SHA-256 thực sự là gì

Một dấu vân tay toán học gói gọn trong sáu mươi bốn ký tự và sẽ thay đổi hoàn toàn nếu chỉ một dấu phẩy trong văn bản gốc bị dịch chuyển. Tại sao chúng tôi gọi nó là con dấu sáp niêm phong kỹ thuật số.

Ý tưởng đơn giản đằng sau cái tên kỹ thuật

Hãy tưởng tượng có một chiếc máy chỉ có một khe cắm và một màn hình. Bạn đưa một văn bản vào khe cắm: một từ, một câu, hoặc cả một cuốn tiểu thuyết. Trên màn hình xuất hiện, chỉ tích tắc sau đó, một chuỗi ký tự chính xác gồm sáu mươi bốn ký tự. Chuỗi đó, đối với người đọc chuyên nghiệp, chúng tôi gọi là *hash* hoặc *tóm tắt mật mã*; còn với người đọc phổ thông, hiện tại chúng ta có thể gọi nó là dấu vân tay toán học của văn bản, giống như dấu vân tay đối với một con người.

Nếu bạn đưa cùng một văn bản hai lần, máy sẽ hiển thị cùng một dấu vân tay cả hai lần. Nếu bạn đưa một văn bản hơi khác một chút — một dấu phẩy bị dịch chuyển, một chữ hoa chuyển thành chữ thường — máy sẽ hiển thị một dấu vân tay hoàn toàn khác với dấu đầu tiên. Không phải tương tự: mà là khác biệt hoàn toàn. Hai đặc tính này đi cùng nhau — tính tất định và tính nhạy cảm — chính là ý tưởng đơn giản đó. Mọi thứ khác của SHA-256 là bộ máy giúp thực thi chúng một cách hiệu quả.

Cần phải nói ngay từ đầu những gì chiếc máy không làm. Nó không mã hóa văn bản. Nó không che giấu nó. Nó không lưu trữ nó. Chiếc máy nhìn vào văn bản, tính toán dấu vân tay, và quên luôn văn bản đó. Dấu vân tay không cho phép khôi phục lại văn bản đã tạo ra nó; nó chỉ cho phép, với một văn bản ứng viên, kiểm tra xem có khớp với bản gốc hay không. Đó là lý do tại sao chúng tôi nói rằng đó là một bản tóm tắt *một chiều*: đi mà không có đường về.

Hash không giống với mã hóa

Sự nhầm lẫn này rất phổ biến và cần được làm rõ: mã hóa (cifrar) và băm (hashear) là những thao tác khác nhau. Mã hóa bao gồm việc biến đổi một văn bản sao cho chỉ người có khóa mới có thể đưa nó về dạng ban đầu. Băm bao gồm việc tạo ra một dấu vân tay của văn bản mà từ đó văn bản gốc không bao giờ có thể phục hồi được, dù có khóa hay không. Thao tác thứ nhất có thể đảo ngược theo thiết kế; thao tác thứ hai là không thể đảo ngược theo thiết kế.

Hệ quả thực tế rất quan trọng. Khi một ứng dụng nói «chúng tôi lưu trữ mật khẩu của bạn dưới dạng mã hóa», có ai đó có khóa để giải mã nó — chính ứng dụng đó, trong bất kỳ trường hợp nào. Khi một ứng dụng nói «chúng tôi lưu trữ mật khẩu của bạn dưới dạng hash», chính ứng dụng đó cũng không thể đọc được mật khẩu gốc ngay cả khi họ muốn; họ chỉ có thể kiểm tra xem mật khẩu bạn nhập có tạo ra cùng một dấu vân tay hay không. Mô hình thứ hai, nếu được thực hiện đúng cách, sẽ ưu việt hơn nhiều so với mô hình thứ nhất để lưu trữ mật khẩu. Sau này chúng ta sẽ thấy tại sao «thực hiện đúng cách» đòi hỏi nhiều hơn là chỉ dùng SHA-256 thuần túy.

Bốn đặc tính làm nên sự hữu ích của một hàm băm mật mã

Một hàm băm xứng đáng với tính từ *mật mã* phải đáp ứng bốn đặc tính sau:

1. **Tính tất định.** Cùng một đầu vào luôn tạo ra cùng một dấu vân tay.
2. **Hiệu ứng thác đổ (Avalanche effect).** Một thay đổi nhỏ ở đầu vào sẽ tạo ra một dấu vân tay hoàn toàn khác, không có điểm tương đồng rõ ràng với dấu trước đó.
3. **Kháng nghịch ảnh.** Với một dấu vân tay cho trước, việc tìm ra văn bản đã tạo ra nó là không khả thi về mặt tính toán.
4. **Kháng va chạm.** Việc tìm ra hai văn bản khác nhau tạo ra cùng một dấu vân tay là không khả thi về mặt tính toán.

«Không khả thi về mặt tính toán» không có nghĩa là «về mặt toán học là không thể». Nó có nghĩa là chi phí về thời gian, năng lượng và tiền bạc để đạt được điều đó vượt xa tổng năng lực tính toán hiện có một cách hợp lý. Đối với SHA-256, giới hạn đó được đo bằng hàng nghìn tỷ năm ngay cả với những giả định lạc quan nhất với phần cứng chuyên dụng. Điều này, đối với mục đích thực tế của người đọc, tương đương với việc «không thể làm được».

SHA-256, một cách cụ thể

Cái tên đã nói lên tất cả. SHA là viết tắt của *Secure Hash Algorithm*: thuật toán băm an toàn. Con số 256 chỉ kích thước của dấu vân tay tính bằng bit: hai trăm năm mươi sáu bit, tức là ba mươi hai byte, khi hiển thị dưới dạng thập lục phân sẽ là sáu mươi bốn ký tự mà người đọc đã nhận ra. Tiêu chuẩn này được NIST của Hoa Kỳ công bố, cơ quan bình thường hóa các loại hàm này, vào năm 2001 như một phần của họ SHA-2; phiên bản hiện hành của tiêu chuẩn, FIPS 180-4, là từ năm 2015.

Dành cho những ai chưa nắm rõ bit và byte là gì:

```
1 bit    →  0 hoặc 1      (một công tắc: bật hoặc tắt)
1 byte   →  8 bit         (256 kết hợp có thể)
32 byte  →  256 bit       (dấu vân tay SHA-256)
```

Số 256 ở cuối tên cho biết kích thước của dấu vân tay tính bằng bit. Trong hệ thập lục phân — một hệ thống đánh số với mười sáu biểu tượng thay vì mười — 256 bit đó vừa vặn trong chính xác 64 ký tự. Đó là 64 ký tự mà bạn thấy ở cuối mỗi Cuaderno.

Các chiều không gian này đáng để chúng ta dừng lại một chút. Hai trăm năm mươi sáu bit cho phép hai mũ hai trăm năm mươi sáu giá trị khác nhau: một con số với bảy mươi tám chữ số thập phân, lớn hơn nhiều bậc so với số lượng nguyên tử ước tính trong vũ trụ có thể quan sát được. Mỗi văn bản trên thế giới — mỗi cuốn sách, mỗi email, mỗi tin nhắn — đều rơi vào một trong những giá trị đó. Xác suất để hai văn bản khác nhau trùng khớp ngẫu nhiên, trên thực tế, là không thể phân biệt được với số không.

Cách hiển thị trong mã nguồn

Trong Zig, ngôn ngữ mà chúng tôi sử dụng để viết các thành phần hỗ trợ Solo2, việc tính toán con dấu SHA-256 của một văn bản sẽ trông như thế này:

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

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

Chúng ta vừa yêu cầu thư viện chuẩn của Zig tính toán SHA-256 của văn bản trong dấu ngoặc kép. Sau lời gọi đó, biến *resumen* chứa ba mươi hai byte tạo nên con dấu ở dạng thô; khi hiển thị trên màn hình dưới dạng thập lục phân, chúng là sáu mươi bốn ký tự xuất hiện ở cuối bài viết này. Nếu chúng ta thay đổi *Cuadernos Lacre* thành *Cuadernos lacre* — bớt đi một chữ hoa — toàn bộ con dấu sẽ thay đổi. Đó chính là đặc tính trung tâm hỗ trợ cho phần còn lại, chỉ trong năm dòng mã. Đối với những ai muốn xem cách nó hoạt động bên trong, ở cuối bài viết chúng tôi có kèm theo một phiên bản thuật toán dễ hiểu với các chú thích từng bước.

Tại sao chúng tôi gọi nó là con dấu sáp niêm phong

Trong thư từ châu Âu từ thế kỷ 15 đến thế kỷ 19, sáp niêm phong (lacre) dùng để đóng kín lá thư. Một giọt sáp nóng chảy, một con dấu ấn lên trên, và lá thư được đánh dấu một cách không thể lặp lại. Nó không bảo vệ nội dung khỏi những kẻ tò mò quyết tâm — giấy có thể đọc được dưới ánh sáng, sáp có thể bị phá vỡ — nhưng nó đã chứng thực điều đó. Bất kỳ sự thay đổi nào đối với dấu niêm phong đều có thể nhìn thấy được đối với người nhận ngay cả trước khi mở giấy. Sáp niêm phong không ngăn cản hư hại; nó tuyên cáo điều đó.

SHA-256 của nội dung mỗi Cuaderno thực hiện chức năng tương tự trong phiên bản kỹ thuật số của nó. Nếu chỉ một từ trong bài viết thay đổi giữa thời điểm nó được xuất bản và thời điểm bạn đọc nó, con dấu thập lục phân ở cuối văn bản sẽ không còn khớp với SHA-256 của văn bản trước mặt bạn. Bất kỳ người đọc nào với năm dòng mã đều có thể kiểm tra điều

đó. Ấn phẩm không thể viết lại lịch sử của nó mà không bị con dấu tố cáo. Nó không bảo vệ chống lại hư hại; nó làm cho hư hại có thể xác minh được.

Những gì một hàm băm không phải là

Đôi khi người ta yêu cầu SHA-256 thực hiện bốn việc không thuộc về nó:

1. **Mã hóa.** Một hàm băm tóm tắt; nó không che giấu. Nếu bạn muốn văn bản không thể đọc được, bạn cần mã hóa nó, chứ không phải băm nó.
2. **Xác thực tác giả.** Một hàm băm không cho biết ai đã viết văn bản, nó chỉ cho biết văn bản nào đã được băm. Để liên kết quyền tác giả, cần có một chữ ký mật mã trên mã băm đó, chứ không phải chỉ mình mã băm.
3. **Lưu trữ mật khẩu.** Ở đây có một cái bẫy cần phải hiểu rõ. SHA-256 được thiết kế để hoạt động rất nhanh — điều này tốt cho nhiều việc, nhưng lại tệ cho việc này. Một kẻ tấn công với phần cứng chuyên dụng có thể thử hàng tỷ mật khẩu mỗi giây với một mã băm SHA-256 cho đến khi tìm thấy mật khẩu của bạn. Để lưu trữ mật khẩu, phải sử dụng các hàm dẫn xuất khóa chậm một cách cố ý như Argon2, scrypt hoặc bcrypt, kết hợp với một 'muối' (salt - một dữ liệu ngẫu nhiên duy nhất cho mỗi người dùng, giúp ngăn chặn việc hai người có cùng mật khẩu có cùng một mã hash).
4. **Đọc mã băm như định danh của tác giả.** Nó không phải vậy. Một mã băm định danh nội dung. Nếu hai người băm từ *hola* bằng SHA-256, cả hai đều nhận được cùng một tóm tắt — và đó là đặc tính trung tâm, không phải là một lỗi: nếu chúng là các tóm tắt khác nhau, chúng ta sẽ không thể kiểm tra sự trùng khớp giữa những gì đã xuất bản và những gì đã nhận được.

SHA-256 xuất hiện ở đâu trong đời sống hàng ngày của bạn

Mặc dù bạn không nhìn thấy, nhưng SHA-256 hỗ trợ phần lớn những gì bạn sử dụng hàng ngày trên internet. Chuỗi khối của Bitcoin được xây dựng bằng cách liên kết SHA-256 của mỗi khối với khối tiếp theo; việc thay đổi một khối trong quá khứ buộc phải tính toán lại toàn bộ chuỗi sau đó. Git, hệ thống quản lý phiên bản mã nguồn của một nửa thế giới, định danh mỗi lần commit bằng SHA-256 (trong các phiên bản gần đây) hoặc bằng tiền thân của nó là SHA-1 (trong các phiên bản cũ hơn) của toàn bộ nội dung của nó. Chứng chỉ HTTPS xác minh danh tính của một trang web khi bạn truy cập có liên kết với một dấu vân tay SHA-256. Các bản tải xuống phần mềm thường đi kèm với một SHA-256 do nhà phát triển công bố để bạn xác minh rằng tệp không bị thay đổi trên đường đi. Và như chúng tôi đã nói, ở cuối mỗi Cuaderno Lacre.

Dành cho người đọc chuyên nghiệp

Bốn lời nhắc nhở vận hành cho những người quyết định hoặc kiểm định hệ thống:

1. Hash không phải là mã hóa. Nếu một nhà cung cấp nhầm lẫn hai thuật ngữ này trong tài liệu kỹ thuật của họ, bạn nên hỏi chính xác ý của họ là gì.
2. Để lưu trữ mật khẩu, không bao giờ được sử dụng SHA-256 thuần túy. SHA-256 quá nhanh cho nhiệm vụ này (xem điểm 3 của mục *Những gì một hàm băm không phải là*). Tiêu chuẩn hiện tại là **Argon2id**: chậm theo thiết kế, có thể cấu hình theo năng lực của máy chủ, kết hợp với một 'muối' ngẫu nhiên khác nhau cho mỗi người dùng.
3. Đối với tính toàn vẹn của tài liệu — hợp đồng, hồ sơ, tệp tin — SHA-256 vẫn là tiêu chuẩn tham chiếu. Đây là tiêu chuẩn được các nhà cung cấp dịch vụ đóng dấu thời gian đủ điều kiện ở EU sử dụng.
4. Để bảo tồn lâu dài (nhiều thập kỷ), bạn nên tính toán và lưu trữ thêm một SHA-3 hoặc SHA-512 cùng với SHA-256; sự thận trọng về mật mã khuyến nghị không nên chỉ dựa vào một hàm duy nhất cho các kho lưu trữ kéo dài hàng thế kỷ.

Về mặt kỹ thuật, cấu trúc lặp đi lặp lại này — nơi trạng thái trung gian được bảo toàn giữa các khối dữ liệu đầu vào — được gọi là cấu trúc **Merkle-Damgård**, mô hình mà SHA-1, SHA-2 (bao gồm cả SHA-256) và nhiều hàm băm cổ điển khác dựa trên đó. Ngược lại, SHA-3 từ bỏ Merkle-Damgård để chuyển sang một kiến trúc khác gọi là *sponge (bọt biển)*.

SHA-256 hoạt động như thế nào, từng bước một, bằng ngôn ngữ bình dân

Hãy tưởng tượng bạn đã lắp đặt một mạch domino tinh vi nhất thế giới: hàng nghìn quân domino, hàng chục nhánh rẽ, cầu cơ khí và đường dốc chạy khắp phòng, được đặt cẩn thận từng mảnh một.

Nếu bạn chạm vào quân đầu tiên, chuỗi sẽ đổ theo một trình tự chính xác và lặp lại. Cùng cách lắp đặt, cùng cú chạm đầu tiên → cùng một mẫu quân domino đổ cuối cùng, lặp đi lặp lại.

Đây là điểm thú vị: hãy di chuyển **chỉ một quân domino** sang bên cạnh nửa cm trước khi bắt đầu và chạm lại. Một đường dốc đáng lẽ phải hoạt động sẽ đứng yên, một cây cầu không đổ, một nhánh rẽ khác được kích hoạt. Mẫu quân domino cuối cùng trên sàn sẽ hoàn toàn không thể nhận ra so với mẫu đầu tiên.

SHA-256 về mặt toán học chính là mạch điện này. Văn bản bạn viết là vị trí ban đầu của các quân domino. Thuật toán là cú chạm giải phóng chuỗi đổ. Và kết quả cuối cùng — cái mà chúng ta gọi là *hash (mã băm)* — là bức ảnh chụp cố định mặt sàn khi mọi thứ đã dừng lại. Chỉ cần thay đổi một dấu phẩy trong văn bản gốc và bức ảnh sẽ khác biệt hoàn toàn. Đơn giản vậy thôi, nhưng vô cùng mạnh mẽ.

Bước 1. Dịch văn bản thành các quân nhị phân. Máy tính không hiểu chữ cái; trước tiên chúng dịch chúng thành số (ASCII) và số thành nhị phân (số 1 và số 0). Mỗi chữ cái trở thành 8 quân trắng hoặc đen: chữ *A* là `01000001`, chữ *B* là `01000010`, khoảng trắng là `00100000`. Toàn bộ văn bản của bạn — một từ, một bản hợp đồng, một cuốn tiểu thuyết — trở thành một hàng dài các quân trắng và đen.

Bước 2. Lấp đầy cho đến kích thước tiêu chuẩn. Mạch xử lý hàng theo các *đoạn (chunks)* chính xác 512 quân. Nếu tin nhắn của bạn không đạt đến bội số của 512, một quân đánh dấu (giá trị `10000000`) sẽ được thêm vào ngay sau văn bản và sau đó là các số 0 cho đến khi hoàn thành đoạn. 64 vị trí cuối cùng của mỗi đoạn được dành riêng để ghi lại độ dài ban đầu của văn bản. Bằng cách đó, mạch luôn biết nội dung thực kết thúc ở đâu và phần lấp đầy bắt đầu từ đâu.

Bước 3. Đặt tám quân chủ đạo. Trước khi bắt đầu, chúng ta đặt lên bàn **tám quân chủ đạo** ở vị trí ban đầu chính xác. Tám quân này không phải là bí mật: giá trị ban đầu của chúng được cố định bởi một quy tắc toán học công khai (căn bậc hai của tám số nguyên tố đầu tiên — 2, 3, 5, 7, 11, 13, 17, 19 — và các bit đầu tiên của phần thập phân của mỗi căn bậc hai). Mọi người, ở bất kỳ nơi nào trên hành tinh, đều bắt đầu với cùng tám quân chủ đạo ở cùng một vị trí. Số phận của chúng là bị xô đẩy và biến đổi bởi chuỗi đổ.

Bước 4. Chuỗi đổ lớn: sáu mươi tư vòng xô đẩy. Đây là lúc màn trình diễn bắt đầu. Đoạn 512 quân đầu tiên của văn bản sẽ va vào tám quân chủ đạo. Nhưng chúng không đổ ngay lập tức: cơ chế thực hiện **sáu mươi tư vòng liên tiếp**. Trong mỗi vòng, nó thực hiện ba thao tác với các quân bài:

- **Vòng quay ngựa gỗ** (xoay vòng). Các quân bài di chuyển theo vòng tròn: các quân bên phải chuyển sang bên trái. Không có quân nào bị mất hoặc thêm vào; chúng chỉ đơn giản được sắp xếp lại bằng cách quay một vòng hoàn chỉnh quanh vòng quay. Đó là một cách rẻ tiền và có thể đảo ngược để phân phối lại thông tin.
- **Cái phễu logic** (XOR). Các quân bài đi qua một cái phễu so sánh chúng theo từng cặp: nếu cả hai cùng màu, một quân trắng sẽ ra; nếu khác màu, một quân đen sẽ ra. Đó là thao tác đơn giản nhất của logic nhị phân, nhưng kết hợp với các vòng xoay của ngựa gỗ, nó trở nên vô cùng mạnh mẽ để trộn thông tin mà không làm mất nó.
- **Tràn số** (cộng theo mô-đun). Kết quả được cộng với một *quân đẩy không đổi* được lấy từ danh sách công khai gồm sáu mươi tư hằng số (căn bậc ba của sáu mươi tư số nguyên tố đầu tiên). Nếu phép cộng tạo ra các quân bài thừa không vừa với không gian 32 quân bài dự kiến, các quân bài thừa đó sẽ bị loại bỏ. Bàn chỉ có không gian cho 32 quân bài, không hơn một quân.

Vào cuối vòng thứ sáu mươi tư, mỗi quân bài trong đoạn văn bản của bạn đã ảnh hưởng đến vị trí của tám quân chủ đạo. Năng lượng của cú đẩy đã truyền đi khắp mạch.

Bước 5. Thêm đoạn tiếp theo (không khởi động lại). Nếu văn bản của bạn dài và còn một đoạn 512 quân khác cần xử lý, **mạch không khởi động lại**. Tám quân chủ đạo vẫn giữ nguyên vị trí mà chuỗi đổ đầu tiên để lại, và đoạn thứ hai được tung vào chúng để kích hoạt sáu mươi tư vòng tiếp theo. Giống như việc thêm một căn phòng mới đầy quân domino vào cuối căn phòng vừa mới đổ: sự xáo trộn của căn phòng thứ nhất sẽ quy định hoàn toàn cách căn phòng thứ hai đổ.

Bước 6. Chụp ảnh cuối cùng. Khi không còn đoạn nào để xử lý, chuỗi đổ dừng lại. Chúng ta nhìn vào vị trí cuối cùng mà tám quân chủ đạo đã dừng lại. Chúng ta dịch cấu hình của chúng sang một mã gồm chữ cái và số trong hệ thập lục phân. Kết quả là một chuỗi gồm chính xác sáu mươi tư ký tự: đó chính là con dấu SHA-256 của bạn.

Bốn đặc tính nảy sinh từ cách lắp đặt mạch điện này:

1. **Tính xác định.** Cùng một văn bản luôn tạo ra cùng một bức ảnh cuối cùng, trên bất kỳ máy tính nào trên thế giới. Không ngẫu nhiên, không bất ngờ.

2. **Hiệu ứng domino (Avalanche effect).** Một dấu phẩy được thêm vào, một chữ hoa được thay đổi, một dấu phụ bị quên: bức ảnh kết quả sẽ hoàn toàn không thể nhận ra. Đây là sự nhạy cảm cực độ mà chúng ta đã mô tả ngay từ đầu.
3. **Một chiều.** Từ bức ảnh cuối cùng, bạn không thể khôi phục lại văn bản gốc. Các vòng xoay, phễu và tràn số phá hủy mọi thông tin định hướng về *mỗi bit đến từ đâu* và chỉ giữ lại *tổng số đã được cộng vào*.
4. **Khả năng chống va chạm (Collision resistance).** Trong hai mươi lăm năm phân tích mật mã công khai, chưa ai tìm được hai văn bản khác nhau mà có bức ảnh cuối cùng trùng khớp. Và độ khó của việc này nằm ngoài khả năng tính toán của bất kỳ nền văn minh nào có thể tưởng tượng được.

Phụ lục mã dưới đây thực hiện chính xác sáu bước này trong Zig. Bây giờ bạn có thể đọc nó và hiểu ý nghĩa của từng thao tác bit, thay vì chấp nhận các thao tác xử lý một cách mù quáng.

Thuật ngữ kỹ thuật

Dành cho người đọc muốn hiểu mỗi thao tác làm gì. Bạn có thể bỏ qua phần này: bài viết vẫn có thể hiểu được mà không cần nó.

ASCII và Unicode — cách chữ cái trở thành con số. Máy tính không thấy chữ cái; chúng thấy con số. Một tiêu chuẩn có tên **ASCII** (năm 1963) gán cho mỗi ký tự bàn phím một con số cụ thể: chữ *A* là 65, *B* là 66, *a* là 97, số *0* là 48, khoảng trắng là 32, dấu phẩy là 44. Các hệ thống hiện đại mở rộng nó bằng **Unicode**, gán một con số cho mỗi ký tự của mọi bảng chữ cái trên thế giới: chữ Cyrillic, tiếng Ả Rập, tiếng Trung, tiếng Nhật và cả biểu tượng cảm xúc (emoji). Khi bạn gõ một ký tự hoặc mở một tệp văn bản, máy tính sẽ đọc con số bên dưới, không phải hình dạng trên màn hình. SHA-256 làm việc trên những con số này, xử lý bất kỳ văn bản nào như một chuỗi số dài. Đó là lý do tại sao nó có thể băm một bài báo bằng tiếng Tây Ban Nha, một bài thơ bằng tiếng Nhật và một tệp nhị phân bằng cùng một thuật toán.
XOR — bộ so sánh từng bit. XOR (phát âm là *«exor»*, từ tiếng Anh *exclusive or*, «hoặc loại trừ») là một trong những thao tác đơn giản nhất mà máy tính có thể thực hiện với hai số nhị phân. Nó so sánh hai bit theo từng vị trí và trả về: **1** nếu chính xác một trong hai là 1 (một nhưng không phải cả hai), **0** nếu cả hai giống nhau (cùng là 0 hoặc cùng là 1). Ví dụ: XOR của `1010` và `1100` là `0110`. Nó có một đặc tính đáng chú ý: có thể đảo ngược — nếu bạn thực hiện XOR hai lần với cùng một khóa, bạn sẽ trở lại bản gốc —. Đó là lý do tại sao nó là công cụ đắc lực của mật mã học: nó trộn các bit mà không làm mất thông tin, nhưng kết quả không tiết lộ bất cứ điều gì về đầu vào nếu bạn không biết một trong số chúng.
Thập lục phân — đếm theo cơ số 16. Hầu hết tất cả các con số hàng ngày đều sử dụng mười chữ số (0-9). Hệ thập lục phân sử dụng mười sáu: từ 0-9 thông thường cộng với sáu chữ cái đại diện cho các giá trị sau: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Tại sao lại là mười sáu? Bởi vì máy tính tư duy theo nhóm bốn bit, và bốn bit có thể đại diện cho chính xác mười sáu giá trị khác nhau — do đó, một ký tự thập lục phân tương ứng gọn gàng với bốn bit —. Một mã băm SHA-256 có độ dài 256 bit, chính xác là **64 ký tự thập lục phân**. Nếu chúng ta viết nó bằng hệ thập phân thông thường, nó sẽ chiếm khoảng 78 chữ số và kém thuận tiện hơn. Lựa chọn này mang tính thẩm mỹ và gọn gàng; con số bên dưới là như nhau.
Xoay bit — vòng quay nhị phân. Hãy tưởng tượng một hàng bảy bóng đèn, một số bóng đang bật (1) và số khác đang tắt (0): `1 0 1 1 0 0 1`. Xoay sang phải một vị trí bao gồm việc lấy bóng đèn ở ngoài cùng bên phải, đưa nó sang ngoài cùng bên trái và dịch chuyển các bóng đèn khác sang phải một vị trí: `1 1 0 1 1 0 0`. Không có bóng đèn nào bị mất hoặc thêm vào: chúng chỉ đơn giản là nhảy múa theo vòng tròn. SHA-256 sử dụng tính năng xoay bit hàng trăm lần trong mỗi lần tính toán; đó là một cách rẻ tiền và không gây mất mát dữ liệu để phân phối lại thông tin trong trạng thái.
Hằng số *«nothing-up-my-sleeve»* — tại sao chúng lại bắt nguồn từ các số nguyên tố. Tám quân bài chủ đạo và sáu mươi tư hằng số vòng của SHA-256 không được chọn ngẫu nhiên. Chúng bắt nguồn từ căn bậc hai và căn bậc ba của các số nguyên tố đầu tiên. Tại sao? Bởi vì các nhà thiết kế muốn các hằng số *«không có gì giấu trong tay áo»* (nothing up my sleeve): những giá trị mà bất kỳ ai cũng có thể kiểm chứng nguồn gốc của chúng. Nếu ai đó nói với bạn rằng *«hãy tin tôi: hãy sử dụng số ngẫu nhiên 32 bit này»*, bạn sẽ có lý do chính đáng để nghi ngờ về một điểm yếu tiềm ẩn hoặc một cửa hậu (backdoor). Nhưng bất kỳ ai có máy tính bỏ túi đều có thể kiểm chứng rằng 32 bit đầu tiên của căn bậc hai của 2 là `0x6a09e667`. Các giá trị là toán học, công khai và có thể tái tạo: không có bẫy ẩn nào có thể lọt vào công thức.

Phụ lục: SHA-256 trong mã nguồn dễ hiểu

Phụ lục này dành cho những độc giả muốn xem thuật toán từ bên trong. Đây là một triển khai mang tính giáo khoa trong Zig tuân theo đặc tả FIPS 180-4. Đây không phải là phiên bản mà Solo2 sử dụng — phiên bản thực nằm trong `std.crypto.hash.sha2.Sha256` của thư viện chuẩn Zig, đã được tối ưu hóa và kiểm định. Nhưng thuật toán là giống nhau: những gì bạn thấy ở đây là, từng bước một, những gì xảy ra khi lời gọi năm ký tự đó thực hiện công việc của nó.

```
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
}
```

Bất kỳ việc viết lại nào bằng ngôn ngữ khác tuân theo cùng một cấu trúc — các hằng số ban đầu, mở rộng lịch trình, sáu mươi bốn vòng lặp, tích lũy — đều tạo ra cùng một kết quả. Thuật toán không có bí mật: giá trị của nó nằm ở chỗ các đặc

tính được liệt kê ở trên vẫn đứng vững sau hai thập kỷ phân tích mật mã công khai từ hàng ngàn cặp mắt.

Nếu bạn quay lại cuối bài viết này, bạn sẽ thấy một con dấu thập lục phân gồm sáu mươi bốn ký tự. Đó là SHA-256 của văn bản bạn vừa đọc, bằng ngôn ngữ này. Nếu chúng tôi dịch bài viết, con dấu sẽ khác; nếu một từ trong phiên bản tiếng Tây Ban Nha thay đổi, con dấu tiếng Tây Ban Nha sẽ thay đổi. Con dấu không bảo vệ nội dung — đã có các công cụ khác cho việc đó — mà nó định danh nội dung một cách duy nhất. Và điều đó, dù nghe có vẻ khiêm tốn, cũng đủ để không một bước nào trong chuỗi biên tập có thể thay đổi những gì đã nói mà không bị phát hiện. Những thứ còn lại — mã hóa, ký tên, định danh — đều được xây dựng dựa trên ý tưởng đơn giản này.

Nguồn tham khảo và đọc thêm

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, tháng 8 năm 2015. Đặc tả chính thức của họ SHA-2, bao gồm SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, tháng 5 năm 2011. Phiên bản quy chuẩn cho những người triển khai.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Chương 5 và 6 đề cập đến các hàm băm và các cách sử dụng hợp pháp và bất hợp pháp của chúng.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Ví dụ thực tế về việc sử dụng SHA-256 để liên kết các khối trong một cấu trúc bất biến theo thiết kế.
- Quy định (EU) 910/2014 (eIDAS) — khung pháp lý cho các nhà cung cấp dịch vụ đóng dấu thời gian đủ điều kiện. SHA-256 là hàm tham chiếu cho các chữ ký và dấu điện tử đủ điều kiện được cấp tại EU.
- Triển khai tham chiếu trong Zig: `std.crypto.hash.sha2.Sha256` trong kho lưu trữ chính thức của ngôn ngữ (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Đây là phiên bản tối ưu hóa và kiểm định mà thực tế Solo2 đang sử dụng. Hữu ích để đối chiếu với triển khai giáo khoa trong phụ lục.

← TrướcCUADERNOS_LIST_SCHREMS_TITLETiếp theo →CUADERNOS_LIST_KILLSWITCH_TITLE

Các bài đọc gần đây

- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE

Tải bài viết này để sử dụng ở bất cứ đâu bạn cần.

↓ Markdown ↓ Văn bản thuần túy ↓ PDF

Tệp sẽ được tải xuống thiết bị của bạn. Từ đó, bạn có thể lưu trữ, nhập vào Solo2 hoặc chia sẻ ở bất cứ đâu. Cuadernos không quyết định điểm đến thay cho bạn.

Dấu sáp · SHA-256 fbef0db250f523047870f00aa5a0b89b71f82ecae15347d2d7899ce3caa29234

Cuadernos Lacre · Một ấn phẩm của Menzuri Gestión S.L. ·
viết bởi R.Eugenio · được biên tập bởi đội ngũ Solo2.

Trang web này không sử dụng cookie và không tải tài nguyên từ bên thứ ba. Chúng tôi sử dụng bộ đếm lượt truy cập ẩn danh tự lưu trữ (Umami, trên máy chủ Châu Âu của chúng tôi) và lượng JavaScript tối thiểu cần thiết cho lựa chọn giao diện sáng/tối của bạn. Không trình theo dõi, không hồ sơ hóa, không chia sẻ dữ liệu. Nếu bạn muốn theo dõi chúng tôi: RSS.