**XOR — bitový porovnávač.** XOR (vyslovované *„ex-or“*, z anglického *exclusive or*, „exkluzívne alebo“) je jedna z najjednoduchších operácií, ktoré počítač dokáže vykonať s dvoma binárnymi číslami. Porovnáva dva bity pozíciu po pozícii a vráti: **1**, ak práve jeden z nich je 1 (jeden, ale nie obaja), **0**, ak sú oba rovnaké (obaja 0 alebo obaja 1). Príklad: XOR pre `1010` a `1100` je `0110`. Má pozoruhodnú vlastnosť: je vratný – ak vykonáte XOR dvakrát s rovnakým kľúčom, vrátite sa k originálu. Preto je ťažným koňom kryptografie: mieša bity bez straty informácií, ale výsledok neprezrádza nič o vstupoch, ak nepoznáte jeden z nich.
**Hexadecimálna sústava — počítanie v základe 16.** Takmer všetky čísla v každodennom živote používajú desať číslic (0 – 9). Hexadecimálna sústava používa šestnásť: zvyčajných 0 – 9 plus šesť písmen, ktoré predstavujú nasledujúce hodnoty: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Prečo šestnásť? Pretože počítače rozmýšľajú v skupinách po štyroch bitoch a štyri bity môžu predstavovať presne šestnásť rôznych hodnôt – jeden hexadecimálny znak teda čistým spôsobom zodpovedá štyrom bitom. Odtlačok SHA-256 má 256 bitov, čo je presne **64 hexadecimálnych znakov**. Ak by sme ho zapísali v bežnej desiatkovej sústave, zabral by približne 78 číslic a bol by nepohodlnejší. Voľba je estetická a kompaktná; číslo v pozadí je rovnaké.
**Rotácia bitov — binárny kolotoč.** Predstavte si rad siedmich žiaroviek, niektoré svietia (1) a iné nie (0): `1 0 1 1 0 0 1`. Rotácia doprava o jednu pozíciu spočíva v tom, že vezmete žiarovku úplne vpravo, prenesiete ju na ľavý okraj a ostatné posuniete o jedno miesto doprava: `1 1 0 1 1 0 0`. Žiadna žiarovka sa nestratí ani nepridá: jednoducho tancujú v kruhu. SHA-256 používa rotáciu bitov pri každom výpočte stokrát; je to lacný a bezstratový spôsob prerozdelenia informácií v rámci stavu.
**Konštanty *„nothing-up-my-sleeve“* — prečo pochádzajú z prvočísiel.** Osem hlavných kameňov a šesťdesiatštyri konštánt kôl SHA-256 nebolo vybraných náhodne. Pochádzajú z druhých a tretích odmocnín prvých prvočísiel. Prečo? Pretože ich dizajnéri chceli konštanty *„bez ničoho v rukáve“* (*„nothing-up-my-sleeve“*): hodnoty, ktorých pôvod si môže overiť ktokoľvek. Ak by vám niekto povedal *„ver mi: použi toto náhodné 32-bitové číslo“*, oprávnene by ste podozrievali skrytú slabinu alebo zadné vrátka. Ale ktokoľvek s kalkulačkou si môže overiť, že prvých 32 bitov druhej odmocniny z 2 je `0x6a09e667`. Hodnoty sú matematické, verejné a reprodukovateľné: do receptu sa nemôže vlúdiť žiadny skrytý trik.

# Príloha: SHA-256 v čitateľnom kóde

Táto príloha je určená pre čitateľa, ktorý chce vidieť algoritmus zvnútra. Ide o didaktickú implementáciu v jazyku Zig, ktorá sa riadi špecifikáciou FIPS 180-4. Nie je to verzia, ktorú používa Solo2 — tá skutočná sa nachádza v `std.crypto.hash.sha2.Sha256` v štandardnej knižnici Zig, optimalizovaná a auditovaná. Algoritmus je však rovnaký: to, čo tu vidíte, krok za krokom ukazuje, čo sa stane, keď to päťznakové volanie vykoná svoju prácu.

```
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
}
```

Akékoľvek prepísanie do iného jazyka, ktoré dodržiava rovnakú štruktúru — počiatočné konštanty, expanzia plánu, šesťdesiatštyri kôl, akumulácia — prinesie rovnaký výsledok. Algoritmus nemá žiadne tajomstvá: jeho hodnota spočíva v tom, že vlastnosti uvedené vyššie pretrvávajú aj po dvoch desaťročiach verejnej kryptografickej analýzy tisíckami očí.

---

*Ak sa vrátite na koniec tohto článku, uvidíte šesťdesiatštyriznakovú hexadecimálnu pečať. Je to SHA-256 textu, ktorý ste práve dočítali v tomto jazyku. Keby sme článok preložili, pečať by bola iná; keby sa zmenilo jediné slovo v španielskej verzii, španielska pečať by sa zmenila. Pečať nechráni obsah — na to slúžia iné nástroje — ale jednoznačne ho identifikuje. A to, akokoľvek skromne to znie, stačí na to, aby žiadny krok v redakčnom reťazci nemohol nenápadne zmeniť to, čo bolo povedané. Zvyšok — šifrovanie, podpisovanie, identifikácia — sa buduje na tejto jednoduchej myšlienke.*

## Zdroje a ďalšie čítanie

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, august 2015. Oficiálna špecifikácia rodiny SHA-2 vrátane SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, máj 2011. Normatívna verzia pre implementátorov.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Kapitoly 5 a 6 sa zaoberajú hashovacími funkciami a ich legitímnym aj nelegitímnym využitím.

- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Praktický príklad použitia SHA-256 na reťazenie blokov do štruktúry z princípu nemennej.
- Nariadenie (EÚ) 910/2014 (eIDAS) — rámec pre kvalifikované elektronické časové pečiatky. SHA-256 je referenčnou funkciou pre kvalifikované elektronické podpisy a pečate vydávané v EÚ.
- Referenčná implementácia v jazyku Zig: `std.crypto.hash.sha2.Sha256` v oficiálnom repozitári jazyka (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Toto je optimalizovaná a auditovaná verzia, ktorú Solo2 skutočne používa. Užitočná na porovnanie s didaktickou implementáciou v prílohe.

← PredchádzajúciCUADERNOS_LIST_SCHREMS_TITLENasledujúci →CUADERNOS_LIST_KILLSWITCH_TITLE

## Nedávne čítanie

- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE

Vezmite si tento článok tam, kam potrebujete.

↓ Markdown ↓ Čistý text ↓ PDF

Súbor sa stiahne do vášho zariadenia. Odtiaľ si ho môžete uložiť, importovať do Solo2 alebo zdieľať, kdekoľvek chcete. Cuadernos za vás o cieli nerozhoduje.

Vosková pečať · SHA-256 a83de5b3883e9cf4ab504fa9509a4d3b4ebc5f500fa3b15864f61bd9ad1eef0e

Cuadernos Lacre · Publikácia spoločnosti Menzuri Gestión S.L. ·
napísal R.Eugenio · edituje tím Solo2.