Concepto · 19 de maio de 2026

← Anterior·Seguinte →

# Que é realmente SHA-256

Unha pegada matemática que cabe en sesenta e catro caracteres e que cambia enteira se unha soa coma do texto orixinal se move. Por que lle chamamos selo de lacre dixital.

## A idea sinxela detrás do nome técnico

Imaxina que existe unha máquina cunha soa rañura e unha soa pantalla. Pola rañura introduces un texto: unha palabra, unha frase, unha novela enteira. Na pantalla aparece, instantes despois, unha secuencia exactamente de sesenta e catro caracteres. Esa secuencia, ao lector profesional chamámoslle *hash* ou *resumo criptográfico*; ao lector xeral, podemos chamala por agora unha pegada matemática do texto, como a pegada dactilar o é dunha persoa.

Se introduces o mesmo texto dúas veces, a máquina mostra a mesma pegada as dúas veces. Si introduces un texto lixeiramente distinto —unha soa coma desprazada, unha maiúscula que pasa a minúscula— a máquina mostra unha pegada completamente distinta da primeira. Non parecida: distinta. Esas dúas propiedades xuntas —o determinismo e a sensibilidade— son a idea sinxela. Todo lo demais do SHA-256 é a maquinaria que as fai cumprir ben.

Convén dicir desde o principio o que a máquina non fai. Non cifra o texto. Non o oculta. No o garda. A máquina mira o texto, calcula a pegada, e esquécese do texto. A pegada non permite reconstruír o texto que a produciu; só permite, dado un texto candidato, comprobar se coincide ou non co orixinal. Por iso dicimos que é un resumo *dunha soa dirección*: vaise, non se volve.

## Un hash non é o mesmo que cifrar

A confusión é frecuente e convén despexala: cifrar e hashear son operacións distintas. Cifrar consiste en transformar un texto de forma que só o posuidor da clave poida devolvelo á súa forma orixinal. Hashear consiste en producir unha pegada do texto da que o texto orixinal non se pode recuperar nunca, ni con clave nin sen ela. A primeira é reversible por deseño; a segunda, irreversible por deseño.

A consecuencia práctica importa. Cando unha aplicación di «gardamos o teu contrasinal cifrado», hai alguén que ten a clave para descifralo — a aplicación mesma, en calquera caso. Cando unha aplicación di «gardamos o teu contrasinal hasheado», a aplicación mesma non pode ler o contrasinal orixinal aínda que quixese; só pode comprobar se a que ti escribes volve producir a mesma pegada. O segundo modelo, feito ben, é moi preferible ao primeiro para almacenar contrasinais. Máis adiante veremos por que «feito ben» esixe algo máis que SHA-256 a secas.

## As catro propiedades que fan útil un hash criptográfico

Unha función hash que merece o adxectivo *criptográfico* cumpre catro propiedades:

1. **Determinismo.** A mesma entrada produce sempre a mesma pegada.
2. **Efecto avalancha.** Un cambio pequeno na entrada produce unha pegada completamente distinta, sen parecido visible coa anterior.
3. **Resistencia á inversión.** Dada unha pegada, non é viable computacionalmente atopar o texto que a produciu.
4. **Resistencia a colisións.** Non é viable computacionalmente atopar dous textos distintos que produzan a mesma pegada.

«Non é viable computacionalmente» non significa «é matematicamente imposible». Significa que o custo en tempo, enerxía e diñeiro de conseguilo excede en ordes de magnitude a suma de toda a capacidade de cómputo razoablemente

dispoñible. Para SHA-256, esa cota mídese en miles de billóns de anos incluso para os planteamentos máis optimistas con hardware especializado. O cal, a efectos prácticos do lector, é o mesmo que «non se pode».

## SHA-256, en concreto

O nome dío todo. SHA son as siglas de *Secure Hash Algorithm*: algoritmo de hash seguro. O número 256 indica o tamaño da pegada en bits: douscentos cincuenta e seis bits, é dicir trinta e dous bytes, que mostrados en hexadecimal son os sesenta e catro caracteres que o lector recoñece xa. O estándar publicouno o NIST estadounidense, o organismo que normaliza este tipo de funcións, en 2001 como parte da familia SHA-2; a versión vixente do estándar, FIPS 180-4, é de 2015.

**Para quen aínda non ten presente que son bits e bytes:**

```
1 bit    →  0 ó 1        (un interruptor: aceso ou apagado)
1 byte   →  8 bits       (256 combinacións posibles)
32 bytes →  256 bits     (a pegada SHA-256)
```

O número 256 ao final do nome di o tamaño da pegada en bits. En hexadecimal —un sistema de numeración con dezaseis símbolos en lugar de dez— eses 256 bits caben en exactamente 64 caracteres. Eses son os 64 caracteres que ves ao pé de cada Cuaderno.

As dimensións merecen un instante. Douscentos cincuenta e seis bits permiten dous elevado a douscentos cincuenta e seis valores distintos: un número con setenta e oito díxitos decimais, varios ordes de magnitude maior que o número estimado de átomos no universo observable. Cada texto do mundo —cada libro, cada correo electrónico, cada mensaxe— cae sobre un deses valores. A probabilidade de que dous textos distintos coincidan por azar é, a efectos prácticos, indistinguible de cero.

## Como se ve en código

En Zig, linguaxe na que escribimos as pezas que sosteñen Solo2, calcular o selo SHA-256 dun texto vese así:

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

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

Acabamos de pedirlle á biblioteca estándar de Zig que calcule o SHA-256 do texto entre comiñas. Despois da chamada, a variable *resumo* contén os trinta e dous bytes que compoñen o selo na súa forma crúa; cando se mostran en pantalla en hexadecimal, son os sesenta e catro caracteres que aparecen ao pé deste artigo. Se cambiáramos *Cuadernos Lacre* por *Cuadernos lacre* —unha maiúscula menos— o selo cambiaría enteiro. Esa é, en cinco liñas, a propiedade central que sostén o resto. Para quen queira ver como funciona internamente, ao final do artigo incluímos unha versión lexible do algoritmo con comentarios paso a paso.

## Por que lle chamamos selo de lacre

Na correspondencia europea dos séculos quince ao dezanove, o lacre pechaba a carta. Unha gota de cera derretida, un selo presionado encima, e a carta quedaba marcada de forma irrepetible. Non protexía o contido do fisgón decidido —o papel podíase ler ao trasluz, o lacre podíase romper— pero si o evidenciaba. Calquera alteración do peche era visible ao destinatario antes incluso de abrir o papel. O lacre non impedía o dano; declarábao.

O SHA-256 do corpo de cada Cuaderno cumpre a mesma función na súa versión dixital. Se unha soa palabra do artigo cambiara entre o momento en que se publicou e o momento en que ti o les, o selo hexadecimal ao pé do texto xa non coincidiría co SHA-256 do texto que tes diante. Calquera lector con cinco liñas de código podería comprobalo. A publicación non pode reescribir a súa historia sen que o selo a delate. No protexe contra o dano; faio verificable.

## O que un hash non é

Catro usos pídenselle ás veces a SHA-256 que non lle corresponden:

1. **Cifrar.** Un hash resume; no oculta. Se queres que o texto non se poida ler, necesitas cifralo, non hashearlo.
2. **Autenticar ao autor.** Un hash no di quen escribiu o texto, só que texto se hasheou. Para asociar autoría fai falta unha firma criptográfica encima do hash, non o hash a secas.
3. **Almacenar contrasinais.** Aquí hai unha trampa que convén entender. SHA-256 está deseñado para ser moi rápido —o cal é bo para moitas cousas, pero malo para esta. Un atacante con hardware especializado pode probar miles de millóns de contrasinais por segundo contra un hash SHA-256 ata dar co teu. Para gardar contrasinais hai que usar funcións de derivación de clave deliberadamente lentas como Argon2, scrypt o bcrypt, combinadas cunha *sal* (un dato aleatorio único por usuario, que evita que dúas persoas co mesmo contrasinal teñan o mesmo hash).
4. **Ler o hash como identificador do autor.** No o é. Un hash identifica o contido. Se dúas persoas hashean a palabra *ola* con SHA-256, as dúas obteñen o mesmo resumo — e iso é a propiedade central, non un defecto: se foran resumos distintos, non poderiamos comprobar coincidencia entre o publicado e o recibido.

## Onde aparece SHA-256 no teu día a día

Aínda que non o vexas, SHA-256 sostén boa parte do que usas a diario en internet. A cadea de bloques de Bitcoin constrúese encadeando SHA-256 de cada bloque ao seguinte; alterar un bloque pasado obriga a recalcular toda a cadea posterior. Git, o sistema co que se versiona o código de medio mundo, identifica cada confirmación polo SHA-256 (en versións recentes) ou polo seu predecesor SHA-1 (en versións máis antigas) do seu contido completo. Os certificados HTTPS que verifican a identidade dun sitio web cando entras levan unha pegada SHA-256 asociada. As descargas de software acompáñanse a miúdo dun SHA-256 publicado polo desenvolvedor para que verifiques que o arquivo non se alterou polo camiño. E, como dixemos, ao pé de cada Cuaderno Lacre.

## Para o lector profesional

Catro recordatorios operativos para quen decide ou audita sistemas:

1. Hash non é cifrado. Se un provedor confunde os dous termos na súa documentación técnica, convén preguntar que quere dicir exactamente.
2. Para almacenar contrasinais nunca se debe usar SHA-256 a secas. SHA-256 é demasiado rápido para esta tarefa (ver punto 3 de *O que un hash non é*). O estándar actual é **Argon2id**: lento por deseño, configurable segundo a capacidade do servidor, combinado cunha *sal* aleatoria distinta por usuario.
3. Para integridade de documentos —contratos, expedientes, arquivos— SHA-256 segue a ser o estándar de referencia. É o que usan os seladores temporais cualificados na UE.
4. Para conservación a longo prazo (decenios) convén calcular e arquivar tamén un SHA-3 ou un SHA-512 xunto ao SHA-256; a prudencia criptográfica recomenda non apoiarse nunha soa función durante arquivos centenarios.

Tecnicamente, esta estrutura iterada —onde o estado intermedio se conserva entre bloques de entrada— coñécese como unha construción de **Merkle-Damgård**, o patrón no que se basean SHA-1, SHA-2 (incluído SHA-256) e moitas outras funcións hash clásicas. SHA-3, pola contra, abandona Merkle-Damgård en favor dunha arquitectura distinta chamada *esponxa*.

## Como funciona SHA-256, paso a paso, en palabras sinxelas

Imaxina que montaches o circuíto de dominó máis elaborado do mundo: miles de fichas, decenas de bifurcacións, pontes mecánicas e ramplas que cruzan toda a habitación, coidadosamente colocadas peza a peza.

Se dás un toque á primeira ficha, a cadea cae nunha secuencia precisa e repetible. Mesmo montaxe, mesmo toque inicial → idéntico patrón final de fichas caídas, unha e outra vez.

Aquí está o interesante: move **unha soa ficha** medio centímetro a un lado antes de empezar e volve tocar. Unha rampla que debía activarse queda inerte, unha ponte non cae, unha bifurcación distinta dispárase. O patrón final de fichas no chan é completamente irrecoñecible comparado co primeiro.

SHA-256 é matematicamente este circuíto. O texto que escribes é a posición inicial das fichas. O algoritmo é o toque que libera a fervenza. E o resultado final —o que chamamos *hash*— é a foto fixa do chan cando se detivo todo. Cambia unha

soa coma do texto orixinal e a foto será radicalmente distinta. Así de simple, e así de drástico.

**Paso 1. Traducir o texto a fichas binarias.** Os ordenadores non entenden de letras; tradúcenas primeiro a números (ASCII) e os números a binario (uns e ceros). Cada letra convértese en 8 fichas brancas ou negras: a *A* é `01000001`, a *B* é `01000010`, o espazo é `00100000`. O teu texto enteiro —unha palabra, un contrato, unha novela— vólvese unha longa fila de fichas brancas e negras.

**Paso 2. Reencher ata o tamaño estándar.** O circuíto procesa a fila en *tramos* de exactamente 512 fichas. Se a túa mensaxe non chega a un múltiplo de 512, engádese unha ficha marcadora (a do valor `10000000`) xusto despois do texto e logo ceros ata completar o tramo. As últimas 64 posicións de cada tramo resérvanse para anotar a lonxitude orixinal do texto. Así o circuíto sempre sabe onde acabou o contido real e onde empezou o recheo.

**Paso 3. Colocar as oito fichas mestras.** Antes de empezar, situamos sobre a mesa **oito fichas mestras** nunha posición inicial precisa. Estas oito fichas non son ningún segredo: o seu valor inicial está fixado por unha regra matemática pública (as raíces cadradas dos oito primeiros números primos —2, 3, 5, 7, 11, 13, 17, 19— e os primeiros bits da parte decimal de cada raíz). Todo o mundo, en calquera rincón do planeta, empeza coas mesmas oito fichas mestras na mesma posición. O seu destino é ser empurradas e transformadas pola fervenza.

**Paso 4. A gran fervenza: sesenta e catro roldas de empuxóns.** Aquí empeza o espectáculo. O primeiro tramo de 512 fichas do teu texto faise chocar contra as oito fichas mestras. Pero non caen de golpe: o mecanismo executa **sesenta e catro roldas consecutivas**. En cada rolda fai tres operacións coas fichas:

- **O Tiovivo** (rotación). As fichas móvense en círculo: as da dereita pasan á esquerda. Ningunha ficha se perde nin se engade; simplemente reordénanse dando unha volta completa ao tiovivo. É unha maneira barata e reversible de redistribuír a información.
- **O Funil Lóxico** (XOR). As fichas pasan por un funil que as compara de dúas en dúas: se as dúas son da mesma cor, sae unha branca; se son distintas, sae unha negra. É a operación máis sinxela da lóxica binaria, pero combinada coas rotacións do tiovivo vólvese poderosísima para mesturar información sen perdela.
- **O Desborde** (suma modular). O resultado súmase cunha *ficha de empuxón constante* traída dunha lista pública de sesenta e catro constantes (as raíces cúbicas dos sesenta e catro primeiros números primos). Se a suma xera fichas extras que non caben no espazo de 32 fichas previsto, esas fichas sobrantes descártanse. A mesa só ten espazo para 32 fichas, nin unha máis.

Ao final da rolda sesenta e catro, cada unha das fichas do tramo do teu texto influíu na posición das oito fichas mestras. A enerxía do empuxón viaxou por todo o circuíto.

**Paso 5. Engadir o seguinte tramo (sen reiniciar).** Se o teu texto era longo e queda outro tramo de 512 fichas por procesar, **o circuíto non se reinicia**. As oito fichas mestras quédanse tal e como as deixou a primeira fervenza, e o segundo tramo lánzase contra elas para activar outras sesenta e catro roldas. É como engadir unha habitación nova chea de dominós ao final da que acaba de caer: a desorde da primeira condiciona enteiramente como caerá a segunda.

**Paso 6. Facer a foto final.** Cando xa non quedan máis tramos por procesar, a fervenza detense. Miramos a posición final na que quedaron as ochos fichas mestras. Traducimos a súa configuración a un código de letras e números en sistema hexadecimal. O resultado é unha cadea de exactamente sesenta e catro caracteres: ese é o teu selo SHA-256.

Catro propiedades caen por si soas de como está montado o circuíto:

1. **Determinismo.** O mesmo texto produce sempre a mesma foto final, en calquera ordenador do mundo. Cero aleatoriedade, cero sorpresas.
2. **Efecto fervenza.** Unha coma engadida, unha maiúscula cambiada, un til esquecido: a foto resulta completamente irrecoñecible. Esta é a sensibilidade extrema que xa describimos ao principio.
3. **Unha soa dirección.** Dada a foto final, non podes reconstruír o texto orixinal. As rotacións, os funís e os desbordes destrúen toda a información direccional sobre *de onde viña cada bit* e conservan só *que se sumou en total*.
4. **Resistencia a colisións.** En vinte e cinco anos de criptoanálise público, ninguén conseguiu atopar dous textos distintos cuxas fotos finais coincidan. E a dificultade de facelo está fóra do alcance computacional de calquera civilización razoablemente imaxinable.

O apéndice de código que segue implementa exactamente estes seis pasos en Zig. Agora podes lelo sabendo que significa cada operación de bits, en vez de aceptar as manipulacións a cegas.

# Glosario técnico

*Para o lector que queira entender que fai cada operación. Sáltao libremente: o artigo séguese entendendo sen el.*

**ASCII e Unicode — como as letras se volven números.** Os ordenadores non ven letras; ven números. Un estándar chamado **ASCII** (*American Standard Code for Information Interchange*, de 1963) asigna a cada carácter de teclado un número específico: a *A* é 65, a *B* é 66, a *a* es 97, o *0* es 48, o espazo es 32, a coma es 44. Os sistemas modernos esténdeno con **Unicode**, que asigna un número a cada carácter de cada alfabeto do mundo: cirílico, árabe, chinés, xaponés, e incluso emojis. Cando escribes un carácter ou abres un ficheiro de texto, o ordenador le o número de fondo, non a forma en pantalla. SHA-256 traballa sobre estes números, tratando calquera texto como unha secuencia longa de cifras. Por iso pode selar un artigo en español, un poema en xaponés e un arquivo binario co mesmo algoritmo.
**XOR — o comparador bit a bit.** XOR (pronunciado *«exor»*, do inglés *exclusive or*, «ou exclusivo») é unha das operacións máis sinxelas que un ordenador pode facer con dous números binarios. Compara dous bits posición por posición e devolve: **1** se exactamente un dos dous é 1 (un pero non os dous), **0** se os dous son iguais (ambos 0 ou ambos 1). Exemplo: XOR de `1010` e `1100` é `0110`. Ten unha propiedade notable: é reversible —se fas XOR dúas veces coa mesma clave, volves ao orixinal—. Por iso é o cabalo de batalla da criptografía: mestura bits sen perder información, pero o resultado non revela nada sobre as entradas se non coñeces unha delas.
**Hexadecimal — contar en base 16.** Case todos os números do día a día usan dez díxitos (0-9). O hexadecimal usa dezaseis: os habituais 0-9 máis seis letras que representan os valores seguintes: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Por que dezaseis? Porque os ordenadores pensan en grupos de catro bits, e catro bits poden representar exactamente dezaseis valores distintos —así, un carácter hexadecimal corresponde limpamente a catro bits—. Unha pegada SHA-256 mide 256 bits, o que son exactamente **64 caracteres hexadecimais**. Se a escribísemos en decimal corrente, ocuparía uns 78 díxitos e resultaría máis incómoda. A elección é estética e compacta; o número de fondo é o mesmo.
**Rotación de bits — o tiovivo binario.** Imaxina unha fila de sete bombillas, unhas acendidas (1) e outras apagadas (0): `1 0 1 1 0 0 1`. Rotar á dereita unha posición consiste en tomar a bombilla da dereita de todo, levala ao extremo esquerdo e desprazar as demais un sitio á dereita: `1 1 0 1 1 0 0`. Ningunha bombilla se perde nin se engade: simplemente bailan en círculo. SHA-256 utiliza a rotación de bits centos de veces en cada cálculo; é unha maneira barata e sen perdas de redistribuír a información dentro do estado.
**Constantes *«nothing-up-my-sleeve»* — por que proveñen de números primos.** As oito fichas mestras e as sesenta e catro constantes de rolda de SHA-256 non se elixiron ao azar. Proveñen das raíces cadradas e cúbicas dos primeiros números primos. Por que? Porque os seus deseñadores querían constantes *«sen nada baixo a manga»*: valores cuxa orixe calquera poida verificar. Se alguén che dixese *«fíate de min: usa este número aleatorio de 32 bits»*, sospeitarías razoablemente dunha debilidade oculta ou dunha porta traseira. Pero calquera cunha calculadora pode comprobar que os primeiros 32 bits da raíz cadrada de 2 son `0x6a09e667`. Os valores son matemáticos, públicos e reproducibles: ningunha trampa oculta pode colarse na receita.

# Apéndice: SHA-256 en código lexible

Este apéndice é para o lector que queira ver o algoritmo por dentro. É unha implementación didáctica en Zig que segue a especificación FIPS 180-4. Non é a versión que usa Solo2 —a real está en `std.crypto.hash.sha2.Sha256` da biblioteca estándar de Zig, optimizada e auditada—. Pero o algoritmo é o mesmo: o que ves aquí é, paso a paso, lo que ocorre cando aquela chamada de cinco caracteres executa o seu traballo.

```
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
}
```

Calquera reescritura noutra linguaxe que siga a mesma estrutura —constantes iniciais, expansión do schedule, sesenta e catro roldas, acumulación— produce o mesmo resultado. O algoritmo non ten segredos: o seu valor reside en que as propiedades enumeradas máis arriba seguen sosténdose despois de dúas décadas de criptoanálise pública sobre miles de ollos.

---

*Se volves ao pé deste artigo, verás un selo hexadecimal de sesenta e catro caracteres. É o SHA-256 do texto que acabas de ler, neste idioma. Se traducíramos o artigo, o selo sería outro; se cambiara unha palabra da versión española, o selo español cambiaría. O selo non protexe o contido —para iso están outras ferramentas— senón que o identifica univocamente. E iso, por modesto que soe, abonda para que ningún paso da cadea editorial poida alterar o dito sen que se note. O demais —cifrar, firmar, identificar— constrúese encima desta idea sinxela.*

## Fontes e lectura adicional


- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, agosto de 2015. Especificación oficial da familia SHA-2, incluíndo SHA-256.

- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, maio de 2011. Versión normativa para implementadores.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Capítulos 5 e 6 cobren funcións hash e os seus usos lexítimos e ilexítimos.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Exemplo práctico do uso de SHA-256 para encadear bloques nunha estrutura inmutable por construción.
- Regulamento (UE) 910/2014 (eIDAS) — marco dos seladores temporais cualificados. SHA-256 é a función de referencia para as firmas e selos electrónicos cualificados emitidos na UE.
- Implementación de referencia en Zig: `std.crypto.hash.sha2.Sha256` no repositorio oficial da linguaxe (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). É a versión optimizada e auditada que de feito usa Solo2. Útil para contrastar coa implementación didáctica do apéndice.



← AnteriorCUADERNOS_LIST_SCHREMS_TITLESeguinte →CUADERNOS_LIST_KILLSWITCH_TITLE


## Lecturas recentes


- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE


Leva este artigo onde o necesites.

↓ Markdown ↓ Texto plano ↓ PDF

O arquivo descárgase no teu dispositivo. Desde aí podes gardalo, importalo a Solo2, o compartilo onde queiras. Cuadernos no decide o destino por ti.

Selo de lacre · SHA-256 3bb35d444281279a256ca1e7d203d2f72bd5f6fe1224755de73c65955fcce6be


Cuadernos Lacre · Unha publicación de Menzuri Gestión S.L. ·
escrita por R.Eugenio · editada polo equipo de Solo2.