Zer den benetan SHA-256

Hirurogeita lau karakteretan sartzen den aztarna matematikoa, jatorrizko testuko koma bakar bat mugitzen bada osorik aldatzen dena. Zergatik deitzen diogun lakre-zigilu digitala.

Izen teknikoaren atzean dagoen ideia sinplea

Imajinatu zirrikitu bakarra eta pantaila bakarra dituen makina bat dagoela. Zirrikitutik testu bat sartzen duzu: hitz bat, esaldi bat, nobela oso bat. Pantailan, une batzuk geroago, zehazki hirurogeita lau karaktereko sekuentzia bat agertzen da. Sekuentzia horri, irakurle profesionalarentzat, *hash* edo *laburpen kriptografikoa* deitzen diogu; irakurle orokorrarentzat, testuaren aztarna matematikoa deitu diezaiokegu oraingoz, hatz-marka pertsona batena den bezala.

Testu bera bi aldiz sartzen baduzu, makinak aztarna bera erakusten du bi aldiz. Testu apur bat desberdina sartzen baduzu —koma bakar bat mugituta, letra larri bat xehe bihurtuta—, makinak lehenengoaren guztiz desberdina den aztarna erakusten du. Ez antzekoa: desberdina. Bi propietate horiek batera —determinismoa eta sentikortasuna— dira ideia sinplea. SHA-256-ren gainerako guztia horiek ondo betearazten dituen makineria da.

Hasieratik esan behar da makinak zer ez duen egiten. Ez du testua zifratzen. Ez du ezkutatzen. Ez du gordetzen. Makinak testua begiratzen du, aztarna kalkulatzen du eta testuaz ahazten da. Aztarnak ez du ahalbidetzen hura sortu zuen testua berreraikitzea; hautagai den testu bat emanda, jatorrizkoarekin bat datorren ala ez egiaztatzea besterik ez du ahalbidetzen. Horregatik diogu *norabide bakarreko* laburpena dela: badoa, ez da itzultzen.

Hash bat ez da zifratzea bezalakoa

Nahasmena maiz gertatzen da eza argitzea komeni da: zifratzea eta hasheatzea eragiketa desberdinak dira. Zifratzea testu bat eraldatzean datza, gakoaren jabeak soilik bere jatorrizko formara itzul dezan. Hasheatzea testuaren aztarna bat sortzean datza, eta aztarna horretatik ezin da inoiz jatorrizko testua berreskuratu, ez gakoarekin ez gakorik gabe. Lehenengoa itzulgarria da diseinuz; bigarrena, itzulezina diseinuz.

Ondorio praktikoak garrantzia du. Aplikazio batek «zure pasahitza zifratuta gordetzen dugu» esaten duenean, badago norbait gakoa duena hura deszifratzeko —aplikazioa bera, edonola ere—. Aplikazio batek «zure pasahitza hasheatuta gordetzen dugu» esaten duenean, aplikazioak berak ezin du jatorrizko pasahitza irakurri nahi izanda ere; idazten duzunak aztarna bera sortzen duela egiaztatu besterik ezin du egin. Bigarren eredua, ondo eginez gero, askoz hobetsiagoa da pasahitzak gordetzeko lehenengoa baino. Geroago ikusiko dugu zergatik «ondo eginda» horrek SHA-256 hutsa baino zerbait gehiago eskatzen duen.

Hash kriptografiko bat erabilgarri egiten duten lau propietateak

Kriptografiko adjektiboa merezi duen hash funtzio batek lau propietate betetzen ditu:

1. **Determinismoa.** Sarrera berak aztarna bera sortzen du beti.
2. **Lauxtasun-efektua (Avalanche effect).** Sarreran aldaketa txiki bat egiteak guztiz desberdina den aztarna sortzen du, aurrekoarekin inolako antzekotasun ikusgarritik gabe.
3. **Inbertsioarekiko erresistentzia.** Aztarna bat emanda, ez da bideragarria konputazionalki hura sortu zuen testua aurkitzea.
4. **Talkarekiko erresistentzia.** Ez da bideragarria konputazionalki aztarna bera sortzen duten bi testu desberdin aurkitzea.

«Ez da bideragarria konputazionalki» esateak ez du esan nahi «matematikoki ezinezkoa denik». Esan nahi du hori lortzeko denbora-, energia- eta diru-kostua eskuragarri dagoen konputazio-ahalmen guztiaren batura baino handiagoa dela magnitude-ordenetan. SHA-256-rako, muga hori milaka bilioi urtetan neurtzen da, hardware espezializatua duten planteamendu baikorrenetan ere. Horrek, irakurlearen helburu praktikoetarako, «ezin dela» esan nahi du.

SHA-256, zehazki

Izenak dena dio. SHA *Secure Hash Algorithm*-en siglak dira (Hash Algoritmo Segurua). 256 zenbakiak aztarnaren tamaina adierazten du bitetan: berrehun eta berrogeita hamasei bit, hau da, hogeita hamabi byte, hamaseitarrean (hexadecimal) irakurleak dagoeneko ezagutzen dituen hirurogeita lau karakterekoak direnak. NIST estatubatuarrak argitaratu zuen estandarra, mota honetako funtzioak normalizatzen dituen erakundeak, 2001ean, SHA-2 familiaren barruan; indarrean dagoen estandarraren bertsioa, FIPS 180-4, 2015ekoa da.

Bitak eta byteak zer diren oraindik ez daukatenentzat:

```
Bit 1    →  0 edo 1        (etengailu bat: piztuta edo itzalita)
Byte 1   →  8 bit          (256 konbinazio posible)
32 byte  →  256 bit        (SHA-256 aztarna)
```

Izenaren amaierako 256 zenbakiak aztarnaren tamaina bitetan dio. Hamaseitarrean —hamar ikurren ordez hamasei dituen zenbakitze-sistema bat—, 256 bit horiek zehazki 64 karakteretan sartzen dira. Horiek dira Cuaderno bakoitzaren oinean ikusten dituzun 64 karaktereak.

Dimentsioek une bat merezi dute. Berrehun eta berrogeita hamasei bitek bi berrehun eta berrogeita hamasei balio desberdin ahalbidetzen dituzte: hirurogeita hamazortzi digitu hamartar dituen zenbaki bat, unibertso behagarriko atomoen kopuru estimatua baino magnitude-ordena batzuk handiagoa. Munduko testu bakoitza —liburu bakoitza, posta elektroniko bakoitza, mezu bakoitza— balio horietako baten gainean caditzen da. Bi testu desberdin kasualitatez bat etortzeko probabilitatea, helburu praktikoetarako, zeroren parekoa da.

Kodean nola ikusten den

Zig hizkuntzan, Solo2 sostengatzen duten piezak idazteko erabiltzen duguna, testu baten SHA-256 zigilua kalkulatzea honela ikusten da:

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

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

Zig-en liburutegi estandarrari komatxo arteko testuaren SHA-256 kalkulatzeko eskatu diogu. Deia egin ondoren, *resumen* aldagaian zigilua osatzen duten hogeita hamabi byteak daude forma gordinean; pantailan hamaseitarrean erakusten direnean, artikulu honen oinean agertzen diren hirurogeita lau karaktereak dira. *Cuadernos Lacre*-ren ordez *Cuadernos lacre* jartzen badugu —letra larri bat gutxiago—, zigilua osorik aldatuko litzateke. Hori da, bost lerrotan, gainerakoa sostengatzen duen propietate nagusia. Barruan nola funtzionatzen duen ikusi nahi duenarentzat, artikuluaren amaieran algoritmoaren bertsio irakurgarria sartu dugu, urratsez urratseko iruzkinekin.

Zergatik deitzen diogun lakre-zigilua

XV. mendetik XIX. mendera bitarteko Europako korrespondentzian, lakreak ixten zuen gutuna. Argizari urtu tanta bat, zigilu bat gainean sakatuta, eta gutuna modu errepikaezinean markatuta geratzen zen. Ez zuen edukia babesten begiluze saiatuaren aurrean —papera argiaren kontra irakur zitekeen, lakrea hautsi zitekeen—, baina frogatu egiten zuen. Ixtearen edozein aldaketa ikusgarria zen hartzailearentzat papera ireki aurretik ere. Lakreak ez zuen kaltea eragozten; aitortu egiten zuen.

Cuaderno bakoitzaren gorputzaren SHA-256-ak funtzio bera betetzen du bere bertsio digitalean. Artikuluko hitz bakar bat aldatuko balitz argitaratu zenetik irakurtzen duzun unera arte, testuaren oineko zigilu hamaseitarra ez litzateke bat etorriko aurrean duzun testuaren SHA-256-arekin. Bost lerroko kodea duen edozein irakurlek egiaztatu ahal izango luke. Argitalpenak ezin du bere historia berridatzi zigiluak salatu gabe. Ez du kaltearen aurka babesten; egiaztagarri egiten du.

Hash bat ez dena

Batzuetan SHA-256-ri dagozkionak ez diren lau erabilera eskatzen zaizkio:

1. **Zifratzea.** Hash batek laburtu egiten du; ez du ezkutatzen. Testua irakurtzerik ez izatea nahi baduzu, zifratu egin behar duzu, ez hasheatu.
2. **Egilea autentifikatzea.** Hash batek ez du esaten nork idatzi duen testua, zer testu hasheatu den baizik. Egiletza lotzeko, sinadura kriptografiko bat behar da hasha-ren gainean, ez hasha hutsa.
3. **Pasahitzak gordetzea.** Hemen tranpa bat dago, ulertzea komeni dena. SHA-256 oso azkarra izateko diseinatuta dago —hori ona da gauza askotarako, baina txarra honetarako—. Hardware espezializatua duen erasozaile batek milaka milioi pasahitz proba ditzake segundoko SHA-256 hash baten aurka, zurea aurkitu arte. Pasahitzak gordetzeko, gakoak deribatzeko funtzio nahita motelak erabili behar dira, hala nola Argon2, scrypt edo bcrypt, *sal* batekin (erabiltzaile bakoitzeko ausazko datu bakar bat, pasahitz bera duten bi pertsonak hash bera izatea eragozten duena) konbinatuta.
4. **Hasha egilearen identifikatzaile gisa irakurtzea.** Ez da horrela. Hash batek edukia identifikatzen du. Bi pertsonak *hola* (kaixo) hitza SHA-256-rekin hasheatzen badute, biek laburpen bera lortzen dute —eta hori propietate nagusia da, ez akatsa: laburpen desberdinak balira, ezingo genuke egiaztatu argitaratutakoaren eta jasotakoaren arteko kointzidentzia.

Non agertzen den SHA-256 zure egunerokoan

Ikusten ez baduzu ere, SHA-256-ak interneten egunero erabiltzen duzunaren zati handi bat sostengatzen du. Bitcoin-en bloke-katea bloke bakoitzaren SHA-256-a hurrengoarekin kateatuz eraikitzen da; iraganeko bloke bat aldatzeak ondorengo kate osoa berriro kalkulatzea dakar. Gitek, mundu erdiko kodea bertsionatzeko erabiltzen den sistemak, konpromiso (commit) bakoitza bere eduki osoaren SHA-256-aren bidez (bertsio berrietan) edo aurreko SHA-1 baten bidez (bertsio zaharragoetan) identifikatzen du. Webgune batean sartzean haren identitatea egiaztatzen duten HTTPS ziurtagiriek SHA-256 aztarna bat dute lotuta. Software deskargak sarritan garatzaileak argitaratutako SHA-256 batekin batera doaz, fitxategia bidean aldatu ez dela egiazta dezazun. Eta, esan dugun bezala, Cuaderno Lacre bakoitzaren oinean.

Irakurle profesionalarentzat

Lau gogorarazpen operatibo sistemak erabakitzen edo ikuskatzen dituenarentzat:

1. Hash-a ez da zifratzea. Hornitzaile batek bi terminoak nahasten baditu bere dokumentazio teknikoan, zehazki zer esan nahi duen galdetzea komeni da.
2. Pasahitzak gordetzeko ez da inoiz SHA-256 hutsa erabili behar. SHA-256 oso azkarra da zeregin honetarako (ikus *Hash bat ez dena*-ko 3. puntua). Gaur egungo estandarra **Argon2id** da: motel diseinuz, zerbitzariaren ahalmenaren arabera konfiguragarria, eta erabiltzaile bakoitzeko *sal* ausazko desberdin batekin konbinatua.
3. Dokumentuen osotasunerako —kontratuak, espedienteak, fitxategiak—, SHA-256-ak erreferentziazko estandarra izaten jarraitzen du. EBko denbora-zigilatzaile kualifikatuek erabiltzen dutena da.
4. Epe luzerako kontserbaziorako (hamarkadak), SHA-3 edo SHA-512 bat ere kalkulatzea eta artxibatzea komeni da SHA-256arekin batera; zuhurtzia kriptografikoak funtzio bakarrean ez fidatzea gomendatzen du ehun urteko artxiboetan.

Teknikoki, sarrera-blokeen artean bitarteko egoera mantentzen duen egitura iteratu hau **Merkle-Damgård** eraikuntza gisa ezagutzen da. SHA-1, SHA-2 (SHA-256 barne) eta beste hash funtzio klasiko asko eredu horretan oinarritzen dira. SHA-3k, aldiz, Merkle-Damgård alde batera uzten du eta *esponja* izeneko arkitektura baten alde egiten du.

Nola funtzionatzen duen SHA-256k, pausoz pauso, hitz lauetan

Imajinatu munduko domino-zirkuiturik landuena muntatu duzula: milaka fitxa, dozenaka sardetze, zubi mekanikoak eta gela osoa zeharkatzen duten arrapalak, piezaz pieza kontu handiz jarrita.

Lehenengo fitxari kolpe bat ematen badiozu, katea sekuentzia zehatz eta errepikagarrian erortzen da. Muntaketa bera, hasierako kolpe bera → eroritako fitxen amaierako patroi bera, behin eta berriz.

Hona hemen interesgarria dena: mugitu **fitxa bakar bat** zentimetro erdi batera hasi baino lehen eta jo berriro. Aktibatu behar zen arrapala bat geldirik geratzen da, zubi bat ez da erortzen, beste sardetze bat aktibatzen da. Lurreko fitxen amaierako patroia guztiz ezagutezina da lehenengoarekin alderatuta.

SHA-256 matematikoki zirkuitu hau da. Idazten duzun testua fitxen hasierako posizioa da. Algoritmoa jausia askatzen duen kolpea da. Eta amaierako emaitza —*hash* deitzen duguna— dena gelditu denean lurrean geratzen den argazki finkoa da. Aldatu koma bakar bat jatorrizko testuan eta argazkia goitik behera aldatuko da. Hain sinplea, eta hain erabatekoa.

1. pausoa. Testua fitxa bitarretara itzuli. Ordenagailuek ez dituzte letrak ulertzen; lehenik zenbakietara (ASCII) itzultzen dituzte eta zenbakiak bitarrera (bata eta zeroak). Letra bakoitza 8 fitxa zuri edo beltz bihurtzen da: *A* letra `01000001` da, *B* letra `01000010` da, espazioa `00100000` da. Zure testu osoa —hitz bat, kontratu bat, nobela bat— fitxa zuri eta beltzezko lerro luze bat bihurtzen da.

2. pausoa. Tamaina estandarrera arte bete. Zirkuituak lerroa taldetan prozesatzen du, zehazki 512 fitxako **tarteetan**. Zure mezua 512ren multiploa ez bada, fitxa markatzaile bat (`10000000` balioa duena) gehitzen da testuaren ondoren eta gero zeroak tartea osatu arte. Tarte bakoitzeko azken 64 posizioak testuaren jatorrizko luzera idazteko gordetzen dira. Horrela, zirkuituak beti daki non amaitu zen benetako edukia eta non hasi zen betegarria.

3. pausoa. Zortzi fitxa maisuak jarri. Hasi baino lehen, mahai gainean **zortzi fitxa maisu** jartzen ditugu hasierako posizio zehatz batean. Zortzi fitxa hauek ez dira sekretua: haien hasierako balioa arau matematiko publiko batek finkatzen du (lehenengo zortzi zenbaki lehenen erro karratuak —2, 3, 5, 7, 11, 13, 17, 19— eta erro bakoitzaren parte hamartarraren lehen bitak). Munduko edozein txokotan, denak zortzi fitxa maisu berdinekin eta posizio berean hasten dira. Haien patua elur-jausiak bultzatzea eta eraldatzea da.

4. pausoa. Jausi handia: hirurogeita lau bultzada-txanda. Hemen hasten da ikuskizuna. Zure testuko 512 fitxako lehen tartea zortzi fitxa maisuen aurka talka egiten da. Baina ez dira kolpe batez erortzen: mekanismoak **hirurogeita lau txanda jarraian** exekutatzen ditu. Txanda bakoitzean hiru eragiketa egiten ditu fitxekin:

- **Zaldiko-maldikoa** (biraketa). Fitxak zirkuluan mugitzen dira: eskuinekoak ezkerrera pasatzen dira. Ez da fitxarik galtzen edo gehitzen; zaldiko-maldikoari bira osoa emanez berrordenatzen dira, besterik gabe. Informazioa birbanatzeko modu merkea eta itzulgarria da.
- **Onil Logikoa** (XOR). Fitxak onil batetik pasatzen dira eta binaka konparatzen dira: biak kolore berekoak badira, zuri bat ateratzen da; desberdinak badira, beltz bat. Logika bitarreko eragiketarik sinpleena da, baina zaldiko-maldikoaren biraketarekin konbinatuta oso indartsua bihurtzen da informazioa nahasteko, galdu gabe.
- **Gainezkatzea** (batuketa modularra). Emaitza hirurogeita lau konstanteren zerrenda publiko batetik hartutako *bultzada-fitxa konstante* batekin batzen da (lehen hirurogeita lau zenbaki lehenen erro kubikoak). Batuketak aurreikusitako 32 fitxako espazioan kabitzen ez diren fitxa gehigarriak sortzen baditu, soberako fitxa horiek baztertu egiten dira. Mahaiak 32 fitxarentzako lekua baino ez du, ez bat gehiago.

Hirurogeita laugarren txandaren amaieran, zure testuaren tarteko fitxa bakoitzak zortzi fitxa maisuen posizioan eragina izan du. Bultzadaren energiak zirkuitu osoa zeharkatu du.

5. pausoa. Hurrengo tartea gehitu (hasieratu gabe). Zure testua luzea bazen eta prozesatzeko 512 fitxako beste tarte bat badago, **zirkuitua ez da hasieratzen**. Zortzi fitxa maisuak lehen jausiak utzi zituen bezala geratzen dira, eta bigarren tartea haien aurka jaurtitzen da beste hirurogeita lau txanda aktibatzeko. Erori berri denaren amaieran dominoz betetako gela berri bat gehitzea bezala da: lehenengoaren desordenak guztiz baldintzatzen du bigarrena nola eroriko den.

6. pausoa. Amaierako argazkia egin. Prozesatzeko tarte gehiago geratzen ez denean, jausia gelditu egiten da. Zortzi fitxa maisuak geratu diren amaierako posizioari begiratzen diogu. Haien konfigurazioa sistema hamaseitarreko letra eta zenbakizko kode batera itzultzen dugu. Emaitza zehazki hirurogeita lau karaktere dituen katea da: hori da zure SHA-256 zigilua.

Zirkuitua nola dagoen muntatuta ikusita, lau propietate berez ateratzen dira:

1. **Determinismoa.** Testu berak beti argazki finko bera sortzen du, munduko edozein ordenagailutan. Zero ausazkotasun, zero sorpresa.
2. **Elur-jausi efektua.** Koma bat gehitzea, letra larri bat aldatzea, azentu bat ahaztea: argazkia guztiz ezagutezina suertatzen da. Hau da hasieran deskribatu dugun muturreko sentikortasuna.

3. **Norabide bakarra.** Amaierako argazkia izanda, ezin duzu jatorrizko testua berreraiki. Biraketek, onilek eta gainezkatzeek bit bakoitza *nondik zetorren* buruzko norabide-informazio guztia suntsitzen dute, eta *guztira zer gehitu zen* bakarrik gordetzen dute.
4. **Talkarekiko erresistentzia.** Kriptoanalisi publikoko hogeita bost urtetan, inork ez du lortu argazki finkoak berdinak dituzten bi testu desberdin aurkitzea. Eta hori egiteko zailtasuna zentzuz imajina daitekeen edozein zibilizazioren kalkulu-ahalmenetik kanpo dago.

Hurrengo kode-eranskineak zehazki sei pauso hauek inplementatzen ditu Zig-en. Orain bit-eragiketa bakoitzak zer esan nahi duen jakinda irakur dezakezu, manipulazioak itsu-itsu onartu beharrean.

Glosario teknikoa

Eragiketa bakoitzak zer egiten duen ulertu nahi duen irakurlearentzat. Saltatu lasai: artikulua hori gabe ere ulertzen da.

ASCII eta Unicode — letrak nola bihurtzen diren zenbaki. Ordenagailuek ez dituzte letrak ikusten; zenbakiak ikusten dituzte. **ASCII** izeneko estandar batek (*American Standard Code for Information Interchange*, 1963koa) zenbaki zehatz bat esleitzen dio teklatuko karaktere bakoitzari: *A* 65 da, *B* 66 da, *a* 97 da, *0* 48 da, espazioa 32 da, koma 44 da. Sistema modernoek **Unicode**rekin hedatzen dute hori, munduko alfabeto guztietako karaktere bakoitzari zenbaki bat esleitzen diona: zirilikoa, arabiera, txinera, japoniera eta baita emojiak ere. Karaktere bat idazten duzunean edo testu-fitxategi bat irekitzen duzunean, ordenagailuak azpiko zenbakia irakurtzen du, ez pantailako forma. SHA-256k zenbaki hauen gainean egiten du lan, edozein testu zifra-sekuentzia luze gisa tratatuz. Horregatik zigilatu ditzake gaztelaniazko artikulu bat, japonierazko poema bat eta fitxategi bitar bat algoritmo berarekin.
XOR — bit bakoitzeko konparatzailea. XOR (ingelesezko *exclusive or*, «o esklusiboa») ordenagailu batek bi zenbaki bitarrekin egin dezakeen eragiketarik sinpleenetako bat da. Bi bitak posizioz posizio konparatzen ditu eta hau itzultzen du: **1** bietatik zehazki bat 1 bada (bat baina ez biak), **0** biak berdinak badira (biak 0 edo biak 1). Adibidea: `1010` eta `1100`-ren XOR `0110` da. Propietate nabarmen bat du: itzulgarria da —gako berarekin XOR bi aldiz egiten baduzu, jatorrizkora itzultzen zara—. Horregatik da kriptografiaren zaldia: bitak nahasten ditu informazioa galdu gabe, mas el resultado no revela nada sobre las entradas si no conoces una de ellas.
Hamaseitarra — 16 oinarrian kontatzea. Eguneroko zenbaki ia guztiek hamar digitu erabiltzen dituzte (0-9). Hamaseitarrak hamasei erabiltzen ditu: ohiko 0-9 gehi sei letra ondorengo balioak ordezkatzeko: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Zergatik hamasei? Ordenagailuek lau bitako taldetan pentsatzen dutelako, eta lau bitek zehazki hamasei balio desberdin ordezkatu ditzakete —horrela, karaktere hamaseitar bat lau biti dagokio garbi—. SHA-256 hatz-marka batek 256 bit neurtzen ditu, hau da, zehazki **64 karaktere hamaseitar**. Ohiko hamartarrean idatziko bagenu, 78 digitu inguru beharko lituzke eta baldarra izango litzateke. Aukera estetikoa eta konpaktua da; azpiko zenbakia bera da.
Bit-biraketa — biraketa-maldiko bitarra. Imajinatu zazpi bonbillako lerro bat, batzuk piztuta (1) eta beste batzuk itzalita (0): `1 0 1 1 0 0 1`. Eskuinera posizio bat biratzea eskuineko bonbilla hartu, ezkerreko muturrera eraman eta besteak leku bat eskuinera desplazatzean datza: `1 1 0 1 1 0 0`. Ez da bonbillarik galtzen edo gehitzen: zirkuluan dantzan egiten dute, besterik gabe. SHA-256k bit-biraketa ehunka aldiz erabiltzen du kalkulu bakoitzean; egoeraren barruan informazioa birbanatzeko modu merkea eta galerarik gabea da.
Konstante *«nothing-up-my-sleeve»* — zergatik datozen zenbaki lehenetatik. SHA-256ren zortzi fitxa maisuak eta hirurogeita lau txanda-konstanteak ez ziren ausaz aukeratu. Lehenengo zenbaki lehenen erro karratu eta kubikoetatik datoz. Zergatik? Haien diseinatzaileek konstante *«sin nada bajo la manga»* nahi zituztelako: jatorria edonork egiaztatu dezakeen balioak. Norbaitek *«fida zaitez nitaz: erabili 32 biteko ausazko zenbaki hau»* esango balizu, ezkutuko ahultasun edo atzeko ate baten susmoa izango zenuke arrazoiz. Baina kalkulagailua duen edonork egiaztatu dezake 2ren erro karratuaren lehen 32 bitak `0x6a09e667` direla. Balioak matematikoak, publikoak eta erreproduzigarriak dira: tranpa ezkuturik ezin da errezetan sartu.

Apendizea: SHA-256 kode irakurgarrian

Apendize hau algoritmoa barrutik ikusi nahi duen irakurlearentzat da. FIPS 180-4 espezifikazioari jarraitzen dion Zig-eko inplementazio didaktiko bat da. Ez da Solo2-k erabiltzen duen bertsioa —benetakoa Zig-en liburutegi estandarreko `std.crypto.hash.sha2.Sha256`-n dago, optimizatuta eta auditoreek aztertuta—. Baina algoritmoa bera da: hemen ikusten duzuna da, urratsez urratseko lana, bost karaktereko dei horrek bere lana egiten duenean gertatzen dena.

```
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
}
```

Egitura bera jarraitzen duen beste edozein hizkuntzatan egindako beste edozein idazketak —hasierako konstanteak, schedule-aren hedapena, hirurogeita lau erronda, metaketa— emaitza bera ematen du. Algoritmoak ez du sekreturik: haren balioa lehen aipatutako propietateek hor jarraitzen dutela da, milaka begik bi hamarkadako kriptoanalisi publikoa egin ondoren.

Artikulu honen oinera itzultzen bazara, hirurogeita lau karaktereko zigilu hamaseitar bat ikusiko duzu. Irakurri berri duzun testuaren SHA-256-a da, hizkuntza honetan. Artikulua itzuliko bagenu, zigilua beste bat litzateke; euskarazko bertsioaren hitz bat aldatuko balitz, euskal zigilua aldatu egingo litzateke. Zigiluak ez du edukia babesten —horretarako beste tresna batzuk daude—, baina modu unibokoan identifikatzen du. Eta hori, apala badirudi ere, nahikoa da kate editorialeko urrats batek ere esandakoa aldatu ezin dezan ohartu gabe. Gainerakoa —zifratzea, sinatzea, identifikatzea— ideia sinple honen gainean eraikitzen da.

Iturriak eta irakurgai gehiago

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, 2015eko abuztua. SHA-2 familiaren espezifikazio ofiziala, SHA-256 barne.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, 2011ko maiatza. Inplementatzaileentzako bertsio arauemailea.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). 5. eta 6. kapituluek hash funtzioak eta haien erabilera legitimo eta ez-legitimoak jorratzen dituzte.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Eraikuntzaz aldaezina den egitura batean blokeak kateatzeko SHA-256 erabiltzearen adibide praktikoa.
- 910/2014 (EB) Erregelamendua (eIDAS) — denbora-zigilatzaile kualifikatuek esparrua. SHA-256 erreferentziazko funtzioa da EBn ematen diren sinadura eta zigilu elektroniko kualifikatuetarako.
- Erreferentziazko inplementazioa Zig-en: `std.crypto.hash.sha2.Sha256` hizkuntzaren biltegi ofizialean (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Solo2-k erabiltzen duen bertsio optimizatua eta auditorea da. Baliagarria apendizeko inplementazio didaktikoarekin alderatzeko.

← AurrekoaCUADERNOS_LIST_SCHREMS_TITLEHurrengoa →CUADERNOS_LIST_KILLSWITCH_TITLE

Azken irakurketak

- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE

Eraman artikulu hau zurekin behar duzun lekura.

↓ Markdown ↓ Testu arrunta ↓ PDF

Fitxategia zure gailura deskargatuko da. Bertatik gorde, Solo2ra inportatu edo nahi duzun lekuan parteka dezakezu. Cuadernosek ez du helburua zure ordez erabakitzen.

Lakre-zigilua · SHA-256 77372ceef6f6bacce1bfde373d2638393e2b80e6ebacbd4ac156823e436b02b6

Cuadernos Lacre · Menzuri Gestión S.L.ren argitalpena ·
R.Eugeniok idatzia · Solo2 taldeak editatua.