Apa itu SHA-256 sebenarnya

Cap jari matematik yang memuatkan enam puluh empat aksara dan berubah sepenuhnya jika satu koma dalam teks asal dialihkan. Mengapa kita memanggilnya meterai lakri digital.

Secara ringkasnya: Bayangkan sebuah mesin yang membaca teks apa pun dan mengembalikan urutan 64 aksara. Jika teks yang masuk adalah serupa, urutan yang keluar juga serupa. Jika anda menggerakkan satu koma sahaja, urutannya akan menjadi sangat berbeza. Urutan itu adalah lakri digital.

Idea ringkas di sebalik nama teknikal

Bayangkan sebuah mesin dengan satu slot dan satu skrin. Anda memasukkan teks melalui slot tersebut: sepatah perkataan, sebaris ayat, atau sebuah novel lengkap. Di skrin muncul, sejurus selepas itu, urutan tepat enam puluh empat aksara. Urutan tersebut, bagi pembaca profesional kami memanggilnya *hash* atau *ringkasan kriptografi*; bagi pembaca umum, buat masa ini kita boleh memanggilnya cap jari matematik teks tersebut, sebagaimana cap jari bagi manusia.

Jika anda memasukkan teks yang sama dua kali, mesin menunjukkan cap jari yang sama kedua-dua kali. Jika anda memasukkan teks yang sedikit berbeza —satu koma dialihkan, huruf besar menjadi huruf kecil— mesin menunjukkan cap jari yang sama sekali berbeza daripada yang pertama. Bukan hampir sama: tetapi berbeza terus. Kedua-dua sifat ini bersama —determinisme dan sensitiviti— adalah idea ringkasnya. Segala-galanya tentang SHA-256 adalah mekanisme yang memastikan ia dipatuhi dengan baik.

Adalah wajar untuk menyatakan sejak awal apa yang mesin tersebut tidak lakukan. Ia tidak menyulitkan teks. Ia tidak menyembunyikannya. Ia tidak menyimpannya. Mesin tersebut melihat teks, mengira cap jari, dan melupakan teks tersebut. Cap jari tersebut tidak membenarkan pembinaan semula teks yang menghasilkannya; ia hanya membenarkan, berdasarkan teks calon, untuk menyemak sama ada ia sepadan dengan yang asal atau tidak. Itulah sebabnya kita katakan ia adalah ringkasan *satu hala*: ia pergi, dan tidak kembali.

Hash tidak sama dengan menyulit (encrypt)

Kekeliruan sering berlaku dan wajar diperjelaskan: menyulit (encrypt) dan melakukan hash adalah operasi yang berbeza. Menyulit melibatkan penukaran teks supaya hanya pemegang kunci yang boleh mengembalikannya kepada bentuk asal. Melakukan hash melibatkan penghasilan cap jari teks yang mana teks asal tidak dapat dipulihkan lagi, sama ada dengan kunci atau tanpa kunci. Yang pertama adalah boleh balik (reversible) mengikut reka bentuk; yang kedua, tidak boleh balik (irreversible) mengikut reka bentuk.

Kesan praktikalnya adalah penting. Apabila sebuah aplikasi berkata «kami menyimpan kata laluan anda yang disulitkan», ada seseorang yang mempunyai kunci untuk menyahsulitnya — dalam apa jua keadaan, aplikasi itu sendiri. Apabila sebuah aplikasi berkata «kami menyimpan kata laluan anda yang di-hash», aplikasi itu sendiri tidak dapat membaca kata laluan asal walaupun ia mahu; ia hanya boleh menyemak sama ada apa yang anda tulis menghasilkan cap jari yang sama semula. Model kedua, jika dilakukan dengan betul, adalah jauh lebih baik daripada yang pertama untuk menyimpan kata laluan. Kita akan melihat kemudian mengapa «dilakukan dengan betul» memerlukan sesuatu yang lebih daripada sekadar SHA-256 sahaja.

Empat sifat yang menjadikan hash kriptografi berguna

Fungsi hash yang layak dipanggil *kriptografi* memenuhi empat sifat:

1. **Determinisme.** Input yang sama sentiasa menghasilkan cap jari yang sama.
2. **Kesan runtuhan (Avalanche effect).** Perubahan kecil pada input menghasilkan cap jari yang sama sekali berbeza, tanpa sebarang persamaan yang kelihatan dengan yang sebelumnya.
3. **Rintangan terhadap penyongsangan.** Berdasarkan cap jari, adalah tidak berdaya maju secara komputasi untuk mencari teks yang menghasilkannya.
4. **Rintangan terhadap perlanggaran.** Adalah tidak berdaya maju secara komputasi untuk mencari dua teks berbeza yang menghasilkan cap jari yang sama.

«Tidak berdaya maju secara komputasi» tidak bermakna «mustahil secara matematik». Ia bermakna kos dari segi masa, tenaga dan wang untuk mencapainya melebihi magnitud jumlah keseluruhan kapasiti pengkomputeran yang tersedia secara munasabah. Bagi SHA-256, had tersebut diukur dalam ribuan trilion tahun walaupun untuk pendekatan paling optimis dengan perkakasan khas. Yang mana, untuk tujuan praktikal pembaca, adalah sama dengan «tidak boleh dilakukan».

SHA-256 secara khusus

Namanya sudah menjelaskan segalanya. SHA bermaksud *Secure Hash Algorithm*: algoritma hash selamat. Nombor 256 menunjukkan saiz cap jari dalam bit: dua ratus lima puluh enam bit, iaitu tiga puluh dua bait, yang dipaparkan dalam bentuk heksadesimal adalah enam puluh empat aksara yang pembaca sudah kenali. Standard ini diterbitkan oleh NIST Amerika Syarikat, badan yang menstandardkan fungsi jenis ini, pada tahun 2001 sebagai sebahagian daripada keluarga SHA-2; versi standard yang sedang berkuat kuasa, FIPS 180-4, adalah dari tahun 2015.

Bagi mereka yang masih belum memahami apa itu bit dan bait:

```
1 bit    →  0 atau 1      (suis: hidup atau mati)
1 byte   →  8 bits       (256 kombinasi mungkin)
32 bytes →  256 bits     (cap jari SHA-256)
```

Nombor 256 di hujung nama menyatakan saiz cap jari dalam bit. Dalam heksadesimal —satu sistem penomboran dengan enam belas simbol dan bukannya sepuluh— 256 bit tersebut muat dalam tepat 64 aksara. Itulah 64 aksara yang anda lihat di kaki setiap Cuaderno.

Dimensinya layak diberi perhatian sejenak. Dua ratus lima puluh enam bit membenarkan dua kuasa dua ratus lima puluh enam nilai berbeza: sebuah nombor dengan tujuh puluh lapan digit perpuluhan, beberapa magnitud lebih besar daripada anggaran jumlah atom dalam alam semesta yang boleh diperhatikan. Setiap teks di dunia —setiap buku, setiap e-mel, setiap mesej— jatuh pada salah satu daripada nilai tersebut. Kebarangkalian untuk dua teks berbeza bertindih secara kebetulan adalah, untuk tujuan praktikal, tidak dapat dibezakan daripada sifar.

Bagaimana ia kelihatan dalam kod

Dalam Zig, bahasa yang kami gunakan untuk menulis bahagian yang menyokong Solo2, mengira meterai SHA-256 bagi satu teks kelihatan seperti ini:

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

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

Kami baru sahaja meminta perpustakaan standard Zig untuk mengira SHA-256 bagi teks dalam petikan. Selepas panggilan tersebut, pembolehubah *resumen* mengandungi tiga puluh dua bait yang membentuk meterai dalam bentuk mentahnya; apabila dipaparkan di skrin dalam bentuk heksadesimal, ia adalah enam puluh empat aksara yang muncul di kaki artikel ini. Jika kita menukar *Cuadernos Lacre* kepada *Cuadernos lacre* —satu huruf besar dikurangkan— meterai tersebut akan berubah sepenuhnya. Itulah, dalam lima baris, sifat utama yang menyokong selebihnya. Bagi sesiapa yang ingin melihat cara ia berfungsi secara dalaman, di hujung artikel kami sertakan versi algoritma yang mudah dibaca dengan ulasan langkah demi langkah.

Mengapa kami memanggilnya meterai lakri

Dalam surat-menyurat Eropah dari abad ke-15 hingga ke-19, lakri menutup surat tersebut. Setitis lilin cair, sebentuk meterai ditekan di atasnya, dan surat itu ditandakan dengan cara yang tidak boleh diulang. Ia tidak melindungi kandungan daripada pengintip yang gigih —kertas tersebut boleh dibaca dengan menembusi cahaya, lakri tersebut boleh dipecahkan — tetapi ia memberi bukti. Sebarang gangguan pada penutup adalah jelas kepada penerima sebelum kertas itu dibuka. Lakri tidak menghalang kerosakan; ia mengisytiharkannya.

SHA-256 pada badan setiap Cuaderno memenuhi fungsi yang sama dalam versi digitalnya. Jika sepatah perkataan pun dalam artikel ini berubah antara masa ia diterbitkan dan masa anda membacanya, meterai heksadesimal di kaki teks tidak lagi akan sepadan dengan SHA-256 teks yang ada di hadapan anda. Mana-mana pembaca dengan lima baris kod boleh menyemaknya. Penerbitan tidak boleh menulis semula sejarahnya tanpa meterai tersebut mendedahkannya. Ia tidak melindungi daripada kerosakan; ia menjadikannya boleh disahkan.

Apa yang bukan fungsi hash

Empat kegunaan kadangkala diminta daripada SHA-256 yang sebenarnya tidak sesuai untuknya:

1. **Menyulit (Encrypt).** Hash meringkaskan; ia tidak menyembunyikan. Jika anda mahu teks tersebut tidak boleh dibaca, anda perlu menyulitkannya, bukan melakukan hash.
2. **Mengesahkan pengarang.** Hash tidak memberitahu siapa yang menulis teks tersebut, hanya teks apa yang di-hash. Untuk mengaitkan kepengarangan, tandatangan kriptografi diperlukan di atas hash tersebut, bukan sekadar hash sahaja.
3. **Menyimpan kata laluan.** Di sini terdapat perangkap yang perlu difahami. SHA-256 direka untuk menjadi sangat pantas —yang mana ia bagus untuk banyak perkara, tetapi buruk untuk tujuan ini. Penyerang dengan perkakasan khas boleh mencuba berbilion kata laluan sesaat terhadap hash SHA-256 sehingga menemui milik anda. Untuk menyimpan kata laluan, fungsi terbitan kunci yang sengaja dilambatkan seperti Argon2, scrypt atau bcrypt mesti digunakan, digabungkan dengan *sal* (data rawak unik bagi setiap pengguna, yang menghalang dua orang dengan kata laluan yang sama daripada mempunyai hash yang sama).
4. **Membaca hash sebagai pengecam pengarang.** Ia bukan begitu. Hash mengenal pasti kandungan. Jika dua orang melakukan hash pada perkataan *hola* dengan SHA-256, kedua-duanya mendapat ringkasan yang sama — dan itu adalah sifat utamanya, bukannya kecacatan: jika ia adalah ringkasan yang berbeza, kita tidak akan dapat mengesahkan persamaan antara apa yang diterbitkan dan apa yang diterima.

Di mana SHA-256 muncul dalam kehidupan seharian anda

Walaupun anda tidak melihatnya, SHA-256 menyokong sebahagian besar daripada apa yang anda gunakan setiap hari di internet. Rantai blok Bitcoin dibina dengan merantai SHA-256 setiap blok kepada blok seterusnya; mengubah blok yang lalu memaksa pengiraan semula keseluruhan rantai seterusnya. Git, sistem yang digunakan untuk versi kod sebahagian besar dunia, mengenal pasti setiap pengesahan (commit) melalui SHA-256 (dalam versi terkini) atau melalui pendahulunya SHA-1 (dalam versi lama) daripada keseluruhan kandungannya. Sijil HTTPS yang mengesahkan identiti sesebuah laman web apabila anda melayarinya membawa cap jari SHA-256 yang dikaitkan. Muat turun perisian sering kali disertakan dengan SHA-256 yang diterbitkan oleh pembangun supaya anda boleh mengesahkan bahawa fail tersebut tidak diubah semasa dalam perjalanan. Dan, seperti yang telah kami katakan, di kaki setiap Cuaderno Lacre.

Untuk pembaca profesional

Empat peringatan operasi bagi sesiapa yang membuat keputusan atau mengaudit sistem:

1. Hash bukan penyulitan. Jika pembekal mengelirukan kedua-dua istilah tersebut dalam dokumentasi teknikal mereka, adalah wajar untuk bertanya apa sebenarnya yang mereka maksudkan.
2. Untuk menyimpan kata laluan, jangan sesekali menggunakan SHA-256 sahaja. SHA-256 terlalu pantas untuk tugas ini (lihat perkara 3 dalam *Apa yang bukan fungsi hash*). Standard semasa ialah **Argon2id**: lambat mengikut reka bentuk, boleh dikonfigurasi mengikut kapasiti pelayan, digabungkan dengan *sal* (salt) rawak yang berbeza bagi setiap pengguna.
3. Untuk integriti dokumen —kontrak, fail, arkib— SHA-256 kekal sebagai standard rujukan. Ia digunakan oleh pembekal meterai masa berkelayakan di EU.
4. Untuk pemeliharaan jangka panjang (berdekad), adalah wajar untuk mengira dan mengarkibkan juga SHA-3 atau SHA-512 bersama-sama SHA-256; kebijaksanaan kriptografi mengesyorkan supaya tidak bergantung pada satu

fungsi sahaja untuk arkib yang bertahan berabad-abad.

Secara teknikal, struktur berulang ini — di mana keadaan perantaraan dikekalkan antara blok input — dikenali sebagai pembinaan **Merkle-Damgård**, corak yang menjadi asas kepada SHA-1, SHA-2 (termasuk SHA-256) dan banyak fungsi hash klasik yang lain. Sebaliknya, SHA-3 meninggalkan Merkle-Damgård untuk memihak kepada seni bina berbeza yang dipanggil *span* (sponge).

Bagaimana SHA-256 berfungsi: langkah demi langkah dalam bahasa mudah

Bayangkan anda telah membina litar domino yang paling rumit di dunia: beribu-ribu kepingan, berpuluh-puluh simpangan, jambatan mekanikal dan tanjakan yang merentasi seluruh bilik, disusun dengan teliti satu demi satu.

Jika anda menyentuh kepingan pertama, rantaian itu akan jatuh dalam urutan yang tepat dan boleh diulang. Susunan yang sama, sentuhan permulaan yang sama → corak akhir kepingan yang jatuh adalah identik, berkali-kali.

Di sinilah letaknya keunikan: gerakkan **hanya satu kepingan** setengah sentimeter ke tepi sebelum bermula dan sentuh semula. Tanjakan yang sepatutnya diaktifkan kekal tidak bergerak, jambatan tidak jatuh, simpangan berbeza tercetus. Corak akhir kepingan di lantai adalah benar-benar tidak dapat dikenali berbanding yang pertama.

Secara matematik, SHA-256 adalah litar ini. Teks yang anda tulis adalah kedudukan awal kepingan domino. Algoritma adalah sentuhan yang melepaskan lata tersebut. Dan hasil akhir — apa yang kita panggil *hash* — adalah foto pegun lantai apabila semuanya telah berhenti. Ubah satu koma daripada teks asal dan foto tersebut akan menjadi radikal berbeza. Begitu mudah, dan begitu drastik.

Langkah 1. Menterjemah teks kepada kepingan binari. Komputer tidak faham huruf; ia menterjemahkannya terlebih dahulu kepada nombor (ASCII) dan nombor tersebut kepada binari (satu dan sifar). Setiap huruf menjadi 8 kepingan putih atau hitam: *A* ialah 65 (ASCII), *B* ialah 66. Dalam memori komputer, *A* menjadi `01000001`, *B* – `01000010`, ruang – `00100000`. Keseluruhan teks anda — sepatah kata, kontrak, novel — menjadi barisan panjang kepingan putih dan hitam.

Langkah 2. Mengisi sehingga saiz standard. Litar memproses barisan tersebut dalam *blok* tepat 512 kepingan. Jika mesej anda tidak mencapai gandaan 512, kepingan penanda (dengan nilai `10000000`) ditambah sejurus selepas teks dan kemudian sifar sehingga melengkapkan blok. 64 kedudukan terakhir setiap blok dikhaskan untuk mencatatkan panjang asal teks. Oleh itu, litar sentiasa tahu di mana kandungan sebenar berakhir and di mana pengisian bermula.

Langkah 3. Meletakkan lapan kepingan induk. Sebelum bermula, kita letakkan di atas meja **lapan kepingan induk** dalam kedudukan awal yang tepat. Lapan kepingan ini bukan rahsia: nilai awalnya ditetapkan oleh peraturan matematik awam (punca kuasa dua bagi lapan nombor perdana pertama — 2, 3, 5, 7, 11, 13, 17, 19 — dan bit pertama bagi bahagian perpuluhan setiap punca kuasa). Semua orang di mana-mana sahaja di planet ini bermula dengan lapan kepingan induk yang sama dalam kedudukan yang sama. Nasib mereka adalah untuk ditolak dan diubah oleh runtuhan tersebut.

Langkah 4. Runtuhan besar: enam puluh empat pusingan tolakan. Di sinilah persembahan bermula. Blok 512 kepingan pertama teks anda dilanggarkan dengan lapan kepingan induk. Tetapi ia tidak jatuh sekali gus: mekanisme menjalankan **enam puluh empat pusingan berturut-turut**. Dalam setiap pusingan, ia melakukan tiga operasi dengan kepingan tersebut:

- **Karusel** (putaran). Kepingan bergerak dalam bulatan: kepingan di sebelah kanan berpindah ke kiri. Tiada kepingan yang hilang atau ditambah; ia hanya disusun semula dengan membuat pusingan lengkap di karusel. Ia adalah cara yang murah dan boleh balik untuk mengagihkan semula maklumat.
- **Corong Logik** (XOR). Kepingan melalui corong yang membandingkannya dua demi dua: jika kedua-duanya berwarna sama, kepingan putih keluar; jika berbeza, kepingan hitam keluar. Ia adalah operasi paling mudah dalam logik binari, tetapi digabungkan dengan putaran karusel, ia menjadi sangat berkuasa untuk mencampurkan maklumat tanpa kehilangannya.
- **Limpahan** (penambahan modular). Hasilnya ditambah dengan *kepingan tolakan malar* yang diambil daripada senarai awam enam puluh empat pemalar (punca kuasa tiga bagi enam puluh empat nombor perdana pertama). Jika penambahan menghasilkan kepingan tambahan yang tidak muat dalam ruang 32 kepingan yang disediakan, kepingan yang berlebihan itu dibuang. Meja hanya mempunyai ruang untuk 32 kepingan, tidak lebih.

Pada akhir pusingan enam puluh empat, setiap kepingan blok teks anda telah mempengaruhi kedudukan lapan kepingan induk. Tenaga tolakan telah bergerak ke seluruh litar.

Langkah 5. Menambah blok seterusnya (tanpa mula semula). Jika teks anda panjang and terdapat satu lagi blok 512 kepingan untuk diproses, **litar tidak bermula semula**. Lapan kepingan induk kekal seperti yang ditinggalkan oleh runtuhan pertama, dan blok kedua dilancarkan terhadapnya untuk mengaktifkan enam puluh empat pusingan lagi. Ia seperti menambah bilik baharu yang penuh dengan domino di hujung bilik yang baru jatuh: keadaan bilik pertama menentukan sepenuhnya bagaimana bilik kedua akan jatuh.

Langkah 6. Mengambil foto final. Apabila tiada lagi blok untuk diproses, runtuhan berhenti. Kita melihat kedudukan akhir di mana lapan kepingan induk berada. Kita menterjemah konfigurasinya kepada kod huruf dan nombor dalam sistem perpuluhan perenam (heksadesimal). Hasilnya ialah rentetan tepat enam puluh empat aksara: itulah mohor SHA-256 anda.

Empat sifat muncul dengan sendirinya daripada bagaimana litar ini dipasang:

1. **Determinisme.** Teks yang sama sentiasa menghasilkan foto final yang sama pada mana-mana komputer di dunia. Sifar kerawakan, sifar kejutan.
2. **Kesan runtuhan (Avalanche effect).** Satu koma ditambah, satu huruf besar diubah, satu tanda baca dilupakan: foto tersebut menjadi tidak dapat dikenali sepenuhnya. Inilah sensitiviti melampau yang telah kita huraikan pada mulanya.
3. **Satu hala sahaja.** Berdasarkan foto final, anda tidak boleh membina semula teks asal. Putaran, corong dan limpahan memusnahkan semua maklumat arah tentang *dari mana datangnya setiap bit* dan hanya mengekalkan *apa yang ditambah secara keseluruhan*.
4. **Ketahanan terhadap pelanggaran (Collision resistance).** Dalam dua puluh lima tahun analisis kripto awam, tiada siapa yang berjaya mencari dua teks berbeza yang foto finalnya sama. Dan kesukaran untuk melakukannya adalah di luar jangkauan keupayaan pengiraan mana-mana tamadun yang boleh dibayangkan secara munasabah.

Lampiran kod yang menyusul melaksanakan enam langkah ini dengan tepat dalam Zig. Kini anda boleh membacanya dengan mengetahui maksud setiap operasi bit, dan bukannya menerima manipulasi tersebut secara buta.

Glosari teknikal

Untuk pembaca yang ingin memahami apa yang dilakukan oleh setiap operasi. Sila langkau secara bebas: artikel ini masih boleh difahami tanpanya.

ASCII dan Unicode — bagaimana huruf menjadi nombor. Komputer tidak melihat huruf; ia melihat nombor. Standard yang dipanggil **ASCII** (*American Standard Code for Information Interchange*, dari 1963) memberikan nombor khusus kepada setiap aksara papan kekunci: *A* ialah 65, *B* ialah 66, *a* ialah 97, *0* ialah 48, ruang ialah 32, koma ialah 44. Sistem moden meluaskannya dengan **Unicode**, yang memberikan nombor kepada setiap aksara dalam setiap abjad di dunia: Cyrillic, Arab, Cina, Jepun, and juga emoji. Apabila anda menulis aksara atau membuka fail teks, komputer membaca nombor di sebaliknya, bukan rupa pada skrin. SHA-256 berfungsi berdasarkan nombor-nombor ini, menganggap sebarang teks sebagai urutan nombor yang panjang. Sebab itu ia boleh memeterai artikel dalam bahasa Sepanyol, puisi dalam bahasa Jepun and fail binari dengan algoritma yang sama.
XOR — pembanding bit demi bit. XOR (disebut *«exor»*, dari bahasa Inggeris *exclusive or*, «atau eksklusif») adalah salah satu operasi paling mudah yang boleh dilakukan oleh komputer dengan dua nombor binari. Ia membandingkan dua bit kedudukan demi kedudukan and mengembalikan: **1** jika tepat satu daripada keduanya adalah 1 (satu tetapi bukan kedua-duanya), **0** jika kedua-duanya sama (kedua-duanya 0 atau kedua-duanya 1). Contoh: XOR bagi `1010` and `1100` ialah `0110`. Ia mempunyai sifat yang luar biasa: ia boleh balik — jika anda melakukan XOR dua kali dengan kunci yang sama, anda akan kembali kepada asal. Sebab itulah ia menjadi tonggak kriptografi: mencampurkan bit tanpa kehilangan maklumat, tetapi hasilnya tidak mendedahkan apa-apa tentang input jika anda tidak mengetahui salah satunya.
Heksadesimal — mengira dalam asas 16. Hampir semua nombor harian menggunakan sepuluh digit (0-9). Heksadesimal menggunakan enam belas: 0-9 biasa ditambah enam huruf yang mewakili nilai berikut: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Mengapa enam belas? Kerana komputer berfikir dalam kumpulan empat bit, and empat bit boleh mewakili tepat enam belas nilai berbeza — jadi, satu aksara heksadesimal sepadan dengan kemas kepada empat bit. Cap jari SHA-256 berukuran 256 bit, iaitu tepat **64 aksara heksadesimal**. Jika kita menulisnya dalam perpuluhan biasa, ia akan mengambil masa kira-kira 78 digit and akan menjadi lebih tidak selesa. Pilihan ini adalah estetik and padat; nombor di sebaliknya adalah sama.
Putaran bit — karusel binari. Bayangkan barisan tujuh mentol lampu, ada yang menyala (1) and ada yang mati (0): `1 0 1 1 0 0 1`. Memutar ke kanan satu kedudukan terdiri daripada mengambil mentol di hujung kanan sekali, membawanya ke hujung kiri and menggerakkan yang lain satu tempat ke kanan: `1 1 0 1 1 0 0`. Tiada mentol yang hilang atau ditambah: ia hanya menari dalam bulatan. SHA-256 menggunakan putaran bit beratus-ratus kali dalam setiap pengiraan; ia adalah cara yang murah and tanpa kehilangan untuk mengagihkan semula maklumat dalam keadaan tersebut.

Pemalar *«nothing-up-my-sleeve»* — mengapa ia berasal dari nombor perdana. Lapan kepingan induk and enam puluh empat pemalar pusingan SHA-256 tidak dipilih secara rawak. Ia berasal dari punca kuasa dua and punca kuasa tiga bagi nombor perdana pertama. Mengapa? Kerana pereka bentuknya mahukan pemalar *«tanpa apa-apa di dalam lengan baju»*: nilai yang asal-usulnya boleh disahkan oleh sesiapa sahaja. Jika seseorang memberitahu anda *«percayalah saya: gunakan nombor rawak 32 bit ini»*, anda akan mengesyaki secara munasabah adanya kelemahan tersembunyi atau pintu belakang. Tetapi sesiapa sahaja yang mempunyai kalkulator boleh mengesahkan bahawa 32 bit pertama punca kuasa dua bagi 2 ialah `0x6a09e667`. Nilainya adalah matematik, awam and boleh dihasilkan semula: tiada perangkap tersembunyi yang boleh menyelinap ke dalam resipi itu.

Lampiran: SHA-256 dalam kod yang mudah dibaca

Lampiran ini adalah untuk pembaca yang ingin melihat algoritma dari dalam. Ia adalah implementasi didaktik dalam Zig yang mengikuti spesifikasi FIPS 180-4. Ia bukan versi yang digunakan oleh Solo2 —yang sebenar berada dalam `std.crypto.hash.sha2.Sha256` dalam perpustakaan standard Zig, dioptimumkan dan diaudit—. Tetapi algoritmanya adalah sama: apa yang anda lihat di sini adalah, langkah demi langkah, apa yang berlaku apabila panggilan lima aksara itu melaksanakan tugasnya.

```
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
}
```

Sebarang penulisan semula dalam bahasa lain yang mengikuti struktur yang sama —pemalar awal, pengembangan jadual (schedule expansion), enam puluh empat pusingan, pengumpulan— menghasilkan keputusan yang sama. Algoritma ini tidak mempunyai rahsia: nilainya terletak pada sifat-sifat yang disenaraikan di atas yang tetap bertahan selepas dua dekad analisis kripto awam oleh ribuan mata.

Jika anda kembali ke kaki artikel ini, anda akan melihat meterai heksadesimal enam puluh empat aksara. Ia adalah SHA-256 bagi teks yang baru anda baca, dalam bahasa ini. Jika kami menterjemah artikel ini, meterainya akan berbeza; jika sepatah perkataan dalam versi Sepanyol berubah, meterai Sepanyol akan berubah. Meterai tersebut tidak melindungi kandungan —terdapat alat lain untuk itu— sebaliknya ia mengenal pasti kandungan secara unik. Dan itu, walaupun kedengaran sederhana, sudah memadai supaya tiada langkah dalam rantaian editorial boleh mengubah apa yang dikatakan tanpa disedari. Selebihnya —menyulit, menandatangani, mengenal pasti— dibina di atas idea ringkas ini.

Nota editor: apabila Cuadernos ini menyebut nama syarikat atau produk, ia bukan bertujuan untuk menuduh. Mereka yang membinanya melakukan tugas yang digunakan dan dihargai oleh berjuta-juta orang. Apa yang kami tekankan adalah isu struktur — modelnya, bukan jenamanya. Jenama muncul sebagai contoh kerana itulah yang dikenali oleh pembaca.

Sumber dan bacaan lanjut

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, Ogos 2015. Spesifikasi rasmi keluarga SHA-2, termasuk SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, Mei 2011. Versi normatif untuk pembangun.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Bab 5 dan 6 merangkumi fungsi hash serta kegunaannya yang sah dan tidak sah.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Contoh praktikal penggunaan SHA-256 untuk merantai blok dalam struktur yang tidak boleh diubah mengikut binaannya.
- Peraturan (UE) 910/2014 (eIDAS) — rangka kerja bagi pembekal meterai masa berkelayakan. SHA-256 adalah fungsi rujukan bagi tandatangan dan meterai elektronik berkelayakan yang dikeluarkan di EU.
- Implementasi rujukan dalam Zig: `std.crypto.hash.sha2.Sha256` dalam repositori rasmi bahasa tersebut (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Ia adalah versi yang dioptimumkan dan diaudit yang sebenarnya digunakan oleh Solo2. Berguna untuk membandingkan dengan implementasi didaktik dalam lampiran.

← SebelumnyaSchrems II, lima tahun kemudianSeterusnya →Kill switch dan penangkapan institusi

Bacaan terkini

- Analisis · 18 Mei 2026 Privasi nyata vs semu: soalan yang perlu anda tanya diri sendiri
- Analisis · 18 Mei 2026 Self-hosting sebagai amalan profesional
- Konsep · 18 Mei 2026 24 perkataan: apakah itu identiti kriptografi

Bawa artikel ini bersama anda ke mana sahaja anda memerlukannya.

↓ Markdown ↓ Teks biasa ↓ PDF

Fail akan dimuat turun ke peranti anda. Dari sana anda boleh menyimpannya, mengimportnya ke Solo2, atau berkongsinya di mana sahaja anda mahu. Cuadernos tidak memutuskan destinasi untuk anda.

Mohor lilin · SHA-256 b99b2f80e7c7147e6b7880a8439a1842c48c1317e7096439cf9ce8f938429efe

Cuadernos Lacre · Penerbitan daripada Menzuri Gestión S.L. ·
ditulis oleh R.Eugenio · disunting oleh pasukan Solo2.

Laman web ini tidak menggunakan kuki dan tidak memuatkan sumber pihak ketiga. Ia menggunakan pembilang lawatan tanpa nama yang dihoskan sendiri (Umami, pada pelayan Eropah kami) dan JavaScript minimum yang diperlukan untuk dua kawalan pengepala: tema terang atau gelap, dan pemilih bahasa. Tiada penjejak, tiada pemprofilan, tiada perkongsian data. Jika anda ingin mengikuti kami: RSS.