Co je vlastně SHA-256

Matematický otisk, který se vejde do šedesáti čtyř znaků a který se celý změní, pokud se v původním textu pohne byť jen jediná čárka. Proč mu říkáme digitální pečetní vosk.

Abychom si rozuměli: Představte si stroj, který přečte jakýkoli text a vrátí sekvenci 64 znaků. Pokud je text identický, vyjde identická sekvence. Pokud změníte jedinou čárku, sekvence je úplně jiná. Tato sekvence je digitální pečetní vosk.

Jednoduchá myšlenka za technickým názvem

Představte si, že existuje stroj s jednou štěrbinou a jednou obrazovkou. Štěrbinou vložíte text: slovo, větu, celý román. Na obrazovce se po chvíli objeví sekvence přesně šedesáti čtyř znaků. Této sekvenci my profesionálové říkáme *hash* nebo *kryptografický souhrn*; pro běžného čtenáře ji zatím můžeme nazývat matematickým otiskem textu, podobně jako je otisk prstu otiskem člověka.

Pokud vložíte stejný text dvakrát, stroj pokaždé ukáže stejný otisk. Pokud vložíte text jen mírně odlišný —jedna posunutá čárka, velké písmeno změněné na malé— stroj ukáže otisk zcela odlišný od toho prvního. Ne podobný, ale odlišný. Tyto dvě vlastnosti dohromady —determinismus a citlivost— tvoří onu jednoduchou myšlenku. Vše ostatní u SHA-256 je jen mechanismus, který zajišťuje jejich správné fungování.

Je dobré hned na začátku říci, co stroj nedělá. Nešifruje text. Neskrývá ho. Neukládá ho. Stroj se na text podívá, vypočítá otisk a text zapomene. Otisk neumožňuje rekonstruovat text, který jej vytvořil; umožňuje pouze u kandidátního textu ověřit, zda se shoduje s originálem či nikoli. Proto říkáme, že jde o *jednosměrný* souhrn: cesta vede tam, ale ne zpět.

Hash není totéž co šifrování

Záměna je častá a je dobré ji vyjasnit: šifrování a hashování jsou odlišné operace. Šifrování spočívá v transformaci textu tak, aby jej pouze držitel klíče mohl vrátit do původní podoby. Hashování spočívá ve vytvoření otisku textu, ze kterého nelze původní text nikdy získat zpět, a to ani s klíčem, ani bez něj. První je z principu vratné; druhé je z principu nevratné.

Praktický důsledek je důležitý. Když aplikace říká „vaše heslo ukládáme šifrované“, existuje někdo, kdo má klíč k jeho dešifrování — v každém případě samotná aplikace. Když aplikace říká „vaše heslo ukládáme hashované“, samotná aplikace nemůže původní heslo přečíst, i kdyby chtěla; může pouze ověřit, zda to, co napíšete, znovu vytvoří stejný otisk. Druhý model, je-li proveden správně, je pro ukládání hesel mnohem vhodnější. Později uvidíme, proč „správně provedený“ vyžaduje víc než jen samotné SHA-256.

Čtyři vlastnosti, díky kterým je kryptografický hash užitečný

Hashovací funkce, která si zaslouží přívlastek *kryptografická*, splňuje čtyři vlastnosti:

1. **Determinismus.** Stejný vstup vždy vytvoří stejný otisk.
2. **Lavinový efekt.** Malá změna na vstupu vytvoří zcela odlišný otisk bez viditelné podobnosti s předchozím.
3. **Odolnost proti inverzi.** Na základě otisku není výpočetně proveditelné najít text, který jej vytvořil.
4. **Odolnost proti kolizím.** Není výpočetně proveditelné najít dva různé texty, které by vytvořily stejný otisk.

„Není výpočetně proveditelné“ neznamená „je matematicky nemožné“. Znamená to, že náklady v čase, energii a penězích na dosažení cíle o řády převyšují součet veškeré rozumně dostupné výpočetní kapacity. U SHA-256 se tato hranice měří v

tisících miliard let i pro ty nejoptimističtější scénáře se specializovaným hardwarem. Což je pro praktické účely čtenáře totéž jako „nelze“.

Konkrétně SHA-256

Název mluví za vše. SHA je zkratka pro *Secure Hash Algorithm*: algoritmus pro bezpečný hash. Číslo 256 udává velikost otisku v bitech: dvě stě padesát šest bitů, tedy třicet dva bajtů, které zobrazeny v šestnáctkové soustavě tvoří oněch šedesát čtyři znaků, které čtenář již zná. Standard publikoval americký NIST, orgán, který tyto funkce normalizuje, v roce 2001 jako součást rodiny SHA-2; současná verze standardu, FIPS 180-4, je z roku 2015.

Pro ty, kteří si ještě neuvědomují, co jsou bity a bajty:

```
1 bit    →  0 nebo 1       (vypínač: zapnuto nebo vypnuto)
1 bajt   →  8 bitů         (256 možných kombinací)
32 bajtů →  256 bitů      (otisk SHA-256)
```

Číslo 256 na konci názvu udává velikost otisku v bitech. V šestnáctkové soustavě —číselné soustavě se šestnácti symboly místo deseti— se těchto 256 bitů vejde přesně do 64 znaků. To je těch 64 znaků, které vidíte na konci každého Cuaderno.

Rozměry si zaslouží chvilku pozornosti. Dvě stě padesát šest bitů umožňuje dvě na dvě stě padesátou šestou různých hodnot: číslo se sedmdesáti osmi desítkovými ciframi, o několik řádů větší než odhadovaný počet atomů v pozorovatelném vesmíru. Každý text na světě —každá kniha, každý e-mail, každá zpráva— připadne na jednu z těchto hodnot. Pravděpodobnost, že by se dva různé texty shodovaly náhodou, je pro praktické účely nerozeznatelná od nuly.

Jak to vypadá v kódu

V jazyce Zig, ve kterém píšeme součásti systému Solo2, vypadá výpočet pečetě SHA-256 textu takto:

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

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

Právě jsme požádali standardní knihovnu Zig, aby vypočítala SHA-256 textu v uvozovkách. Po volání obsahuje proměnná *resumen* třicet dva bajtů, které tvoří pečeť v její syrové podobě; když se zobrazí na obrazovce v šestnáctkové soustavě, je to oněch šedesát čtyři znaků, které se objevují na konci tohoto článku. Pokud bychom změnili *Cuadernos Lacre* na *Cuadernos lacre* —jedno velké písmeno méně— pečeť by se celá změnila. To je v pěti řádcích ústřední vlastnost, na které stojí vše ostatní. Pro ty, kteří chtějí vidět, jak to funguje uvnitř, přikládáme na konci článku čitelnou verzi algoritmu s komentáři krok za krokem.

Proč mu říkáme pečetní vosk

V evropské korespondenci patnáctého až devatenáctého století listinu uzavíral pečetní vosk. Kapka roztaveného vosku, na ni přitisknuté pečetidlo a dopis byl označen neopakovatelným způsobem. Nechránilo to obsah před odhodlaným slídilem —papír se dal přečíst proti světlu, vosk se dal rozlomit— ale dalo to zásah jasně najevo. Jakákoli změna uzávěru byla pro příjemce viditelná ještě před otevřením listu. Vosk nezabránil poškození; dával ho najevo.

SHA-256 těla každého Cuaderno plní v digitální verzi stejnou funkci. Pokud by se v článku změnilo jediné slovo mezi okamžikem publikace a okamžikem, kdy jej čtete, šestnáctková pečeť na konci textu by již nesouhlasila s SHA-256 textu, který máte před sebou. Každý čtenář s pěti řádky kódu by si to mohl ověřit. Publikace nemůže přepsat svou historii, aniž by ji pečeť prozradila. Nechrání před poškozením; činí jej ověřitelným.

Co hash není

Od SHA-256 jsou někdy vyžadována čtyři použití, která mu nepřísluší:

1. **Šifrování.** Hash shrnuje; neskrývá. Pokud chcete, aby text nebyl čitelný, musíte jej zašifrovat, nikoli zahashovat.

2. **Autentizace autora.** Hash neříká, kdo text napsal, pouze jaký text byl zahashován. K přiřazení autorství je potřeba kryptografický podpis nad hashem, nikoli samotný hash.
3. **Ukládání hesel.** Zde je past, kterou je dobré pochopit. SHA-256 je navržen tak, aby byl velmi rychlý —což je pro mnoho věcí dobré, ale pro tuto špatné. Útočník se specializovaným hardwarem může proti hashi SHA-256 vyzkoušet miliardy hesel za sekundu, dokud nenajde to vaše. Pro ukládání hesel je třeba používat záměrně pomalé funkce pro derivaci klíče jako Argon2, scrypt nebo bcrypt v kombinaci se *solí* (náhodný údaj unikátní pro každého uživatele, který zabrání tomu, aby dva lidé se stejným heslem měli stejný hash).
4. **Čtení hashe jako identifikátoru autora.** Tím není. Hash identifikuje obsah. Pokud dva lidé zahashují slovo *ahoj* pomocí SHA-256, oba získají stejný souhrn — a to je ústřední vlastnost, nikoli vada: pokud by souhrny byly různé, nemohli bychom ověřit shodu mezi publikovaným a přijatým.

Kde se SHA-256 objevuje ve vašem každodenním životě

I když to nevidíte, SHA-256 zajišťuje velkou část toho, co na internetu denně používáte. Blockchain Bitcoinu je postaven na řetězení SHA-256 každého bloku k následujícímu; změna minulého bloku vyžaduje přepočítání celého následného řetězce. Git, systém, ve kterém se verzuje kód poloviny světa, identifikuje každé potvrzení (commit) pomocí SHA-256 (v novějších verzích) nebo jeho předchůdce SHA-1 (ve starších verzích) jeho kompletního obsahu. Certifikáty HTTPS, které ověřují identitu webu při vstupu, mají přidružený otisk SHA-256. Stažené programy jsou často doprovázeny hashem SHA-256 publikovaným vývojářem, abyste si mohli ověřit, že soubor nebyl cestou změněn. A jak jsme řekli, na konci každého Cuadernos Lacre.

Pro profesionálního čtenáře

Čtyři provozní připomínky pro ty, kteří rozhodují o systémech nebo je auditují:

1. Hash není šifrování. Pokud dodavatel ve své technické dokumentaci zaměňuje tyto dva termíny, je dobré se zeptat, co přesně tím myslí.
2. K ukládání hesel by se nikdy nemělo používat samotné SHA-256. SHA-256 je pro tento úkol příliš rychlé (viz bod 3 sekce *Co hash není*). Současným standardem je **Argon2id**: pomalý z principu, konfigurovatelný podle kapacity serveru, v kombinaci s různou náhodnou *solí* pro každého uživatele.
3. Pro integritu dokumentů —smluv, spisů, souborů— zůstává SHA-256 referenčním standardem. Používají jej kvalifikovaní poskytovatelé služeb vytváření časových razítek v EU.
4. Pro dlouhodobé uchovávání (desetiletí) je vhodné vypočítat a archivovat také SHA-3 nebo SHA-512 společně s SHA-256; kryptografická obezřetnost doporučuje nespoléhat se při stoleté archivaci na jedinou funkci.

Technicky vzato je tato iterovaná struktura — kde se mezistav uchovává mezi vstupními bloky — známá jako konstrukce **Merkle-Damgård**, což je vzor, na kterém jsou založeny SHA-1, SHA-2 (včetně SHA-256) a mnoho dalších klasických hashovacích funkcí. SHA-3 naproti tomu Merkle-Damgård opouští ve prospěch jiné architektury nazývané *sponge* (houba).

Jak funguje SHA-256, krok za krokem, lidskou řečí

Představte si, že jste sestavili nejpropracovanější domino dráhu na světě: tisíce kostek, desítky odboček, mechanické mosty a rampy křižující celou místnost, pečlivě umístěné kostku po kostce.

Pokud cvrnknete do první kostky, řetězec padá v přesném a opakovatelném pořadí. Stejná sestava, stejný počáteční impuls → identický výsledný vzor spadlých kostek, znovu a znovu.

Tady je to zajímavé: posuňte **jedinou kostku** o půl centimetru stranou, než začnete, a znovu cvrnkněte. Rampa, která se měla aktivovat, zůstane nehybná, most nespadne, spustí se úplně jiná odbočka. Výsledný vzor kostek na zemi je ve srovnání s tím prvním naprosto k nerozeznání.

SHA-256 je matematicky tato dráha. Text, který napíšete, je počáteční poloha kostek. Algoritmus je impuls, který uvolní kaskádu. A konečný výsledek — to, čemu říkáme *hash* — je statický snímek podlahy, když se vše zastavilo. Změňte jedinou čárku v původním textu a snímek bude radikálně odlišný. Tak jednoduché a tak drastické.

Krok 1. Přeložit text do binárních kostek. Počítače nerozumí písmenům; překládají je nejprve do čísel (ASCII) a čísla do binární soustavy (jedničky a nuly). Každé písmeno se změní na 8 bílých nebo černých kostek: *A* je `01000001`, *B* je `01000010`, mezera je `00100000`. Celý váš text — slovo, smlouva, román — se stane dlouhou řadou bílých a černých kostek.

Krok 2. Doplnit do standardní velikosti. Dráha zpracovává řadu v *úsecích* o délce přesně 512 kostek. Pokud vaše zpráva nedosahuje násobku 512, přidá se hned za text značící kostka (s hodnotou `10000000`) a poté nuly, dokud se úsek nedoplní. Posledních 64 pozic každého úseku je vyhrazeno pro zaznamenání původní délky textu. Dráha tak vždy ví, kde skončil skutečný obsah a kde začala výplň.

Krok 3. Umístit osm hlavních kostek. Než začneme, položíme na stůl **osm hlavních kostek** v přesné počáteční poloze. Těchto osm kostek není žádným tajemstvím: jejich počáteční hodnota je pevně daná veřejným matematickým pravidlem (odmocniny z prvních osmi prvočísel — 2, 3, 5, 7, 11, 13, 17, 19 — a první bity desetinné části každé odmocniny). Každý v jakémkoli koutě planety začíná se stejnými osmi hlavními kostkami ve stejné poloze. Jejich osudem je být posunuty a transformovány lavinou.

Krok 4. Velká lavina: šedesát čtyři kol postrčení. Tady začíná podívaná. První úsek 512 kostek vašeho textu narazí do osmi hlavních kostek. Nespadnou ale najednou: mechanismus provede **šedesát čtyři po sobě jdoucích kol**. V každém kole provede s kostkami tři operace:

- **Kolotoč** (rotace). Kostky se pohybují v kruhu: ty vpravo přecházejí doleva. Žádná kostka se neztratí ani nepřidá; prostě se přerovnají po úplném otočení kolotoče. Je to levný a vratný způsob redistribuce informací.
- **Logický trychtýř** (XOR). Kostky procházejí trychtýřem, který je porovnává po dvou: pokud mají obě stejnou barvu, vyjde bílá; pokud se liší, vyjde černá. Je to nejjednodušší operace binární logiky, ale v kombinaci s rotacemi kolotoče se stává nesmírně silnou pro míchání informací bez jejich ztráty.
- **Přetečení** (modulární součet). Výsledek se sečte s *kostkou s konstantním postrčením* vzatou z veřejného seznamu šedesáti čtyř konstant (třetí odmocniny z prvních šedesáti čtyř prvočísel). Pokud součet vygeneruje kostky navíc, které se nevejdou do vymezeného prostoru 32 kostek, tyto přebývající kostky se zahodí. Na stole je místo jen pro 32 kostek, ani o jednu víc.

Na konci šedesátého čtvrtého kola každá z kostek z úseku vašeho textu ovlivnila polohu osmi hlavních kostek. Energie postrčení prošla celou dráhou.

Krok 5. Přidat další úsek (bez restartování). Pokud byl váš text dlouhý a zbývá další úsek 512 kostek ke zpracování, **dráha se nerestartuje**. Osm hlavních kostek zůstane tak, jak je zanechala první lavina, a druhý úsek je vyslán proti nim, aby spustil dalších šedesát čtyři kol. Je to jako přidat novou místnost plnou domina na konec té, která právě spadla: nepořádek v té první zcela podmiňuje to, jak spadne ta druhá.

Krok 6. Pořídit finální snímek. Když už nezbývají žádné další úseky ke zpracování, lavina se zastaví. Podíváme se na konečnou polohu, v níž zůstalo osm hlavních kostek. Přeložíme jejich konfiguraci do kódu písmen a čísel v hexadecimální soustavě. Výsledkem je řetězec o délce přesně šedesáti čtyř znaků: to je váš pečeť SHA-256.

Z toho, jak je dráha sestavena, vyplývají samy o sobě čtyři vlastnosti:

1. **Determinismus.** Stejný text vytvoří vždy stejný finální snímek na jakémkoli počítači na světě. Nulová náhodnost, nula překvapení.
2. **Lavinový efekt.** Přidaná čárka, změněné velké písmeno, zapomenutý háček: snímek je naprosto k nerozeznání. To je ta extrémní citlivost, kterou jsme popsali již na začátku.
3. **Jednosměrnost.** Z finálního snímku nelze zrekonstruovat původní text. Rotace, trychtýře a přetečení ničí veškeré směrové informace o tom, *odkud který bit přišel*, a uchovávají pouze to, *co se celkem sečetlo*.
4. **Odolnost proti kolizím.** Za pětadvacet let veřejné kryptoanalýzy se nikomu nepodařilo najít dva různé texty, jejichž finální snímky by se shodovaly. A obtížnost takového úkolu je mimo výpočetní dosah jakékoli rozumně představitelné civilizace.

Následující dodatek s kódem implementuje přesně těchto šest kroků v jazyce Zig. Nyní jej můžete číst s vědomím, co která bitová operace znamená, místo abyste slepě přijímali manipulace.

Technický glosář

Pro čtenáře, který chce pochopit, co která operace dělá. Klidně jej přeskočte: článek je srozumitelný i bez něj.

ASCII a Unicode — jak se z písmen stávají čísla. Počítače nevidí písmena; vidí čísla. Standard zvaný **ASCII** (*American Standard Code for Information Interchange*, z roku 1963) přiřazuje každému znaku na klávesnici konkrétní číslo: *A* je 65, *B* je 66, *a* jest 97, *0* je 48, mezera je 32, čárka je 44. Moderní systémy jej rozšiřují o **Unicode**, který přiřazuje číslo každému znaku každé abecedy na světě: cyrilici, arabštině, čínštině, japonštině, a dokonce i emodži. Když napíšete znak nebo otevřete textový soubor, počítač čte skryté číslo, nikoli tvar na obrazovce. SHA-256 pracuje s těmito čísly a s jakýmkony textem zachází jako s dlouhou řadou číslic. Proto může stejným algoritmem zapečetit článek ve španělštině, báseň v japonštině i binární soubor.
XOR — bitový komparátor. XOR (vyslovováno *„ex-or“*, z anglického *exclusive or*, „exkluzivní nebo“) je jednou z nejjednodušších operací, které může počítač se dvěma binárními čísly provést. Porovnává dva bity na stejné pozici a vrací: **1**, pokud je právě jeden ze dvou 1 (jeden, ale ne oba), **0**, pokud jsou oba stejné (oba 0 nebo oba 1). Příklad: XOR čísel `1010` a `1100` je `0110`. Má pozoruhodnou vlastnost: je vratný — pokud provedete XOR dvakrát se stejným klíčem, vrátíte se k originálu. Proto je tahounem kryptografie: míchá bity bez ztráty informací, ale výsledek neprozrazuje nic o vstupech, pokud neznáte jeden z nich.
Hexadecimální soustava — počítání v základu 16. Téměř všechna čísla v běžném životě používají deset číslic (0-9). Hexadecimální soustava jich používá šestnáct: obvyklých 0-9 plus šest písmen představujících následující hodnoty: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Proč šestnáct? Protože počítače uvažují ve skupinách po čtyřech bitech a čtyři bity mohou reprezentovat přesně šestnáct různých hodnot — jeden hexadecimální znak tak čistě odpovídá čtyřem bitům. Otisky SHA-256 měří 256 bitů, což je přesně **64 hexadecimálních znaků**. Kdybychom je psali v běžné desítkové soustavě, zabraly by asi 78 číslic a byly by méně praktické. Volba je estetická a kompaktní; skryté číslo je stejné.
Bitová rotace — binární kolotoč. Představte si řadu sedmi žárovek, některé svítí (1) a jiné jsou zhasnuté (0): `1 0 1 1 0 0 1`. Rotace doprava o jednu pozici spočívá v tom, že vezmete žárovku úplně vpravo, přenesete ji na levý okraj a ostatní posunete o jedno místo doprava: `1 1 0 1 1 0 0`. Žádná žárovka se neztratí ani nepřibude: prostě tančí v kruhu. SHA-256 používá bitovou rotaci stokrát při každém výpočtu; je to levný a bezztrátový způsob redistribuce informací v rámci stavu.
Konstanty *„nothing-up-my-sleeve“* — proč pocházejí z prvočísel. Osm hlavních kostek a šedesát čtyři konstant kol v SHA-256 nebylo vybráno náhodně. Pocházejí z odmocnin a třetích odmocnin prvních prvočísel. Proč? Protože jejich tvůrci chtěli konstanty *„bez ničeho v rukávu“*: hodnoty, jejichž původ si může kdokoli ověřit. Kdyby vám někdo řekl *„věř mi: použij toto náhodné 32bitové číslo“*, oprávněně byste podezřívali skrytou slabinu nebo zadní vrátka. Ale kdokoli s kalkulačkou si může ověřit, že prvních 32 bitů odmocniny ze 2 je `0x6a09e667`. Hodnoty jsou matematické, veřejné a reprodukovovatelné: do receptu se nemůže vloudit žádná skrytá past.

Dodatek: SHA-256 v čitelném kódu

Tento dodatek je pro čtenáře, kteří chtějí vidět algoritmus zevnitř. Jde o didaktickou implementaci v jazyce Zig, která sleduje specifikaci FIPS 180-4. Není to verze, kterou používá Solo2 —ta skutečná se nachází v `std.crypto.hash.sha2.Sha256` ve standardní knihovně Zig, optimalizovaná a auditovaná. Algoritmus je však stejný: to, co zde vidíte, je krok za krokem to, co se děje, když ono pětiznakové volání vykonává svou práci.

```
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
}
```

Jakýkoli přepis do jiného jazyka, který zachová stejnou strukturu —výchozí konstanty, expanze plánu, šedesát čtyři kol, akumulace— vytvoří stejný výsledek. Algoritmus nemá žádná tajemství: jeho hodnota spočívá v tom, že výše uvedené vlastnosti stále platí i po dvou desetiletích veřejné kryptoanalýzy tisíci očí.

Pokud se vrátíte na konec tohoto článku, uvidíte šedesátičtyřmístnou šestnáctkovou pečeť. Je to SHA-256 textu, který jste právě dočetli, v tomto jazyce. Pokud bychom článek přeložili, pečeť by byla jiná; pokud by se v české verzi změnilo jediné slovo, česká pečeť by se změnila. Pečeť nechrání obsah —k tomu slouží jiné nástroje— ale jednoznačně jej identifikuje. A to, jakkoli skromně to zní, stačí k tomu, aby žádný krok v redakčním řetězci nemohl změnit řečené, aniž by si toho někdo všiml. Vše ostatní —šifrování, podepisování, identifikace— se staví na této jednoduché myšlence.

Poznámka redakce: Pokud tyto Cuadernos zmiňují firmy nebo produkty, není to za účelem obviňování. Ti, kteří je vytvářejí, odvádějí práci, kterou miliony lidí používají a oceňují. To, na co poukazujeme, je strukturální — model, nikoliv značka. Značky uvádíme jako příklad, protože jsou pro čtenáře rozpoznatelné.

Zdroje a další čtení

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, srpen 2015. Oficiální specifikace rodiny SHA-2, včetně SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, květen 2011. Normativní verze pro implementátory.

- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Kapitoly 5 a 6 se zabývají hashovacími funkcemi a jejich legitimním a nelegitimním použitím.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Praktický příklad použití SHA-256 k řetězení bloků v neměnné struktuře.
- Nařízení (EU) 910/2014 (eIDAS) — rámec pro kvalifikovaná časová razítka. SHA-256 je referenční funkcí pro kvalifikované elektronické podpisy a pečetě vydávané v EU.
- Referenční implementace v jazyce Zig: `std.crypto.hash.sha2.Sha256` v oficiálním repozitáři jazyka (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Jde o optimalizovanou a auditovanou verzi, kterou Solo2 skutečně používá. Užitečné pro srovnání s didaktickou implementací v dodatku.

← PředchozíSchrems II, o pět let pozdějiDalší →Kill switch a institucionální ovládnutí

Nedávné čtení

- Analýza · 18. května 2026 Skutečné vs. zdánlivé soukromí: otázky, které je vhodné si položit
- Analýza · 18. května 2026 Self-hosting jako profesionální praxe
- Koncept · 18. května 2026 24 slov: co je to kryptografická identita

Vezměte si tento článek tam, kam potřebujete.

↓ Markdown ↓ Prostý text ↓ PDF

Soubor se stáhne do vašeho zařízení. Odtud si jej můžete uložit, importovat do Solo2 nebo sdílet, kdekoli chcete. Cuadernos za vás o cíli nerozhoduje.

Pečeť · SHA-256 c522424f6535ba4ec9726e312e5d094fba08a553d542dc330b16ee13f6d6c8bb

Cuadernos Lacre · Publikace Menzuri Gestión S.L. ·
napsal R.Eugenio · rediguje tým Solo2.

Tento web nepoužívá cookies a nenačítá zdroje třetích stran. Používá anonymní počítadlo návštěv (Umami, na našem evropském serveru) a minimální nezbytný JavaScript pro dva ovládací prvky v záhlaví: světlý nebo tmavý motiv a výběr jazyka. Bez trackerů, bez profilování, bez sdílení dat. Pokud nás chcete sledovat: RSS.