Kas iš tikrųjų yra SHA-256

Matematinis antspaudas, telpantis į šešiasdešimt keturis simbolius, kuris visiškai pasikeičia, jei pajudinamas bent vienas originalaus teksto kablelis. Kodėl jį vadiname skaitmeniniu vaško antspaudu.

Kad suprastume: Įsivaizduokite mašiną, kuri nuskaito bet kokį tekstą ir pateikia 64 simbolių seką. Jei tekstas įvedamas identiškas, seka išeina identiška. Jei pajudinsite bent vieną kablelį, seka bus visiškai kitokia. Ta seka yra skaitmeninis antspaudinis vaškas.

Paprasta idėja už techninio pavadinimo

Įsivaizduokite, kad egzistuoja mašina su viena anga ir vienu ekranu. Per angą įdedate tekstą: žodį, frazę, visą romaną. Ekrane po akimirkos pasirodo seka, sudaryta lygiai iš šešiasdešimt keturių simbolių. Profesionalus skaitytojas šią seką vadina *hash* (maiša) arba kriptografine santrauka; paprastam skaitytojui kol kas galime ją vadinti matematiniu teksto antspaudu, panašiai kaip piršto atspaudas yra žmogaus tapatybės ženklas.

Jei tą patį tekstą įdėsite du kartus, mašina abu kartus parodys tą patį antspaudą. Jei įdėsite šiek tiek kitokį tekstą – paslinksite vieną kablelį, pakeisite didžiąją raidę mažąja – mašina parodys visiškai kitokį antspaudą nei pirmąjį. Ne panašų, o visiškai kitokį. Šios dvi savybės kartu – determinizmas ir jautrumas – yra ta paprasta idėja. Visa kita SHA-256 sistemoje yra tik mechanizmas, užtikrinantis šių savybių veikimą.

Svarbu iš pat pradžių pasakyti, ko mašina nedaro. Ji nešifruoja teksto. Ji jo neslepia. Ji jo nesaugo. Mašina pažiūri į tekstą, apskaičiuoja antspaudą ir tekstą pamiršta. Pagal antspaudą neįmanoma atkurti jį sugeneravusio teksto; galima tik turint kandidatinį tekstą patikrinti, ar jis sutampa su originalu. Todėl sakome, kad tai yra *vienpusė* santrauka: galima nueiti, bet negrįžti.

Maiša nėra tas pats, kas šifravimas

Painiava kyla dažnai, todėl verta ją išsklaidyti: šifravimas ir „hašavimas“ (maiša) yra skirtingos operacijos. Šifravimas – tai teksto transformavimas taip, kad tik rakto turėtojas galėtų jį grąžinti į pradinę formą. Maiša – tai teksto antspaudo sukūrimas, iš kurio pradinio teksto niekada neįmanoma atkurti, nei su raktu, nei be jo. Pirmoji operacija pagal konstrukciją yra grįžtamoji, antroji – negrįžtamoji.

Praktinė pasekmė yra svarbi. Kai programėlė sako „saugome jūsų slaptažodį užšifruotą“, yra kažkas, kas turi raktą jam iššifruoti – bent jau pati programėlė. Kai programėlė sako „saugome jūsų slaptažodį suhašuotą“, pati programėlė negali perskaityti pradinio slaptažodžio, net jei norėtų; ji gali tik patikrinti, ar tai, ką įrašote, vėl sugeneruoja tą patį antspaudą. Antrasis modelis, teisingai įgyvendintas, yra daug geresnis slaptažodžiams saugoti. Vėliau pamatysime, kodėl „teisingai įgyvendintas“ reikalauja šiek tiek daugiau nei vien tik SHA-256.

Keturios savybės, dėl kurių kriptografinė santrauka yra naudinga

Maišos funkcija, verta apibūdinimo *kriptografinė*, atitinka keturias savybes:

1. **Determinizmas.** Ta pati įvestis visada sugeneruoja tą patį antspaudą.
2. **Lavinos efektas.** Nedidelis įvesties pakeitimas sugeneruoja visiškai kitokį antspaudą, neturintį jokio matomo panašumo į ankstesnį.
3. **Atsparumas atvirkštinei inžinerijai.** Turint antspaudą, skaičiavimo požiūriu neįmanoma rasti jį sugeneravusio teksto.

4. **Atsparumas kolizijoms.** Skaičiavimo požiūriu neįmanoma rasti dviejų skirtingų tekstų, kurie sugeneruotų tą patį antspaudą.

„Skaičiavimo požiūriu neįmanoma“ nereiškia „matematiškai neįmanoma“. Tai reiškia, kad laiko, energijos ir pinigų sąnaudos tam pasiekti daug kartų viršija visus pagrįstai prieinamus skaičiavimo pajėgumus. SHA-256 atveju ši riba matuojama tūkstančiais trilijonų metų net ir naudojant optimistiškiausią specializuotą įrangą. Tai skaitytojui praktiškai reiškia tą patį, ką ir „neįmanoma“.

Konkrečiai apie SHA-256

Pavadinimas pasako viską. SHA yra santrumpa iš *Secure Hash Algorithm* – saugus maišos algoritmas. Skaičius 256 nurodo antspaudo dydį bitais: du šimtai penkiasdešimt šeši bitai, t. y. trisdešimt du baitai, kurie šešioliktainėje sistemoje rodomi kaip tie šešiasdešimt keturi simboliai, kuriuos skaitytojas jau atpažįsta. Standartą 2001 m. paskelbė JAV NIST, organizacija, normuojanti tokio tipo funkcijas, kaip SHA-2 šeimos dalį; dabartinė standarto versija FIPS 180-4 yra iš 2015 m.

Tiems, kurie dar ne visai supranta, kas yra bitai ir baitai:

```
1 bitas   →  0 arba 1       (jungiklis: įjungta arba išjungta)
1 baitas  →  8 bitai        (256 galimos kombinacijos)
32 baitai →  256 bitai      (SHA-256 antspaudas)
```

Skaičius 256 pavadinimo gale nurodo antspaudo dydį bitais. Šešioliktainėje sistemoje – skaičiavimo sistemoje su šešiolika simbolių vietoj dešimties – šie 256 bitai telpa lygiai į 64 simbolius. Tai yra tie 64 simboliai, kuriuos matote kiekvieno „Cuaderno“ apačioje.

Verta akimirkai susimąstyti apie mastelius. Du šimtai penkiasdešimt šeši bitai leidžia du pakeltą du šimtai penkiasdešimt šeštuoju laipsniu skirtingų reikšmių: tai skaičius su septyniasdešimt aštuoniais skaitmenimis, keliais eilės laipsniais didesnis nei numatomas atomų skaičius matomoje visatoje. Kiekvienas pasaulio tekstas – kiekviena knyga, kiekvienas el. laiškas, kiekviena žinutė – patenka į vieną iš tų reikšmių. Tikimybė, kad du skirtingi tekstai sutaps atsitiktinai, praktiškai nesiskiria nuo nulio.

Kaip tai atrodo kode

„Zig“ kalba, kuria rašome „Solo2“ pagrindus, SHA-256 antspaudo skaičiavimas tekstui atrodo taip:

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

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

Ką tik paprašėme „Zig“ standartinės bibliotekos apskaičiuoti teksto kabutėse SHA-256. Po iškvietimo kintamasis *resumen* saugo trisdešimt du baitus, sudarančius antspaudą žalia forma; ekrane šešioliktainėje sistemoje jie rodomi kaip šešiasdešimt keturi simboliai šio straipsnio apačioje. Jei pakeistume *Cuadernos Lacre* į *Cuadernos lacre* – viena didžioji raidė taptų mažąja – antspaudas visiškai pasikeistų. Tai yra ta pagrindinė savybė, kuri per penkias eilutes išlaiko visą likusią sistemą. Norintiems pamatyti, kaip tai veikia viduje, straipsnio pabaigoje pateikiame skaitomą algoritmo versiją su komentarais žingsnis po žingsnio.

Kodėl vadiname vaško antspaudu

Europos korespondencijoje nuo penkiolikto iki devyniolikto amžiaus vaškas užsandarindavo laišką. Ištirpusio vaško lašas, ant viršaus prispaustas antspaudas, ir laiškas likdavo pažymėtas nepakartojamai. Tai neapsaugodavo turinio nuo ryžtingo smalsuolio – popierių buvo galima perskaityti prieš šviesą, vašką galima buvo sulaužyti – bet tai tapdavo akivaizdu. Bet koks uždarymo pažeidimas buvo matomas gavėjui dar prieš atidarant popierių. Vaškas neužkirsdavo kelio žalai; jis ją deklaruodavo.

Kiekvieno „Cuaderno“ teksto SHA-256 atlieka tą pačią funkciją skaitmeninėje versijoje. Jei straipsnyje pasikeistų bent vienas žodis nuo jo paskelbimo momento iki to laiko, kai jį skaitote, šešioliktainis antspaudas teksto apačioje nebesutaptų

su teksto, kurį matote, SHA-256 reikšme. Bet kuris skaitytojas, parašęs penkias kodo eilutes, galėtų tai patikrinti. Leidėjas negali perrašyti savo istorijos, kad antspaudas jo neišduotų. Tai neapsaugo nuo žalos; tai padaro ją patikrinamą.

Kuo santrauka (hash) nėra

Kartais iš SHA-256 tikimasi keturių naudojimo būdų, kurie jam nepriklauso:

1. **Šifravimas.** Maiša apibendrina; ji neslepia. Jei norite, kad teksto nebūtų galima perskaityti, turite jį šifruoti, o ne generuoti jo santrauką.
2. **Autoriaus autentiškumo patvirtinimas.** Santrauka nepasako, kas parašė tekstą, tik kokiam tekstui ji buvo sugeneruota. Norint susieti autorystę, reikalingas kriptografinis parašas ant santraukos, o ne pati santrauka savaime.
3. **Slaptažodžių saugojimas.** Čia yra spąstai, kuriuos verta suprasti. SHA-256 sukurtas būti labai greitas – tai gerai daugeliui dalykų, bet blogai šiam. Užpuolikas su specializuota įranga gali išbandyti milijardus slaptažodžių per sekundę lygindamas su SHA-256 santrauka, kol ras jūsiškį. Slaptažodžiams saugoti reikia naudoti sąmoningai lėtas rakto išvedimo funkcijas, tokias kaip „Argon2“, „scrypt“ arba „bcrypt“, derinant su *druska* (salt – unikalūs atsitiktiniai duomenys kiekvienam vartotojui, neleidžiantys dviem žmonėms su tuo pačiu slaptažodžiu turėti tą pačią santrauką).
4. **Santraukos naudojimas kaip autoriaus identifikatoriaus.** Tai nėra identifikatorius. Santrauka identifikuoja turinį. Jei du žmonės sugeneruos žodžio *hola* santrauką per SHA-256, abu gaus tą patį rezultatą – ir tai yra pagrindinė savybė, o ne trūkumas: jei santraukos būtų skirtingos, negalėtume patikrinti, ar tai, kas paskelbta, sutampa su tuo, kas gauta.

Kur SHA-256 pasirodo jūsų kasdienybėje

Nors to ir nematote, SHA-256 palaiko didelę dalį to, ką kasdien naudojate internete. „Bitcoin“ blokų grandinė kuriama jungiant kiekvieno bloko SHA-256 su kitu; pakeitus praėjusį bloką, tenka perskaičiuoti visą vėlesnę grandinę. „Git“, sistema, kuria versijuojamas viso pasaulio kodas, kiekvieną pakeitimą (commit) identifikuoja pagal jo viso turinio SHA-256 (naujesnėse versijose) arba pagal jo pirmtaką SHA-1 (senesnėse versijose). HTTPS sertifikatai, patvirtinantys svetainės tapatybę, turi susietą SHA-256 antspaudą. Programinės įrangos atsisiuntimus dažnai lydi kūrėjo paskelbtas SHA-256, kad galėtumėte patikrinti, ar failas nebuvo pakeistas kelyje. Ir, kaip minėjome, kiekvieno „Cuaderno Lacre“ apačioje.

Profesionaliam skaitytojui

Keturi operaciniai priminimai tam, kas nusprendžia arba audituoja sistemas:

1. Maiša nėra šifravimas. Jei tiekėjas techninėje dokumentacijoje painioja šiuos du terminus, verta paklausti, ką tiksliai jis turi omenyje.
2. Slaptažodžiams saugoti niekada negalima naudoti tik SHA-256. SHA-256 yra per greitas šiai užduočiai (žr. 3 punktą skyriuje *Kuo santrauka nėra*). Dabartinis standartas yra **Argon2id**: lėtas pagal konstrukciją, konfigūruojamas pagal serverio pajėgumus, derinamas su skirtinga atsitiktine *druska* kiekvienam vartotojui.
3. Dokumentų – sutarčių, bylų, failų – vientisumui SHA-256 išlieka pagrindiniu standartu. Būtent jį naudoja kvalifikuoti laiko žymų teikėjai ES.
4. Ilgalaikiam saugojimui (dešimtmečiams) verta apskaičiuoti ir archyvuoti taip pat SHA-3 arba SHA-512 kartu su SHA-256; kriptografinis atsargumas rekomenduoja nesiremti tik viena funkcija saugant šimtmečio archyvus.

Techniškai ši iteracinė struktūra, kurioje tarpinė būsena išsaugoma tarp įvesties blokų, yra žinoma kaip **Merkle-Damgård** konstrukcija. Tai modelis, kuriuo remiasi SHA-1, SHA-2 (įskaitant SHA-256) ir daugelis kitų klasikinių maišos funkcijų. Priešingai, SHA-3 atsisako Merkle-Damgård ir naudoja kitokią architektūrą, vadinamą *kempine* (angl. sponge).

Kaip veikia SHA-256: žingsnis po žingsnio paprastais žodžiais

Įsivaizduokite, kad sukonstravote sudėtingiausią pasaulyje domino grandinę: tūkstančiai kauliukų, dešimtys atsišakojimų, mechaniniai tiltai ir rampos per visą kambarį, kruopščiai sudėlioti po vieną detalę.

Jei paliečiate pirmąjį kauliuką, grandinė griūva tikslia ir atkartojama seka. Ta pati sudėtis, tas pats pradinis prisilietimas → identiškas galutinis nukritusių kauliukų raštas, vėl ir vėl.

Štai čia ir prasideda įdomumai: pajudinkite **tik vieną kauliuką** pusę centimetro į šalį prieš pradedant ir vėl palieskite. Rampa, kuri turėjo suveikti, lieka nejudri, tiltas negriūva, suveikia kita atšaka. Galutinis kauliukų raštas ant grindų yra visiškai neatpažįstamas, lyginant su pirmuoju.

Matematiškai SHA-256 yra būtent ši grandinė. Jūsų rašomas tekstas yra pradinė kauliukų padėtis. Algoritmas yra prisilietimas, kuris paleidžia kaskadą. O galutinis rezultatas – tai, ką vadiname *maiša* (angl. hash) – yra grindų nuotrauka, kai viskas sustoja. Pakeiskite bent vieną kablelį originaliame tekste ir nuotrauka bus radikaliai kitokia. Taip paprasta ir taip drastiška.

1 žingsnis. Teksto vertimas į dvejetainius kauliukus. Kompiuteriai nesupranta raidžių; jie pirmiausia jas paverčia skaičiais (ASCII), o skaičius – dvejetaine sistema (vienetais ir nuliais). Kiekviena raidė virsta 8 baltais arba juodais kauliukais: *A* yra 65 (ASCII), *B* yra 66. Kompiuterio atmintyje *A* tampa `01000001`, *B* – `01000010`, tarpas – `00100000`. Visas jūsų tekstas – žodis, sutartis, romanas – tampa ilga baltų ir juodų kauliukų eile.

2 žingsnis. Užpildymas iki standartinio dydžio. Grandinė apdoroja eilę tiksliai po 512 kauliukų *blokus*. Jei jūsų pranešimas nesiekia 512 kartotinio, pridedamas žymeklio kauliukas (kurio vertė `10000000`) iškart po teksto, o tada nuliai, kol užpildomas blokas. Paskutinės 64 kiekvieno bloko pozicijos rezervuotos pradinio teksto ilgiui pažymėti. Taip grandinė visada žino, kur baigėsi tikrasis turinys ir kur prasidėjo užpildas.

3 žingsnis. Aštuonių meistriškų kauliukų pastatymas. Prieš pradedant, ant stalo tiksliose pradinėse pozicijose pastatome **aštuonis meistriškus kauliukus**. Šie aštuoni kauliukai nėra paslaptis: jų pradinė vertė nustatyta pagal viešą matematinę taisyklę (pirmųjų aštuonių pirminių skaičių – 2, 3, 5, 7, 11, 13, 17, 19 – kvadratines šaknis ir pirmosios kiekvienos šaknies trupmeninės dalies bitus). Visi visame pasaulyje pradeda su tais pačiais aštuoniais meistriškais kauliukais tose pačiose pozicijose. Jų likimas – būti stumiamiems ir transformuojamiems griūties.

4 žingsnis. Didžioji griūtis: šešiasdešimt keturi stūmimų ratai. Čia prasideda reginys. Pirmasis 512 jūsų teksto kauliukų blokas susiduria su aštuoniais meistriškais kauliukais. Tačiau jie negriūva iškart: mechanizmas atlieka **šešiasdešimt keturis nuoseklius ratus**. Kiekviename rate su kauliukais atliekamos trys operacijos:

- **Karuselė** (rotacija). Kauliukai juda ratu: dešinieji pereina į kairę. Nė vienas kauliukas neprarandamas ir nepridedamas; jie tiesiog perrikiuojami apsukant pilną ratą karuselėje. Tai nebrangus ir grįžtamas būdas perskirstyti informaciją.
- **Loginis piltuvas** (XOR). Kauliukai keliauja per piltuvą, kuris juos lygina po du: jei abu yra tos pačios spalvos, išeina baltas; jei skirtingų – juodas. Tai paprasčiausia dvejetainės logikos operacija, tačiau derinant su karuselės rotacijomis ji tampa itin galinga maišant informaciją jos neprarandant.
- **Perpilda** (modulinis sudėjimas). Rezultatas sudedamas su *pastovaus stūmimo kauliuku*, paimtu iš viešo šešiasdešimt keturių konstantų sąrašo (pirmųjų šešiasdešimt keturių pirminių skaičių kubinės šaknys). Jei sudėtis sugeneruoja papildomų kauliukų, kurie netelpa numatytoje 32 kauliukų erdvėje, tie atliekami kauliukai atmetami. Ant stalo yra vietos tik 32 kauliukams, nė vienam daugiau.

Šešiasdešimt ketvirto rato pabaigoje kiekvienas jūsų teksto bloko kauliukas turėjo įtakos aštuonių meistriškų kauliukų padėčiai. Stūmimo energija apkeliavo visą grandinę.

5 žingsnis. Kito bloko pridėjimas (nepradedant iš naujo). Jei jūsų tekstas buvo ilgas ir liko dar vienas 512 kauliukų blokas, **grandinė neprasideda iš naujo**. Aštuoni meistriški kauliukai lieka tokie, kokius juos paliko pirmoji griūtis, ir antrasis blokas paleidžiamas į juos, suaktyvinant dar šešiasdešimt keturis ratus. Tai tarsi naujo kambario, pilno domino, pridėjimas prie ką tik sugriuvusio: pirmojo kambario netvarka visiškai lemia, kaip grius antrasis.

6 žingsnis. Galutinės nuotraukos padarymas. Kai nebelieka blokų apdorojimui, griūtis sustoja. Pažiūrime į galutinę padėtį, kurioje liko aštuoni meistriški kauliukai. Paverčiame jų konfigūraciją į raidžių ir skaičių kodą šešioliktainėje sistemoje. Rezultatas yra tiksliai šešiasdešimt keturių simbolių seka: tai jūsų SHA-256 antspaudas.

Keturios savybės savaime išplaukia iš to, kaip sukonstruota grandinė:

1. **Determinizmas.** Tas pats tekstas visada sukuria tą pačią galutinę nuotrauką bet kuriame pasaulio kompiuteryje. Nulis atsitiktinumo, nulis staigmenų.
2. **Griūties efektas.** Pridėtas kablelis, pakeista didžioji raidė, pamiršta varnelė – nuotrauka tampa visiškai neatpažįstama. Tai tas ypatingas jautrumas, kurį aprašėme pradžioje.
3. **Viena kryptis.** Turėdami galutinę nuotrauką, negalite atkurti originalaus teksto. Rotacijos, piltuvai ir perpildos sunaikina visą krypties informaciją apie tai, *iš kur atėjo kiekvienas bitas*, ir išsaugo tik tai, *kas buvo sudėta bendrai*.

4. **Atsparumas kolizijoms.** Per dvidešimt penkerius viešos kriptoanalizės metus niekam nepavyko rasti dviejų skirtingų tekstų, kurių galutinės nuotraukos sutaptų. O sudėtingumas tai padaryti viršija bet kurios pagrįstai įsivaizduojamos civilizacijos skaičiavimo galimybes.

Toliau pateiktame kodo priede šie šeši žingsniai tiksliai įgyvendinti Zig kalba. Dabar galite jį skaityti žinodami, ką reiškia kiekviena bitų operacija, užuot aklai priėmę manipuliacijas.

Techninis žodynėlis

Skaitytojui, kuris nori suprasti, ką daro kiekviena operacija. Galite drąsiai praleisti: straipsnis suprantamas ir be jo.

ASCII ir Unicode – kaip raidės virsta skaičiais. Kompiuteriai nemato raidžių; jie mato skaičius. Standartas, vadinamas **ASCII** (angl. American Standard Code for Information Interchange, 1963 m.), kiekvienam klaviatūros simboliui priskiria konkretų skaičių: *A* yra 65, *B* yra 66, *a* yra 97, *0* yra 48, tarpas yra 32, kablelis yra 44. Šiuolaikinės sistemos jį išplečia su **Unicode**, kuris priskiria skaičių kiekvienai kiekvienos pasaulio abėcėlės raidei: kirilicai, arabų, kinų, japonų ir net emocijų ženklams. Kai rašote simbolį ar atidarote tekstinį failą, kompiuteris skaito paslėptą skaičių, o ne formą ekrane. SHA-256 veikia su šiais skaičiais, apdorodamas bet kokį tekstą kaip ilgą skaitmenų seką. Todėl jis gali užantspauduoti straipsnį ispanų kalba, eilėraštį japonų kalba ir dvejetainį failą tuo pačiu algoritmu.
XOR – bitų palyginimo įrankis. XOR (tariama „eksor“, iš angl. exclusive or – „išskirtinis arba“) yra viena paprasčiausių operacijų, kurias kompiuteris gali atlikti su dviem dvejetainiais skaičiais. Jis lygina du bitus pozicija po pozicijos ir grąžina: **1**, jei tiksliai vienas iš dviejų yra 1 (vienas, bet ne abu), **0**, jei abu yra vienodi (abu 0 arba abu 1). Pavyzdys: `1010` ir `1100` XOR yra `0110`. Jis turi ypatingą savybę: yra grįžtamas – jei atliekate XOR du kartus su tuo pačiu raktu, grįžtate prie originalo. Todėl tai yra kriptografijos darbinis arklys: sumaišo bitus neprarandant informaciją, tačiau rezultatas nieko neišduoda apie įvestis, jei nežinote vienos iš jų.
Šešioliktainė sistema – skaičiavimas 16-os pagrindu. Beveik visi kasdieniai skaičiai naudoja dešimt skaitmenų (0-9). Šešioliktainė sistema naudoja šešiolika: įprastus 0-9 ir šešias raides, reprezentuojančias šias vertes: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Kodėl šešiolika? Nes kompiuteriai galvoja keturių bitų grupėmis, o keturi bitai gali reprezentuoti tiksliai šešiolika skirtingų verčių – taip šešioliktainis simbolis švariai atitinka keturis bitus. SHA-256 piršto atspaudas yra 256 bitų, o tai yra tiksliai **64 šešioliktainiai simboliai**. Jei jį rašytume įprasta dešimtaine sistema, jis užimtų apie 78 skaitmenis ir būtų nepatogesnis. Šis pasirinkimas yra estetinis ir kompaktiškas; paslėptas skaičius yra tas pats.
Bitų rotacija – dvejetainė karuselė. Įsivaizduokite septynių lempučių eilę, kai kurios dega (1), o kai kurios ne (0): `1 0 1 1 0 0 1`. Rotacija į dešinę per vieną poziciją reiškia paimti lemputę iš pačio dešinio krašto, nunešti ją į kairį galą ir pastumti likusias per vieną vietą į dešinę: `1 1 0 1 1 0 0`. Jokia lemputė neprarandama ir nepridedama: jos tiesiog šoka ratu. SHA-256 naudoja bitų rotaciją šimtus kartų kiekviename skaičiavime; tai pigus ir be nuostolių būdas perskirstyti informaciją būsenos viduje.
Konstantos „be apgaulės“ (angl. nothing-up-my-sleeve) – kodėl jos kyla iš pirminių skaičių. Aštuoni meistriški kauliukai ir šešiasdešimt keturios SHA-256 rato konstantos nebuvo pasirinktos atsitiktinai. Jos kyla iš pirmųjų pirminių skaičių kvadratinių ir kubinių šaknų. Kodėl? Nes jų kūrėjai norėjo konstantų *„be nieko paslėpto rankovėje“*: verčių, kurių kilmę kiekvienas galėtų patikrinti. Jei kas nors jums pasakytų *„pasitikėk manimi: naudok šį 32 bitų atsitiktinį skaičių“*, pagrįstai įtartumėte paslėptą silpnybę ar galines duris. Tačiau kiekvienas, turintis skaičiuotuvą, gali patikrinti, ar pirmieji 32 kvadratinės šaknies iš 2 bitai yra `0x6a09e667`. Vertės yra matematinės, viešos ir atkartojamos: jokie paslėpti spąstai negali patekti į receptą.

Priedas: SHA-256 suprantamu kodu

Šis priedas skirtas skaitytojui, kuris nori pamatyti algoritmą iš vidaus. Tai mokomasis įgyvendinimas „Zig“ kalba, atitinkantis FIPS 180-4 specifikaciją. Tai nėra ta versija, kurią naudoja „Solo2“ – tikroji yra `std.crypto.hash.sha2.Sha256` standartinėje „Zig“ bibliotekoje, optimizuota ir audituota. Tačiau algoritmas yra tas pats: tai, ką čia matote, yra žingsnis po žingsnio tai, kas vyksta, kai tas penkių simbolių iškvietimas atlieka savo darbą.

```
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
}
```

Bet koks perrašymas kita kalba, laikantis tos pačios struktūros – pradinių konstantų, tvarkaraščio išplėtimo, šešiasdešimt keturių raundų, akumuliavimo – duoda tą patį rezultatą. Algoritmas neturi paslapčių: jo vertė yra ta, kad išvardytos savybės išlieka galioti po dviejų dešimtmečių viešos kriptoanalizės tūkstančiais akių.

Jei grįšite į šio straipsnio apačią, pamatysite šešiasdešimt keturių simbolių šešioliktainį antspaudą. Tai yra teksto, kurį ką tik perskaitėte šia kalba, SHA-256. Jei išverstume straipsnį, antspaudas būtų kitas; jei pasikeistų bent žodis ispaniškoje versijoje, ispaniškas antspaudas pasikeistų. Antspaudas neapsaugo turinio – tam yra kiti įrankiai – bet jis jį vienareikšmiškai identifikuoja. Ir to, kad ir kaip kukliai tai skambėtų, pakanka, kad jokia leidybos grandinės dalis negalėtų

pakeisti to, kas pasakyta, to nepastebėjus. Visa kita – šifravimas, pasirašymas, identifikavimas – statoma ant šios paprastos idėjos.

Redaktoriaus pastaba: kai šiose Cuadernos minimos įmonės ar produktai, tai nėra kaltinimas. Tie, kurie juos kuria, atlieka darbą, kurį milijonai žmonių naudoja ir vertina. Mes nurodome struktūrinį dalyką — modelį, o ne prekės ženklą. Prekės ženklai pateikiami kaip pavyzdžiai, nes juos skaitytojas atpažįsta.

Šaltiniai ir papildomas skaitymas

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, 2015 m. rugpjūtis. Oficiali SHA-2 šeimos specifikacija, įskaitant SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, 2011 m. gegužė. Normatyvinė versija programuotojams.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). 5 ir 6 skyriai apima maišos funkcijas ir jų teisėtą bei neteisėtą naudojimą.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Praktinis SHA-256 naudojimo pavyzdys blokams jungti nekintamoje struktūroje.
- Reglamentas (ES) 910/2014 (eIDAS) — kvalifikuotų laiko žymų teikėjų sistema. SHA-256 yra etaloninė funkcija kvalifikuotiems elektroniniams parašams ir antspaudams ES.
- Etaloninis įgyvendinimas „Zig“ kalba: `std.crypto.hash.sha2.Sha256` oficialioje kalbos saugykloje (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Tai optimizuota ir audituota versija, kurią faktiškai naudoja „Solo2“. Naudinga palyginti su mokomuoju įgyvendinimu priede.

← AnkstesnisSchrems II, praėjus penkeriems metamsKitas →Kill switch ir institucinis užvaldymas

Naujausi skaitiniai

- Analizė · 2026 m. gegužės 18 d. Tikras vs tariamas privatumas: klausimai, kuriuos verta užduoti sau
- Analizė · 2026 m. gegužės 18 d. Self-hosting kaip profesinė praktika
- Koncepcija · 2026 m. gegužės 18 d. 24 žodžiai: kas yra kriptografinė tapatybė

Pasiimkite šį straipsnį su savimi ten, kur jums reikia.

↓ Markdown ↓ Paprastas tekstas ↓ PDF

Failas bus atsisiųstas į jūsų įrenginį. Iš ten galite jį išsaugoti, importuoti į Solo2 arba bendrinti bet kur. Cuadernos nusprendžia ne jūsų naudai dėl paskirties vietos.

Vaško antspaudas · SHA-256 cc74a9edaba461f47e31b6bf7826631acf69c3d2a98ea0fc63d75013de7f8335

Cuadernos Lacre · Menzuri Gestión S.L. leidinys ·
parašė R.Eugenio · redagavo Solo2 komanda.

Ši svetainė nenaudoja slapukų ir neįkelia trečiųjų šalių išteklių. Naudojamas priglobtas anoniminis lankytojų skaitiklis („Umami“, mūsų Europos serveryje) ir minimalus „JavaScript“, reikalingas dviem antraštės valdikliams: šviesiai arba tamsiai temai ir kalbos pasirinkimui. Be seklių, be profiliavimo, be dalijimosi duomenimis. Jei norite mus sekti: RSS.