Konsep · 19 Mei 2026

← Sebelumnya·Berikutnya →

Apa itu SHA-256 sebenarnya

Jejak matematika yang muat dalam enam puluh empat karakter dan berubah sepenuhnya jika satu koma saja dari teks asli dipindahkan. Mengapa kami menyebutnya segel lak digital.

Ide sederhana di balik nama teknis

Bayangkan ada sebuah mesin dengan satu slot dan satu layar. Melalui slot tersebut Anda memasukkan teks: sebuah kata, sebuah kalimat, sebuah novel utuh. Di layar muncul, beberapa saat kemudian, urutan tepat enam puluh empat karakter. Urutan itu, bagi pembaca profesional kami menyebutnya *hash* atau *ringkasan kriptografi*; bagi pembaca umum, untuk saat ini kita bisa menyebutnya jejak matematika dari teks tersebut, sebagaimana sidik jari bagi seseorang.

Jika Anda memasukkan teks yang sama dua kali, mesin akan menunjukkan jejak yang sama kedua kalinya. Jika Anda memasukkan teks yang sedikit berbeda — satu koma bergeser, huruf besar yang menjadi huruf kecil — mesin akan menunjukkan jejak yang sama sekali berbeda dari yang pertama. Tidak mirip: berbeda. Kedua sifat itu bersama-sama — determinisme dan sensitivitas — adalah ide sederhananya. Segala hal lain dari SHA-256 adalah mekanisme yang membuatnya bekerja dengan baik.

Perlu dikatakan sejak awal apa yang tidak dilakukan mesin tersebut. Ia tidak mengenkripsi teks. Tidak menyembunyikannya. Tidak menyimpannya. Mesin melihat teks, menghitung jejaknya, dan melupakan teks tersebut. Jejak itu tidak memungkinkan untuk menyusun kembali teks yang menghasilkannya; ia hanya memungkinkan, jika diberikan teks kandidat, untuk memeriksa apakah cocok atau tidak dengan aslinya. Itulah sebabnya kami katakan bahwa itu adalah ringkasan *satu arah*: pergi, tidak kembali.

Hash tidak sama dengan enkripsi

Kebingungan sering terjadi dan perlu diperjelas: enkripsi dan hashing adalah operasi yang berbeda. Enkripsi terdiri dari mengubah teks sedemikian rupa sehingga hanya pemilik kunci yang dapat mengembalikannya ke bentuk aslinya. Hashing terdiri dari menghasilkan jejak teks yang darinya teks asli tidak pernah bisa dipulihkan, baik dengan kunci maupun tanpa kunci. Yang pertama dapat dibalik secara desain; yang kedua tidak dapat dibalik secara desain.

Konsekuensi praktisnya penting. Ketika sebuah aplikasi mengatakan «kami menyimpan kata sandi Anda terenkripsi», ada seseorang yang memiliki kunci untuk mendekripsinya — aplikasi itu sendiri, dalam hal apa pun. Ketika sebuah aplikasi mengatakan «kami menyimpan kata sandi Anda ter-hash», aplikasi itu sendiri tidak dapat membaca kata sandi asli meskipun ia mau; ia hanya dapat memeriksa apakah apa yang Anda tulis menghasilkan jejak yang sama lagi. Model kedua, jika dilakukan dengan benar, jauh lebih disukai daripada yang pertama untuk menyimpan kata sandi. Nanti kita akan melihat mengapa «dilakukan dengan benar» menuntut sesuatu yang lebih dari sekadar SHA-256 saja.

Empat sifat yang membuat hash kriptografi berguna

Fungsi hash yang layak mendapatkan kata sifat *kriptografi* memenuhi empat sifat:

1. **Determinisme.** Input yang sama selalu menghasilkan jejak yang sama.
2. **Efek longsoran (avalanche).** Perubahan kecil pada input menghasilkan jejak yang sama sekali berbeda, tanpa kemiripan yang terlihat dengan yang sebelumnya.
3. **Resistensi terhadap pembalikan.** Diberikan sebuah jejak, tidak layak secara komputasi untuk menemukan teks yang menghasilkannya.

4. **Resistensi terhadap tabrakan (collision).** Tidak layak secara komputasi untuk menemukan dua teks berbeda yang menghasilkan jejak yang sama.

«Tidak layak secara komputasi» bukan berarti «secara matematis tidak mungkin». Artinya biaya dalam waktu, energi, dan uang untuk mencapainya melebihi urutan besarnya dari jumlah total kapasitas komputasi yang tersedia secara wajar. Untuk SHA-256, batas itu diukur dalam ribuan triliun tahun bahkan untuk pendekatan paling optimis dengan perangkat keras khusus. Yang mana, untuk tujuan praktis pembaca, sama dengan «tidak bisa dilakukan».

SHA-256, secara khusus

Nama menjelaskan segalanya. SHA adalah singkatan dari *Secure Hash Algorithm*: algoritme hash aman. Angka 256 menunjukkan ukuran jejak dalam bit: dua ratus lima puluh enam bit, yaitu tiga puluh dua byte, yang ditampilkan dalam heksadesimal adalah enam puluh empat karakter yang sudah dikenali pembaca. Standar tersebut diterbitkan oleh NIST Amerika Serikat, badan yang menormalkan jenis fungsi ini, pada tahun 2001 sebagai bagian dari keluarga SHA-2; versi standar yang berlaku saat ini, FIPS 180-4, berasal dari tahun 2015.

Bagi yang belum paham apa itu bit dan byte:

```
1 bit    →  0 atau 1      (sebuah sakelar: nyala atau mati)
1 byte   →  8 bit         (256 kemungkinan kombinasi)
32 byte  →  256 bit       (jejak SHA-256)
```

Angka 256 di akhir nama menunjukkan ukuran jejak dalam bit. Dalam heksadesimal — sistem penomoran dengan enam belas simbol, bukan sepuluh — 256 bit tersebut muat dalam tepat 64 karakter. Itulah 64 karakter yang Anda lihat di bawah setiap Cuaderno.

Dimensi tersebut layak untuk direnungkan sejenak. Dua ratus lima puluh enam bit memungkinkan dua pangkat dua ratus lima puluh enam nilai berbeda: sebuah angka dengan tujuh puluh delapan digit desimal, beberapa urutan besarnya lebih besar dari perkiraan jumlah atom di alam semesta yang teramati. Setiap teks di dunia — setiap buku, setiap email, setiap pesan — jatuh pada salah satu nilai tersebut. Probabilitas dua teks berbeda bertepatan secara kebetulan, untuk tujuan praktis, tidak dapat dibedakan dari nol.

Bagaimana tampilannya dalam kode

Dalam Zig, bahasa yang kami gunakan untuk menulis bagian-bagian yang mendukung Solo2, menghitung segel SHA-256 dari sebuah teks terlihat 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 saja meminta perpustakaan standar Zig untuk menghitung SHA-256 dari teks di dalam tanda kutip. Setelah pemanggilan, variabel *resumen* berisi tiga puluh dua byte yang membentuk segel dalam bentuk mentahnya; saat ditampilkan di layar dalam heksadesimal, itu adalah enam puluh empat karakter yang muncul di bawah artikel ini. Jika kita mengubah *Cuadernos Lacre* menjadi *Cuadernos lacre* — satu huruf kapital dikurangi — segelnya akan berubah sepenuhnya. Itulah, dalam lima baris, sifat sentral yang menopang sisanya. Bagi yang ingin melihat cara kerjanya secara internal, di akhir artikel kami menyertakan versi algoritme yang dapat dibaca dengan komentar langkah demi langkah.

Mengapa kami menyebutnya segel lak

Dalam korespondensi Eropa abad ke-15 hingga ke-19, lak (lilin segel) menutup surat. Setetes lilin cair, segel ditekan di atasnya, dan surat itu ditandai dengan cara yang tidak dapat diulang. Itu tidak melindungi isi dari orang yang berniat mengintip — kertas bisa dibaca di bawah cahaya, lak bisa dipatahkan — tetapi itu membuktikannya. Perubahan apa pun pada penutup akan terlihat oleh penerima bahkan sebelum membuka kertas. Lak tidak mencegah kerusakan; ia menyatakannya.

SHA-256 dari isi setiap Cuaderno melakukan fungsi yang sama dalam versi digitalnya. Jika satu kata saja dari artikel berubah antara saat diterbitkan dan saat Anda membacanya, segel heksadesimal di bawah teks tidak akan lagi cocok dengan SHA-256 dari teks yang ada di depan Anda. Pembaca mana pun dengan lima baris kode dapat memeriksanya. Publikasi tidak dapat menulis ulang sejarahnya tanpa segel yang membocorkannya. Ia tidak melindungi dari kerusakan; ia membuatnya dapat diverifikasi.

Apa yang bukan kegunaan hash

Empat kegunaan terkadang diminta dari SHA-256 yang bukan merupakan haknya:

1. **Enkripsi.** Sebuah hash meringkas; tidak menyembunyikan. Jika Anda ingin teks tidak dapat dibaca, Anda perlu mengenkripsinya, bukan menghashnya.
2. **Autentikasi penulis.** Sebuah hash tidak mengatakan siapa yang menulis teks, hanya teks apa yang dihash. Untuk mengaitkan kepengarangan, diperlukan tanda tangan kriptografi di atas hash, bukan hash saja.
3. **Menyimpan kata sandi.** Di sini ada jebakan yang perlu dipahami. SHA-256 dirancang untuk menjadi sangat cepat — yang bagus untuk banyak hal, tetapi buruk untuk ini. Penyerang dengan perangkat keras khusus dapat mencoba miliaran kata sandi per detik terhadap hash SHA-256 hingga menemukan milik Anda. Untuk menyimpan kata sandi, harus digunakan fungsi derivasi kunci yang sengaja dibuat lambat seperti Argon2, scrypt, atau bcrypt, dikombinasikan dengan *sal* (salt - data acak unik per pengguna, yang mencegah dua orang dengan kata sandi yang sama memiliki hash yang sama).
4. **Membaca hash sebagai identitas penulis.** Itu bukan identitas. Sebuah hash mengidentifikasi konten. Jika dua orang menghash kata *halo* dengan SHA-256, keduanya mendapatkan ringkasan yang sama — dan itu adalah sifat sentral, bukan cacat: jika ringkasannya berbeda, kita tidak dapat memeriksa kecocokan antara yang diterbitkan dan yang diterima.

Di mana SHA-256 muncul dalam kehidupan sehari-hari Anda

Meskipun Anda tidak melihatnya, SHA-256 menopang sebagian besar dari apa yang Anda gunakan setiap hari di internet. Rantai blok Bitcoin dibangun dengan merantai SHA-256 dari setiap blok ke blok berikutnya; mengubah blok masa lalu memaksa penghitungan ulang seluruh rantai berikutnya. Git, sistem yang digunakan untuk versi kode di seluruh dunia, mengidentifikasi setiap commit dengan SHA-256 (di versi terbaru) atau pendahulunya SHA-1 (di versi lama) dari seluruh isinya. Sertifikat HTTPS yang memverifikasi identitas situs web saat Anda masuk membawa jejak SHA-256 terkait. Unduhan perangkat lunak sering kali disertai dengan SHA-256 yang diterbitkan oleh pengembang agar Anda dapat memverifikasi bahwa file tersebut tidak diubah di tengah jalan. Dan, seperti yang telah kami katakan, di bawah setiap Cuaderno Lacre.

Bagi pembaca profesional

Empat pengingat operasional bagi siapa pun yang memutuskan atau mengaudit sistem:

1. Hash bukan enkripsi. Jika vendor mencampuradukkan kedua istilah tersebut dalam dokumentasi teknisnya, ada baiknya menanyakan apa maksud sebenarnya.
2. Untuk menyimpan kata sandi jangan pernah menggunakan SHA-256 saja. SHA-256 terlalu cepat untuk tugas ini (lihat poin 3 dari *Apa yang bukan kegunaan hash*). Standar saat ini adalah **Argon2id**: lambat secara desain, dapat dikonfigurasi sesuai kapasitas server, dikombinasikan dengan *sal* (salt) acak yang berbeda per pengguna.
3. Untuk integritas dokumen — kontrak, berkas, file — SHA-256 tetap menjadi standar referensi. Inilah yang digunakan oleh penyedia segel waktu (timestamp) berkualitas di UE.
4. Untuk pelestarian jangka panjang (dekade), ada baiknya menghitung dan mengarsipkan juga SHA-3 atau SHA-512 bersama SHA-256; kehati-hatian kriptografi merekomendasikan untuk tidak mengandalkan satu fungsi saja untuk arsip berusia seabad.

Secara teknis, struktur berulang (iterasi) ini — di mana status perantara dipertahankan di antara blok input — dikenal sebagai konstruksi **Merkle-Damgård**, pola yang mendasari SHA-1, SHA-2 (termasuk SHA-256), dan banyak fungsi hash klasik lainnya. Sebaliknya, SHA-3 meninggalkan Merkle-Damgård demi arsitektur berbeda yang disebut *spons (sponge)*.

Cara kerja SHA-256, langkah demi langkah, dalam bahasa awam

Bayangkan Anda telah menyusun sirkuit domino paling rumit di dunia: ribuan balok (fichas), puluhan percabangan, jembatan mekanis, dan tanjakan yang melintasi seluruh ruangan, disusun dengan hati-hati satu per satu.

Jika Anda menyentuh balok pertama, rangkaian tersebut akan jatuh dalam urutan yang tepat dan dapat diulang. Susunan yang sama, sentuhan awal yang sama → pola akhir balok yang jatuh yang identik, berulang kali.

Di sinilah letak kemenarikannya: geser **satu balok saja** setengah sentimeter ke samping sebelum memulai dan sentuh lagi. Tanjakan yang seharusnya aktif tetap diam, jembatan tidak jatuh, percabangan yang berbeda terpicu. Pola akhir balok di lantai benar-benar tidak dapat dikenali dibandingkan dengan yang pertama.

Secara matematis, SHA-256 adalah sirkuit ini. Teks yang Anda tulis adalah posisi awal balok-balok tersebut. Algoritme adalah sentuhan yang melepaskan air terjun (cascade). Dan hasil akhirnya — yang kita sebut *hash* — adalah foto statis lantai saat semuanya telah berhenti. Ubah satu koma saja dari teks asli dan fotonya akan sangat berbeda. Sesederhana itu, dan sedrastis itu.

Langkah 1. Menerjemahkan teks ke dalam balok biner. Komputer tidak memahami huruf; mereka menerjemahkannya terlebih dahulu ke angka (ASCII) dan angka ke biner (satu dan nol). Setiap huruf diubah menjadi 8 balok putih atau hitam: *A* adalah `01000001`, *B* adalah `01000010`, spasi adalah `00100000`. Seluruh teks Anda — sebuah kata, kontrak, novel — menjadi barisan panjang balok putih dan hitam.

Langkah 2. Mengisi hingga ukuran standar. Sirkuit memproses barisan dalam *bagian* yang terdiri dari tepat 512 balok. Jika pesan Anda tidak mencapai kelipatan 512, balok penanda (dengan nilai `10000000`) ditambahkan tepat setelah teks dan kemudian nol hingga bagian tersebut lengkap. 64 posisi terakhir dari setiap bagian dicadangkan untuk mencatat panjang asli teks. Dengan demikian, sirkuit selalu tahu di mana konten asli berakhir dan di mana pengisian (padding) dimulai.

Langkah 3. Menempatkan delapan balok utama. Sebelum memulai, kita letakkan **delapan balok utama** di atas meja dalam posisi awal yang tepat. Kedelapan balok ini bukanlah rahasia: nilai awalnya ditetapkan oleh aturan matematika publik (akar kuadrat dari delapan bilangan prima pertama — 2, 3, 5, 7, 11, 13, 17, 19 — dan bit pertama dari bagian desimal setiap akar). Semua orang, di mana pun di planet ini, mulai dengan delapan balok utama yang sama di posisi yang sama. Nasib mereka adalah didorong dan diubah oleh air terjun (avalancha).

Langkah 4. Air terjun besar: enam puluh empat putaran dorongan. Di sinilah pertunjukan dimulai. Bagian pertama yang terdiri dari 512 balok teks Anda dibenturkan ke delapan balok utama. Tetapi mereka tidak jatuh sekaligus: mekanisme tersebut menjalankan **enam puluh empat putaran berturut-turut**. Di setiap putaran, dilakukan tiga operasi dengan balok-balok tersebut:

- **Komidi Putar (Tiovivo)** (rotasi). Balok-balok bergerak melingkar: yang di kanan pindah ke kiri. Tidak ada balok yang hilang atau bertambah; mereka hanya disusun ulang dengan melakukan putaran penuh pada komidi putar. Ini adalah cara yang murah dan dapat dibalik untuk mendistribusikan ulang informasi.
- **Corong Logika (Embudo Lógico)** (XOR). Balok-balok melewati corong yang membandingkannya berpasangan: jika keduanya berwarna sama, yang keluar adalah balok putih; jika berbeda, yang keluar adalah balok hitam. Ini adalah operasi logika biner yang paling sederhana, tetapi jika dikombinasikan dengan rotasi komidi putar, ini menjadi sangat kuat untuk mencampur informasi tanpa menghilangkannya.
- **Meluap (Desborde)** (penambahan modular). Hasilnya ditambahkan dengan *balok dorong konstan* yang diambil dari daftar publik berisi enam puluh empat konstanta (akar pangkat tiga dari enam puluh empat bilangan prima pertama). Jika penambahan tersebut menghasilkan balok ekstra yang tidak muat dalam ruang 32 balok yang disediakan, balok berlebih tersebut akan dibuang. Meja hanya memiliki ruang untuk 32 balok, tidak lebih.

Pada akhir putaran ke-enam puluh empat, masing-masing balok dari bagian teks Anda telah memengaruhi posisi delapan balok utama. Energi dorongan telah menjalar ke seluruh sirkuit.

Langkah 5. Menambahkan bagian berikutnya (tanpa memulai ulang). Jika teks Anda panjang dan masih ada bagian 512 balok lainnya yang harus diproses, **sirkuit tidak dimulai ulang**. Kedelapan balok utama tetap berada dalam posisi yang ditinggalkan oleh air terjun (avalancha) pertama, dan bagian kedua diluncurkan ke arah mereka untuk memicu enam puluh empat putaran lainnya. Ini seperti menambahkan ruangan baru yang penuh domino di akhir ruangan yang baru saja jatuh: kekacauan ruangan pertama sepenuhnya menentukan bagaimana ruangan kedua akan jatuh.

Langkah 6. Mengambil foto akhir. Ketika tidak ada lagi bagian yang harus diproses, air terjun (avalancha) berhenti. Kita lihat posisi akhir dari kedelapan balok utama. Kita terjemahkan konfigurasinya ke dalam kode huruf dan angka dalam sistem heksadesimal. Hasilnya adalah rangkaian tepat enam puluh empat karakter: itulah segel (sello) SHA-256 Anda.

Empat properti muncul dengan sendirinya dari cara sirkuit tersebut dipasang:

1. **Determinisme.** Teks yang sama selalu menghasilkan foto akhir yang sama, di komputer mana pun di dunia. Nol keacakan, nol kejutan.
2. **Efek Air Terjun (Efecto avalancha).** Satu koma yang ditambahkan, satu huruf kapital yang diubah, satu tanda baca yang terlupakan: fotonya menjadi sama sekali tidak dapat dikenali. Inilah sensitivitas ekstrem yang telah kita jelaskan di awal.
3. **Satu arah.** Berdasarkan foto akhir, Anda tidak dapat merekonstruksi teks asli. Rotasi, corong, dan luapan menghancurkan semua informasi arah tentang *dari mana setiap bit berasal* dan hanya mempertahankan *apa yang ditambahkan secara total*.
4. **Resistensi tabrakan.** Dalam dua puluh lima tahun kriptanalisis publik, tidak ada yang berhasil menemukan dua teks berbeda yang foto akhirnya cocok. Dan kesulitan untuk melakukannya berada di luar jangkauan komputasi peradaban mana pun yang dapat dibayangkan secara wajar.

Lampiran kode berikut menerapkan tepat enam langkah ini di Zig. Sekarang Anda dapat membacanya dengan mengetahui arti dari setiap operasi bit, alih-alih menerima manipulasi tersebut secara buta.

Glosarium teknis

Bagi pembaca yang ingin memahami apa yang dilakukan setiap operasi. Silakan lewati: artikel tetap dapat dipahami tanpanya.

ASCII dan Unicode — bagaimana huruf menjadi angka. Komputer tidak melihat huruf; mereka melihat angka. Sebuah standar yang disebut **ASCII** (*American Standard Code for Information Interchange*, tahun 1963) menetapkan angka tertentu untuk setiap karakter keyboard: *A* adalah 65, *B* adalah 66, *a* adalah 97, *0* adalah 48, spasi adalah 32, koma adalah 44. Sistem modern memperluasnya dengan **Unicode**, yang menetapkan angka untuk setiap karakter dari setiap alfabet di dunia: Sirilik, Arab, Mandarin, Jepang, dan bahkan emoji. Saat Anda mengetik karakter atau membuka file teks, komputer membaca angka di baliknya, bukan bentuk di layar. SHA-256 bekerja pada angka-angka ini, memperlakukan teks apa pun sebagai urutan angka yang panjang. Itulah sebabnya ia dapat menyegel artikel dalam bahasa Spanyol, puisi dalam bahasa Jepang, dan file biner dengan algoritme yang sama.
XOR — pembanding bit demi bit. XOR (diucapkan *«eksor»*, dari bahasa Inggris *exclusive or*, «atau eksklusif») adalah salah satu operasi paling sederhana yang dapat dilakukan komputer dengan dua angka biner. Ia membandingkan dua bit posisi demi posisi dan mengembalikan: **1** jika tepat salah satu dari keduanya adalah 1 (satu tetapi tidak keduanya), **0** jika keduanya sama (keduanya 0 atau keduanya 1). Contoh: XOR dari `1010` dan `1100` adalah `0110`. Ia memiliki properti yang luar biasa: ia dapat dibalik (reversible) — jika Anda melakukan XOR dua kali dengan kunci yang sama, Anda kembali ke aslinya. Itulah mengapa ini menjadi andalan (caballo de batalla) kriptografi: ia mencampur bit tanpa kehilangan informasi, tetapi hasilnya tidak mengungkapkan apa pun tentang input jika Anda tidak mengetahui salah satunya.
Heksadesimal — berhitung dalam basis 16. Hampir semua angka sehari-hari menggunakan sepuluh digit (0-9). Heksadesimal menggunakan enam belas: 0-9 yang biasa ditambah enam huruf yang mewakili nilai-nilai berikut: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Mengapa enam belas? Karena komputer berpikir dalam kelompok empat bit, dan empat bit dapat mewakili tepat enam belas nilai yang berbeda — dengan demikian, satu karakter heksadesimal berhubungan bersih dengan empat bit. Jejak (huella) SHA-256 berukuran 256 bit, yang tepatnya adalah **64 karakter heksadesimal**. Jika kita menulisnya dalam desimal biasa, ia akan memakan sekitar 78 digit dan akan lebih tidak nyaman. Pilihannya bersifat estetis dan ringkas; angka dasarnya tetap sama.
Rotasi bit — komidi putar biner (tiovivo biner). Bayangkan barisan tujuh bola lampu, ada yang menyala (1) dan ada yang mati (0): `1 0 1 1 0 0 1`. Memutar ke kanan satu posisi berarti mengambil lampu paling kanan, memindahkannya ke ujung kiri, dan menggeser sisanya satu tempat ke kanan: `1 1 0 1 1 0 0`. Tidak ada lampu yang hilang atau bertambah: mereka hanya menari dalam lingkaran. SHA-256 menggunakan rotasi bit ratusan kali dalam setiap perhitungan; ini adalah cara yang murah dan tanpa kehilangan untuk mendistribusikan ulang informasi di dalam status tersebut.
Konstanta *«nothing-up-my-sleeve»* — mengapa mereka berasal dari bilangan prima. Delapan balok utama dan enam puluh empat konstanta putaran SHA-256 tidak dipilih secara acak. Mereka berasal dari akar kuadrat dan akar pangkat tiga dari bilangan prima pertama. Mengapa? Karena perancangnya menginginkan konstanta *«tanpa tipu muslihat»* (*«nothing-up-my-sleeve»*): nilai-nilai yang asal-usulnya dapat diverifikasi oleh siapa pun. Jika seseorang berkata kepada Anda *«percayalah pada saya: gunakan nomor acak 32-bit ini»*, Anda wajar jika mencurigai adanya kelemahan tersembunyi atau pintu belakang (backdoor). Tetapi siapa pun yang memiliki kalkulator dapat memverifikasi bahwa 32 bit pertama dari akar kuadrat dari 2 adalah `0x6a09e667`. Nilai-nilainya bersifat matematis, publik, dan dapat direproduksi: tidak ada perangkap tersembunyi yang dapat menyelinap ke dalam resep tersebut.

Lampiran: SHA-256 dalam kode yang dapat dibaca

Lampiran ini ditujukan bagi pembaca yang ingin melihat algoritme dari dalam. Ini adalah implementasi didaktik dalam Zig yang mengikuti spesifikasi FIPS 180-4. Ini bukan versi yang digunakan Solo2 — yang asli ada di `std.crypto.hash.sha2.Sha256` dari perpustakaan standar Zig, dioptimalkan dan diaudit. Tetapi algoritmenya sama: apa yang Anda lihat di sini adalah, langkah demi langkah, apa yang terjadi ketika pemanggilan lima karakter tersebut menjalankan 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
}
```

Penulisan ulang apa pun dalam bahasa lain yang mengikuti struktur yang sama — konstanta awal, ekspansi schedule, enam puluh empat putaran, akumulasi — menghasilkan hasil yang sama. Algoritme ini tidak memiliki rahasia: nilainya terletak pada kenyataan bahwa sifat-sifat yang disebutkan di atas terus bertahan setelah dua dekade kriptanalisis publik atas ribuan mata.

Jika Anda kembali ke bagian bawah artikel ini, Anda akan melihat segel heksadesimal enam puluh empat karakter. Ini adalah SHA-256 dari teks yang baru saja Anda baca, dalam bahasa ini. Jika kita menerjemahkan artikel tersebut, segelnya akan berbeda; jika satu kata dari versi bahasa Indonesia berubah, segel bahasa Indonesia akan berubah. Segel tersebut tidak melindungi konten — untuk itu ada alat lain — melainkan mengidentifikasinya secara unik. Dan itu, meskipun terdengar sederhana, cukup agar tidak ada langkah dalam rantai editorial yang dapat mengubah apa yang dikatakan tanpa ketahuan. Hal lainnya — mengenkripsi, menandatangani, mengidentifikasi — dibangun di atas ide sederhana ini.

Sumber dan bacaan lebih lanjut

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, Agustus 2015. Spesifikasi resmi dari keluarga SHA-2, termasuk SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, Mei 2011. Versi normatif bagi pengembang.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Bab 5 dan 6 membahas fungsi hash serta kegunaan yang sah dan tidak sah.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Contoh praktis penggunaan SHA-256 untuk merantai blok dalam struktur yang tidak dapat diubah secara konstruksi.
- Regulasi (UE) 910/2014 (eIDAS) — kerangka kerja penyedia segel waktu berkualitas. SHA-256 adalah fungsi referensi untuk tanda tangan dan segel elektronik berkualitas yang diterbitkan di UE.
- Implementasi referensi dalam Zig: `std.crypto.hash.sha2.Sha256` di repositori resmi bahasa tersebut (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Ini adalah versi optimal dan diaudit yang sebenarnya digunakan Solo2. Berguna untuk membandingkan dengan implementasi didaktik pada lampiran.

← SebelumnyaCUADERNOS_LIST_SCHREMS_TITLEBerikutnya →CUADERNOS_LIST_KILLSWITCH_TITLE

Bacaan terbaru

- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE

Bawa artikel ini bersama Anda ke mana pun Anda membutuhkannya.

↓ Markdown ↓ Teks murni ↓ PDF

File akan diunduh ke perangkat Anda. Dari sana Anda dapat menyimpannya, mengimpornya ke Solo2, atau membagikannya di mana pun Anda mau. Cuadernos tidak memutuskan tujuan untuk Anda.

Segel lilin · SHA-256 b79bd0a7ca399ab44082c1daf8f134fc568425343b5fc4755bdd474570433b43

Cuadernos Lacre · Publikasi dari Menzuri Gestión S.L. ·
ditulis oleh R.Eugenio · disunting oleh tim Solo2.

diperlukan untuk preferensi tema terang/gelap Anda. Tanpa pelacak, tanpa pemrofilan, tanpa berbagi data. Jika Anda ingin mengikuti kami: RSS.