Concepte · 19 de maig de 2026

← Anterior·Següent →

Què és realment SHA-256

Una empremta matemàtica que cap en seixanta-quatre caràcters i que canvia sencera si una sola coma del text original es mou. Per què en diem segell de lacre digital.

Per entender'ns: Imagina una màquina que llegeix un text qualsevol i torna una seqüència de 64 caràcters. Si el text entra idèntic, la seqüència surt idèntica. Si mous una sola coma, la seqüència és una altra completament. Aquesta seqüència és el lacre digital.

La idea senzilla darrere del nom tècnic

Imagina que existeix una màquina amb una sola ranura i una sola pantalla. Per la ranura introdueixes un text: una paraula, una frase, una novel·la sencera. A la pantalla apareix, instants després, una seqüència exactament de seixanta-quatre caràcters. Aquesta seqüència, al lector professional en diem *hash* o *resum criptogràfic*; al lector general, podem anomenar-la per ara una empremta matemàtica del text, com l'empremta dactilar ho és d'una persona.

Si introdueixes el mateix text dues vegades, la màquina mostra la mateixa empremta les dues vegades. Si introdueixes un text lleugerament diferent —una sola coma desplaçada, una majúscula que passa a minúscula— la màquina mostra una empremta completament diferent de la primera. No semblant: diferent. Aquestes dues propietats juntes —el determinisme i la sensibilitat— són la idea senzilla. Tota la resta del SHA-256 és la maquinària que les fa complir bé.

Convé dir des del principi el que la màquina no fa. No xifra el text. No l'oculta. No el guarda. La màquina mira el text, calcula l'empremta, i s'oblida del text. L'empremta no permet reconstruir el text que la va produir; només permet, donat un text candidat, comprovar si coincideix o no amb l'original. Per això diem que és un resum *d'una sola direcció*: s'hi va, no se'n torna.

Un hash no és el mateix que xifrar

La confusió és freqüent i convé aclarir-la: xifrar i hashejar són operacions distintes. Xifrar consisteix a transformar un text de manera que només el posseïdor de la clau pugui retornar-lo a la seva forma original. Hashejar consisteix a produir una empremta del text de la qual el text original no es pot recuperar mai, ni amb clau ni sense. La primera és reversible per disseny; la segona, irreversible per disseny.

La conseqüència pràctica importa. Quan una aplicació diu «guardem la teva contrasenya xifrada», hi ha algú que té la clau per desxifrar-la — l'aplicació mateixa, en qualsevol cas. Quan una aplicació diu «guardem la teva contrasenya hashejada», l'aplicació mateixa no pot llegir la contrasenya original encara que vulgui; només pot comprovar si la que tu escrius torna a produir la mateixa empremta. El segon model, fet bé, és molt preferible al primer per emmagatzemar contrasenyes. Més endavant veurem per què «fet bé» exigeix alguna cosa més que SHA-256 a seques.

Les quatre propietats que fan útil un hash criptogràfic

Una funció hash que mereix l'adjectiu *criptogràfic* compleix quatre propietats:

1. **Determinisme.** La mateixa entrada produeix sempre la mateixa empremta.
2. **Efecte avalanxa.** Un canvi petit en l'entrada produeix una empremta completament diferent, sense semblança visible amb l'anterior.
3. **Resistència a la inversió.** Donada una empremta, no és viable computacionalment trobar el text que la va produir.

4. **Resistència a col·lisions.** No és viable computacionalment trobar dos textos diferents que produeixin la mateixa empremta.

«No és viable computacionalment» no significa «és matemàticament impossible». Significa que el cost en temps, energia i diners d'aconseguir-ho excedeix en ordres de magnitud la suma de tota la capacitat de càlcul raonablement disponible. Per al SHA-256, aquesta cota es mesura en milers de bilions d'anys fins i tot per als plantejaments més optimistes amb maquinari especialitzat. La qual cosa, a efectes pràctics del lector, és el mateix que «no es pot».

SHA-256, en concret

El nom ho diu tot. SHA són les sigles de *Secure Hash Algorithm*: algorisme de hash segur. El número 256 indica la mida de l'empremta en bits: dos-cents cinquanta-sis bits, és a dir trenta-dos bytes, que mostrats en hexadecimal són els seixanta-quatre caràcters que el lector ja reconeix. L'estàndard el va publicar el NIST nord-americà, l'organisme que normalitza aquest tipus de funcions, el 2001 com a part de la família SHA-2; la versió vigent de l'estàndard, FIPS 180-4, és de 2015.

Per a qui encara no té present què són bits i bytes:

```
1 bit    →  0 o 1        (un interruptor: encès o apagat)
1 byte   →  8 bits       (256 combinacions possibles)
32 bytes →  256 bits     (l'empremta SHA-256)
```

El número 256 al final del nom diu la mida de l'empremta en bits. En hexadecimal —un sistema de numeració amb setze símbols en lloc de deu— aquests 256 bits caben en exactament 64 caràcters. Aquests són els 64 caràcters que veus al peu de cada Cuaderno.

Les dimensions mereixen un instant. Dos-cents cinquanta-sis bits permeten dos elevat a dos-cents cinquanta-sis valors diferents: un número amb setanta-vuit dígits decimals, diversos ordres de magnitud major que el número estimat d'àtoms en l'univers observable. Cada text del món —cada llibre, cada correu electrònic, cada missatge— cau sobre un d'aquests valors. La probabilitat que dos textos diferents coincideixin per atzar és, a efectes pràctics, indistingible de zero.

Com es veu en codi

En Zig, llenguatge en què escrivim les peces que sostenen Solo2, calcular el segell SHA-256 d'un text es veu així:

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

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

Acabem de demanar-li a la biblioteca estàndard de Zig que calculi el SHA-256 del text entre cometes. Després de la crida, la variable *resumen* conté els trenta-dos bytes que componen el segell en la seva forma crua; quan es mostren en pantalla en hexadecimal, són els seixanta-quatre caràcters que apareixen al peu d'aquest article. Si canviéssim *Cuadernos Lacre* per *Cuadernos lacre* —una majúscula menys— el segell canviaria sencer. Aquesta és, en cinc línies, la propietat central que sosté la resta. Per a qui vulgui veure com funciona internament, al final de l'article incloem una versió llegible de l'algorisme amb comentaris pas a pas.

Per què en diem segell de lacre

En la correspondència europea dels segles quinze al dinou, el lacre tancava la carta. Una gota de cera fosa, un segell pressionat a sobre, i la carta quedava marcada de forma irrepetible. No protegia el contingut del xafarder decidit —el paper es podia llegir a contrallum, el lacre es podia trencar— però sí que l'evidenciava. Qualsevol alteració del tancament era visible per al destinatari abans fins i tot d'obrir el paper. El lacre no impedia el dany; el declarava.

El SHA-256 del cos de cada Cuaderno compleix la mateixa funció en la seva versió digital. Si una sola paraula de l'article canviés entre el moment en què es va publicar i el moment en què tu el llegeixes, el segell hexadecimal al peu del text ja no coincidiria amb el SHA-256 del text que tens davant. Qualsevol lector amb cinc línies de codi podria comprovar-ho. La publicació no pot reescriure la seva història sense que el segell la delati. No protegeix contra el dany; el fa verificable.

El que un hash no és

Quatre usos se li demanen de vegades al SHA-256 que no li corresponen:

1. **Xifrar.** Un hash resumeix; no oculta. Si vols que el text no es pugui llegir, necessites xifrar-lo, no hashejar-lo.
2. **Autenticar l'autor.** Un hash no diu qui va escriure el text, només quin text es va hashejar. Per associar autoria cal una signatura criptogràfica a sobre del hash, no el hash a seques.
3. **Emmagatzemar contrasenyes.** Aquí hi ha una trampa que convé entendre. El SHA-256 està dissenyat per ser molt ràpid —la qual cosa és bona per a moltes coses, però dolenta per a aquesta. Un atacant amb maquinari especialitzat pot provar milers de milions de contrasenyes per segon contra un hash SHA-256 fins a trobar la teva. Per guardar contrasenyes cal usar funcions de derivació de clau deliberadament lentes com Argon2, scrypt o bcrypt, combinades amb una *sal* (una dada aleatòria única per usuari, que evita que dues persones amb la mateixa contrasenya tinguin el mateix hash).
4. **Llegir el hash com a identificador de l'autor.** No ho és. Un hash identifica el contingut. Si dues persones hashegen la paraula *hola* amb SHA-256, totes dues obtenen el mateix resum — i això és la propietat central, no un defecte: si fossin resums diferents, no podríem comprovar coincidència entre el que s'ha publicat i el que s'ha rebut.

On apareix SHA-256 en el teu dia a dia

Encara que no ho vegis, el SHA-256 sosté bona part del que uses a diari a internet. La cadena de blocs de Bitcoin es construeix encadenant SHA-256 de cada bloc al següent; alterar un bloc passat obliga a recalcular tota la cadena posterior. Git, el sistema amb què es versiona el codi de mig món, identifica cada confirmació pel SHA-256 (en versions recents) o pel seu predecessor SHA-1 (en versions més antigues) del seu contingut complet. Els certificats HTTPS que verifiquen la identitat d'un lloc web quan entres porten una empremta SHA-256 associada. Les descàrregues de programari s'acompanyen sovint d'un SHA-256 publicat pel desenvolupador perquè verifiquis que el fitxer no s'ha alterat pel camí. I, com hem dit, al peu de cada Cuadernos Lacre.

Per al lector professional

Quatre recordatoris operatius per a qui decideix o audita sistemes:

1. Hash no és xifrat. Si un proveïdor confon els dos termes en la seva documentació tècnica, convé preguntar què vol dir exactament.
2. Per emmagatzemar contrasenyes mai no s'ha d'usar SHA-256 a seques. SHA-256 és massa ràpid per a aquesta tasca (vegeu punt 3 de *El que un hash no és*). L'estàndard actual és **Argon2id**: lent per disseny, configurable segons la capacitat del servidor, combinat amb una *sal* aleatòria diferent per usuari.
3. Per a integritat de documents —contractes, expedients, fitxers— SHA-256 continua sent l'estàndard de referència. És el que usen els segelladors temporals qualificats a la UE.
4. Per a conservació a llarg termini (decennis) convé calcular i arxivar també un SHA-3 o un SHA-512 al costat del SHA-256; la prudència criptogràfica recomana no recolzar-se en una sola funció durant arxius centenaris.

Tècnicament, aquesta estructura iterada —on l'estat intermedi es conserva entre blocs d'entrada— es coneix com una construcció de **Merkle-Damgård**, el patró en què es basen SHA-1, SHA-2 (inclòs SHA-256) i moltes altres funcions hash clàssiques. SHA-3, per contra, abandona Merkle-Damgård en favor d'una arquitectura distinta anomenada *esponja*.

Com funciona SHA-256, pas a pas, en paraules planeres

Imagina que has muntat el circuit de dominó més elaborat del món: milers de fitxes, desenes de bifurcacions, ponts mecànics i rutes que creuen tota l'habitació, curosament col·locades peça a peça.

Si dones un toc a la primera fitxa, la cadena cau en una seqüència precisa i repetible. Mateix muntatge, mateix toc inicial → idèntic patró final de fitxes caigudes, un cop i un altre.

Aquí hi ha l'interès: mou **una sola fitxa** mig centímetre a un costat abans de començar i torna a tocar. Una ruta que s'havia d'activar es queda inerta, un pont no cau, una bifurcació distinta es dispara. El patró final de fitxes a terra és completament irrecognoscible comparat amb el primer.

SHA-256 és matemàticament aquest circuit. El text que escrius és la posició inicial de les fitxes. L'algoritme és el toc que llibera la cascada. I el resultat final —el que anomenem *hash*— és la foto fixa del terra quan s'ha aturat tot. Canvia una sola coma del text original i la foto serà radicalment distinta. Així de simple, i així de dràstic.

Pas 1. Traduir el text a fitxes binàries. Els ordinadors no entenen de lletres; les tradueixen primer a números (ASCII) i els números a binari (uns i zeros). Cada lletra es converteix en 8 fitxes blanques o negres: la *A* és `01000001`, la *B* es `01000010`, l'espai és `00100000`. El teu text sencer —una paraula, un contracte, una novel·la— es torna una llarga fila de fitxes blanques i negres.

Pas 2. Replenar fins a la mida estàndard. El circuit processa la fila en *trams* d'exactament 512 fitxes. Si el teu missatge no arriba a un múltiple de 512, s'afegeix una fitxa marcadora (la del valor `10000000`) just després del text i després zeros fins a completar el tram. Les últimes 64 posicions de cada tram es reserven per anotar la longitud original del text. Així el circuit sempre sap on ha acabat el contingut real i on ha començat el rebliment.

Pas 3. Col·locar les vuit fitxes mestres. Abans de començar, situem sobre la mesa **vuit fitxes mestres** en una posició inicial precisa. Aquestes vuit fitxes no són cap secret: el seu valor inicial està fixat per una regla matemàtica pública (les arrels quadrades dels vuit primers números primers —2, 3, 5, 7, 11, 13, 17, 19— i els primers bits de la part decimal de cada arrel). Tothom, a qualsevol racó del planeta, comença amb les mateixes vuit fitxes mestres en la mateixa posició. El seu destí és ser empeses i transformades per l'allau.

Pas 4. La gran allau: seixanta-quatre rondes d'empentes. Aquí comença l'espectacle. El primer tram de 512 fitxes del teu text es fa xocar contra les vuit fitxes mestres. Però no cauen de cop: el mecanisme executa **seixanta-quatre rondes consecutives**. En cada ronda fa tres operacions amb les fitxes:

- **El Cavallet** (rotació). Les fitxes es mouen en cercle: les de la dreta passen a l'esquerra. Cap fitxa es perd ni s'afegeix; simplement es reordenen donant una volta completa al cavallet. És una manera barata i reversible de redistribuir la informació.
- **L'Embut Lògic** (XOR). Les fitxes passen per un embut que les compara de dues en dues: si totes dues són del mateix color, en surt una de blanca; si són distintes, en surt una de negra. És l'operació més senzilla de la lògica binària, però combinada amb les rotacions del cavallet es torna poderosíssima per barrejar informació sense perdre-la.
- **El Desbordament** (suma modular). El resultat se suma amb una *fitxa d'empenta constant* portada d'una llista pública de seixanta-quatre constants (les arrels cúbiques dels seixanta-quatre primers números primers). Si la suma genera fitxes extres que no caben en l'espai de 32 fitxes previst, aquestes fitxes sobrants es descarten. La mesa només té espai per a 32 fitxes, ni una més.

Al final de la ronda seixanta-quatre, cadascuna de les fitxes del tram del teu text ha influït en la posició de les vuit fitxes mestres. L'energia de l'empenta ha viatjat per tot el circuit.

Pas 5. Afegir el següent tram (sense reiniciar). Si el teu text era llarg i queda un altre tram de 512 fitxes per processar, **el circuit no es reinicia**. Les vuit fitxes mestres es queden tal com les ha deixat la primera allau, i el segon tram es llança contra elles per activar unes altres seixanta-quatre rondes. És com afegir una habitació nova plena de dominós al final de la que acaba de caure: el desordre de la primera condiciona enterament com caurà la segona.

Pas 6. Fer la foto final. Quan ja no queden més trams per processar, l'allau s'atura. Mirem la posició final en què han quedat les vuit fitxes mestres. Traduïm la seva configuració a un codi de lletres i números en sistema hexadecimal. El resultat és una cadena d'exactament seixanta-quatre caràcters: aquest és el teu segell SHA-256.

Quatre propietats cauen per si soles de com està muntat el circuit:

1. **Determinisme.** El mateix text produeix sempre la mateixa foto final, en qualsevol ordinador del món. Zero aleatorietat, zero sorpreses.
2. **Efecte allau.** Una coma afegida, una majúscula canviada, una tilde oblidada: la foto resulta completament irrecognoscible. Aquesta és la sensibilitat extrema que ja hem descrit al principi.
3. **Una sola direcció.** Donada la foto final, no pots reconstruir el text original. Les rotacions, els embuts i els desbordaments destrueixen tota la informació direccional sobre *d'on venia cada bit* i conserven només *què s'ha sumat en total*.
4. **Resistència a col·lisions.** En vint-i-cinc anys de criptoanàlisi pública, ningú ha aconseguit trobar dos textos distints les fotos finals dels quals coincideixin. I la dificultat de fer-ho està fora de l'abast computacional de qualsevol civilització raonablement imaginable.

L'apèndix de codi que segueix implementa exactament aquests sis passos en Zig. Ara pots llegir-lo sabent què significa cada operació de bits, en comptes d'acceptar les manipulacions a cegues.

Glossari tècnic

Per al lector que vulgui entendre què fa cada operació. Salta-te'l lliurement: l'article se segueix entenent sense ell.

ASCII i Unicode — com les lletres es tornen números. Els ordinadors no veuen lletres; veuen números. Un estàndard anomenat **ASCII** (*American Standard Code for Information Interchange*, de 1963) assigna a cada caràcter de teclat un número específic: la *A* és 65, la *B* és 66, la *a* és 97, el *0* és 48, l'espai és 32, la coma és 44. Els sistemes moderns l'estenen amb **Unicode**, que assigna un número a cada caràcter de cada alfabet del món: ciríl·lic, àrab, xinès, japonès, i fins i tot emojis. Quan escrius un caràcter o obres un fitxer de text, l'ordinador llegeix el número de fons, no la forma en pantalla. SHA-256 treballa sobre aquests números, tractant qualsevol text com una seqüència llarga de xifres. Per això pot segellar un article en espanyol, un poema en japonès i un arxiu binari amb el mateix algoritme.
XOR — el comparador bit a bit. XOR (pronunciat *«exor»*, de l'anglès *exclusive or*, «o exclusiu») és una de les operacions més senzilles que un ordinador pot fer amb dos números binaris. Compara dos bits posició per posició i retorna: **1** si exactament un dels dos és 1 (un però no els dos), **0** si els dos són iguals (tots dos 0 o tots dos 1). Exemple: XOR de `1010` i `1100` és `0110`. Té una propietat notable: és reversible —si fas XOR dos cops amb la mateixa clau, tornes a l'original —. Per això és el cavall de batalla de la criptografia: barreja bits sense perdre informació, però el resultat no revela res sobre les entrades si no en coneixes una.
Hexadecimal — comptar en base 16. Quasi tots els números del dia a dia usen deu dígits (0-9). L'hexadecimal n'usa setze: els habituals 0-9 més sis lletres que representen els valors següents: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Per què setze? Perquè els ordinadors pensen en grups de quatre bits, i quatre bits poden representar exactament setze valors distints —així, un caràcter hexadecimal correspon netament a quatre bits—. Una empremta SHA-256 mesura 256 bits, la qual cosa són exactament **64 caràcters hexadecimals**. Si l'escrivíssim en decimal corrent, ocuparia uns 78 dígits i resultaria més incòmoda. L'elecció és estètica i compacta; el número de fons és el mateix.
Rotació de bits — el cavallet binari. Imagina una fila de set bombetes, unes enceses (1) i altres apagades (0): `1 0 1 1 0 0 1`. Rotar a la dreta una posició consisteix a agafar la bombeta de la dreta del tot, portar-la a l'extrem esquerre i desplaçar les altres un lloc a la dreta: `1 1 0 1 1 0 0`. Cap bombeta es perd ni s'afegeix: simplement ballen en cercle. SHA-256 utilitza la rotació de bits centenars de vegades en cada càlcul; és una manera barata i sense pèrdues de redistribuir la informació dins de l'estat.
Constants *«nothing-up-my-sleeve»* — per què provenen de números primers. Les vuit fitxes mestres i les seixanta-quatre constants de ronda de SHA-256 no es van triar a l'atzar. Provenen de les arrels quadrades i cúbiques dels primers números primers. Per què? Perquè els seus dissenyadors volien constants *«sense res sota la màniga»*: valors l'origen dels quals qualsevol pugui verificar. Si algú et digués *«fia't de mi: usa aquest número aleatori de 32 bits»*, sospitaries raonablement d'una debilitat oculta o d'una porta del darrere. Però qualsevol amb una calculadora pot comprovar que els primers 32 bits de l'arrel quadrada de 2 són `0x6a09e667`. Els valors són matemàtics, públics i reproduïbles: cap trampa oculta es pot colar en la recepta.

Apèndix: SHA-256 en codi llegible

Aquest apèndix és per al lector que vulgui veure l'algorisme per dins. És una implementació didàctica en Zig que segueix l'especificació FIPS 180-4. No és la versió que usa Solo2 —la real és a `std.crypto.hash.sha2.Sha256` de la biblioteca estàndard de Zig, optimitzada i auditada—. Però l'algorisme és el mateix: el que veus aquí és, pas a pas, el que passa quan aquella crida de cinc caràcters executa la seva feina.

```
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
}
```

Qualsevol reescriptura en un altre llenguatge que segueixi la mateixa estructura —constants inicials, expansió de l'schedule, seixanta-quatre rondes, acumulació— produeix el mateix resultat. L'algorisme no té secrets: el seu valor resideix en el fet que les propietats enumerades més amunt continuen sostenint-se després de dues dècades de criptoanàlisi pública sobre milers d'ulls.

Si tornes al peu d'aquest article, veuràs un segell hexadecimal de seixanta-quatre caràcters. És el SHA-256 del text que acabes de llegir, en aquest idioma. Si traduíssim l'article, el segell seria un altre; si canviés una paraula de la versió catalana, el segell català canviaria. El segell no protegeix el contingut —per a això hi ha altres eines— sinó que l'identifica unívocament. I això, per modest que soni, n'hi ha prou perquè cap pas de la cadena editorial pugui alterar el que s'ha dit sense que es noti. Tota la resta —xifrar, signar, identificar— es construeix a sobre d'aquesta idea senzilla.

Nota editorial: quan aquests Cuadernos nomenen empreses o productes, no és per acusar. Els qui els construeixen fan treballs que milions de persones usen i aprecien. El que assenyalem és estructural — el model, no la marca. Les marques

apareixen com a exemple perquè són les que el lector reconeix.

Fonts i lectura addicional

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, agost de 2015. Especificació oficial de la família SHA-2, incloent-hi SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, maig de 2011. Versió normativa per a implementadors.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Capítols 5 i 6 cobreixen funcions hash i els seus usos legítims i il·legítims.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Exemple pràctic de l'ús de SHA-256 per encadenar blocs en una estructura immutable per construcció.
- Reglament (UE) 910/2014 (eIDAS) — marc dels segelladors temporals qualificats. SHA-256 és la funció de referència per a les signatures i segells electrònics qualificats emesos a la UE.
- Implementació de referència en Zig: `std.crypto.hash.sha2.Sha256` en el repositori oficial del llenguatge (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). És la versió optimitzada i auditada que de fet usa Solo2. Útil per contrastar amb la implementació didàctica de l'apèndix.

← AnteriorSchrems II, cinc anys desprésSegüent →Kill switch i la captura institucional

Lectures recents

- Anàlisi · 18 de maig de 2026 Privadesa real vs aparent: les preguntes que convé fer-se
- Anàlisi · 18 de maig de 2026 Self-hosting com a pràctica professional
- Concepte · 18 de maig de 2026 Les 24 paraules: què és una identitat criptogràfica

Emporta't aquest article on el necessitis.

↓ Markdown ↓ Text pla ↓ PDF

L'arxiu es descarrega al teu dispositiu. Des d'allà pots guardar-lo, importar-lo a Solo2, o compartir-lo on vulguis. Cuadernos no decideix el destí per tu.

Segell de lacre · SHA-256 6a54cd4b8c443602aea8b713f73b72d26218d6b600d58e1c1767a6b5e8376d7f

Cuadernos Lacre · Una publicació de Menzuri Gestión S.L. ·
escrita per R.Eugenio · editada per l'equip de Solo2.