Концепція · 19 травня 2026

← Попередній·Наступний →

Що таке насправді SHA-256

Математичний відбиток, що вміщується у шістдесят чотири символи і повністю змінюється, якщо змінити хоча б одну кому в оригінальному тексті. Чому ми називаємо його цифровою сургучною печаткою.

Проста ідея за технічною назвою

Уявіть, що існує машина з одним слотом і одним екраном. У слот ви вставляєте текст: слово, фразу, цілий роман. На екрані за мить з'являється послідовність рівно із шістдесяти чотирьох символів. Ми, професійні читачі, називаємо це *хешем* або *криптографічним резюме*; для загального читача ми можемо поки що називати це математичним відбитком тексту, як відбиток пальця є відбитком людини.

Якщо ви вставите один і той самий текст двічі, машина обидва рази покаже той самий відбиток. Якщо ви вставите трохи інший текст — одну зміщену кому, велику літеру замість малої — машина покаже зовсім інший відбиток, ніж перший. Не схожий, а саме інший. Ці дві властивості разом — детермінізм і чутливість — і є тією простою ідеєю. Усе інше в SHA-256 — це механізм, який забезпечує їх виконання.

Варто відразу сказати, чого машина не робить. Вона не шифрує текст. Не приховує його. Не зберігає його. Машина дивиться на текст, обчислює відбиток і забуває про текст. Відбиток не дозволяє відновити текст, який його створив; він лише дозволяє, маючи кандидат на текст, перевірити, чи збігається він з оригіналом. Тому ми кажемо, що це резюме *в один бік*: туди можна, назад — ні.

Хеш — це не те саме, що шифрування

Плутанина виникає часто, і її варто розвіяти: шифрування та хешування — це різні операції. Шифрування полягає в перетворенні тексту так, щоб лише власник ключа міг повернути його до початкового вигляду. Хешування полягає у створенні відбитка тексту, з якого оригінальний текст ніколи не можна відновити, ні з ключем, ні без нього. Перше за задумом є оборотним; друге за задумом — необоротним.

Практичний наслідок важливий. Коли додаток каже: «ми зберігаємо ваш пароль зашифрованим», є хтось, хто має ключ для його розшифрування — у будь-якому разі сам додаток. Коли додаток каже: «ми зберігаємо ваш пароль хешованим», сам додаток не може прочитати оригінальний пароль, навіть якби захотів; він може лише перевірити, чи те, що ви вводите, знову створює той самий відбиток. Друга модель, за правильного виконання, набагато краща за першу для зберігання паролів. Далі ми побачимо, чому «правильне виконання» вимагає дещо більшого, ніж просто SHA-256.

Чотири властивості, що роблять криптографічний хеш корисним

Хеш-функція, що заслуговує на прикметник *криптографічна*, відповідає чотирьом властивостям:

1. **Детермінізм.** Один і той самий вхід завжди створює один і той самий відбиток.
2. **Ефект лавини.** Невелика зміна у вхідних даних створює зовсім інший відбиток, без видимої подібності до попереднього.
3. **Стійкість до зворотного перетворення.** Маючи відбиток, обчислювально неможливо знайти текст, який його створив.
4. **Стійкість до колізій.** Обчислювально неможливо знайти два різних тексти, які створювали б однаковий відбиток.

«Обчислювально неможливо» не означає «математично неможливо». Це означає, що вартість часу, енергії та грошей для досягнення цього на багато порядків перевищує суму всієї обчислювальної потужності, яка є розумно доступною. Для SHA-256 ця межа вимірюється тисячами трильйонів років навіть за найбільш оптимістичних підходів зі спеціалізованим обладнанням. Що для практичних цілей читача те саме, що й «неможливо».

SHA-256, конкретно

Назва говорить сама за себе. SHA — це абревіатура від *Secure Hash Algorithm*: алгоритм безпечного хешування. Число 256 вказує на розмір відбитка в бітах: двісті п'ятдесят шість бітів, тобто тридцять два байти, які у шістнадцятковому форматі виглядають як шістдесят чотири символи, які читач уже впізнає. Стандарт був опублікований американським інститутом NIST, органом, який нормує такі функції, у 2001 році як частина сімейства SHA-2; чинна версія стандарту, FIPS 180-4, датується 2015 роком.

Для тих, хто ще не знає, що таке біти та байти:

```
1 біт    →  0 або 1      (вимикач: увімкнено або вимкнено)
1 байт   →  8 бітів      (256 можливих комбінацій)
32 байти →  256 бітів     (відбиток SHA-256)
```

Число 256 у кінці назви вказує на розмір відбитка в бітах. У шістнадцятковій системі — системі числення з шістнадцятьма символами замість десяти — ці 256 бітів вміщуються рівно у 64 символи. Це ті 64 символи, які ви бачите внизу кожного Cuaderno.

Розміри заслуговують на хвилинку уваги. Двісті п'ятдесят шість бітів дозволяють два у двісті п'ятдесятому шостому ступені різних значень: число з сімдесятьма вісьмома десятковими цифрами, що на кілька порядків більше, ніж орієнтовна кількість атомів у спостережуваному всесвіті. Кожен текст у світі — кожна книга, кожен електронний лист, кожне повідомлення — потрапляє на одне з цих значень. Ймовірність того, що два різні тексти випадково збіжаться, для практичних цілей не відрізняється від нуля.

Як це виглядає в коді

У Zig, мові, якою we пишемо компоненти, що тримають Solo2, обчислення сургучної печатки SHA-256 для тексту виглядає так:

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

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

Ми щойно попросили стандартну бібліотеку Zig обчислити SHA-256 тексту в лапках. Після виклику змінна *resumen* містить тридцять два байти, що складають печатку в її сирому вигляді; коли вони відображаються на екрані у шістнадцятковому форматі, це шістдесят чотири символи, які з'являються внизу цієї статті. Якби ми змінили *Cuadernos Lacre* на *Cuadernos lacre* — на одну велику літеру менше — печатка змінилася б повністю. Це і є, у п'яти рядках, центральна властивість, на якій тримається все інше. Для тих, хто хоче побачити, як це працює всередині, наприкінці статті ми додали зрозумілу версію алгоритму з покроковими коментарями.

Чому мы називаємо його сургучною печаткою

У європейській кореспонденції XV–XIX століть сургуч (lacre) запечатував лист. Крапля розплавленого воску, притиснута зверху печатка — і лист залишався позначеним неповторним чином. Це не захищало вміст від рішучого допитливого ока — папір можна було прочитати на світло, сургуч можна було зламати — але це свідчило про втручання. Будь-яка зміна запечатування була помітна одержувачу ще до того, як він розгорнув папір. Сургуч не запобігав пошкодженню; він його виявляв.

SHA-256 тексту кожного Cuaderno виконує ту саму функцію в його цифровій версії. Якби хоча б одне слово статті змінилося між моментом її публікації та моментом, коли ви її читаєте, шістнадцяткова печатка внизу тексту вже не збігалася б із SHA-256 тексту, який перед вами. Будь-який читач із п'ятьма рядками коду міг би це перевірити.

Публікація не може переписати свою історію так, щоб печатка її не викрила. Вона не захищає від пошкодження; вона робить його таким, що піддається перевірці.

Чим хеш не є

Чотири речі іноді очікують від SHA-256, які йому не притаманні:

1. **Шифрування.** Хеш резюмує, а не приховує. Якщо ви хочете, щоб текст не можна було прочитати, вам потрібно його зашифрувати, а не хешувати.
2. **Автентифікація автора.** Хеш не говорить, хто написав текст, лише який текст був хешований. Щоб пов'язати авторство, потрібен криптографічний підпис поверх хешу, а не просто сам хеш.
3. **Зберігання паролів.** Тут є пастка, яку варто розуміти. SHA-256 розроблений бути дуже швидким — що добре для багатьох речей, але погано для цієї. Зловмисник зі спеціалізованим обладнанням може перевіряти мільярди паролів на секунду проти хешу SHA-256, поки не знайде ваш. Для зберігання паролів слід використовувати навмисно повільні функції виведення ключа, такі як Argon2, scrypt або bcrypt, у поєднанні з *сіллю* (унікальними випадковими даними для кожного користувача, що запобігає ситуації, коли двоє людей з однаковим паролем мають однаковий хеш).
4. **Читання хешу як ідентифікатора автора.** Це не так. Хеш ідентифікує вміст. Якщо двоє людей хешують слово *привіт* за допомогою SHA-256, обидва отримають однакове резюме — і це центральна властивість, а не дефект: якби резюме були різними, ми не могли б перевірити збіг між опублікованим і отриманим.

Де з'являється SHA-256 у вашому повсякденному житті

Хоча ви цього не бачите, SHA-256 підтримує значну частину того, що ви щодня використовуєте в інтернеті. Блокчейн Bitcoin будується шляхом зчеплення SHA-256 кожного блоку з наступним; зміна минулого блоку змушує перераховувати весь наступний ланцюг. Git, система, за допомогою якої версіонується код половини світу, ідентифікує кожне підтвердження (commit) за допомогою SHA-256 (у нових версіях) або його попередника SHA-1 (у старіших версіях) усього його вмісту. Сертифікати HTTPS, які перевіряють автентичність сайту, коли ви на нього заходите, мають пов'язаний відбиток SHA-256. Завантаження програмного забезпечення часто супроводжуються SHA-256, опублікованим розробником, щоб ви могли перевірити, чи не було файл змінено під час передачі. І, як ми вже казали, внизу кожного Cuadernos Lacre.

Для професійного читача

Чотири оперативні нагадування для тих, хто приймає рішення або проводить аудит систем:

1. Хеш — це не шифрування. Якщо постачальник плутає ці два терміни у своїй технічній документації, варто запитати, що саме він має на увазі.
2. Для зберігання паролів ніколи не слід використовувати лише SHA-256. SHA-256 занадто швидкий для цього завдання (див. пункт 3 розділу *Чим хеш не є*). Поточним стандартом є **Argon2id**: повільний за задумом, з можливістю налаштування відповідно до потужності сервера, у поєднанні з різною випадковою *сіллю* для кожного користувача.
3. Для цілісності документів — контрактів, справ, файлів — SHA-256 залишається еталонним стандартом. Саме його використовують кваліфіковані постачальники послуг довіри для позначення часу в ЄС.
4. Для довгострокового зберігання (десятиліття) варто також обчислювати та архівувати SHA-3 або SHA-512 разом із SHA-256; криптографічна обачність рекомендує не покладатися на одну функцію протягом столітніх архівів.

Технічно ця ітераційна структура, де проміжний стан зберігається між вхідними блоками, відома як конструкція **Меркла — Дамґорда**. Це патерн, на якому базуються SHA-1, SHA-2 (включаючи SHA-256) та багато інших класичних хеш-функцій. SHA-3, навпаки, відмовляється від конструкції Меркла — Дамґорда на користь іншої архітектури, що називається *губка (sponge)*.

Як працює SHA-256, крок за кроком, простими словами

Уявіть, що ви побудували найскладнішу у світі схему з доміно: тисячі фішок, десятки розгалужень, механічні мости та рампи, що проходять через усю кімнату, ретельно розставлені фішка за фішкою.

Якщо ви штовхнете першу фішку, ланцюжок впаде у точній і повторюваній послідовності. Однакове налаштування, однаковий перший поштовх → ідентичний фінальний малюнок впалих фішок, знову і знову.

Ось що цікаво: посуньте **лише одну фішку** на пів сантиметра вбік перед початком і знову штовхніть. Рампа, яка мала активуватися, залишається нерухомою, міст не падає, спрацьовує інше розгалуження. Фінальний візерунок фішок на підлозі абсолютно невпізнанний порівняно з першим.

Математично SHA-256 — це і є така схема. Текст, який ви пишете, — це початкове положення фішок. Алгоритм — це поштовх, який запускає каскад. А кінцевий результат — те, що ми називаємо *хешем (hash)*, — це незмінне фото підлоги, коли все зупинилося. Змініть хоча б одну кому в оригінальному тексті, і фото буде кардинально іншим. Так просто і так радикально.

Крок 1. Переклад тексту у бінарні фішки. Комп'ютери не розуміють літер; вони спочатку перекладають їх у числа (ASCII), а числа — у бінарний код (одиниці та нулі). Кожна літера перетворюється на 8 білих або чорних фішок: *A* — це `01000001`, *B* — це `01000010`, пробіл — це `00100000`. Весь ваш текст — слово, контракт, цілий роман — стає довгим рядом білих і чорних фішок.

Крок 2. Заповнення до стандартного розміру. Схема обробляє ряд *фрагментами* рівно по 512 фішок. Якщо ваше повідомлення не досягає розміру, кратного 512, відразу після тексту додається маркерна фішка (значення `10000000`), а потім нулі до заповнення фрагмента. Останні 64 позиції кожного фрагмента резервуються для запису оригінальної довжини тексту. Так схема завжди знає, де закінчився реальний контент, а де почалося заповнення.

Крок 3. Розміщення восьми майстер-фішок. Перед початком ми кладемо на стіл **вісім майстер-фішок** у точно визначеному початковому положенні. Ці вісім фішок не є секретом: їхні початкові значення встановлені загальнодоступним математичним правилом (квадратні корені з перших восьми простих чисел — 2, 3, 5, 7, 11, 13, 17, 19 — і перші біти дробової частини кожного кореня). Будь-хто у будь-якому куточку планети починає з тих самих восьми майстер-фішок у тому самому положенні. Їхня доля — бути зміненими лавиною поштовхів.

Крок 4. Велика лавина: шістдесят чотири раунди поштовхів. Тут починається шоу. Перший фрагмент із 512 фішок вашого тексту врізається у вісім майстер-фішок. Але вони не падають одразу: механізм виконує **шістдесят чотири послідовні раунди**. У кожному раунді він робить три операції з фішками:

- **Карусель** (циклічний зсув). Фішки рухаються по колу: ті, що праворуч, переходять ліворуч. Жодна фішка не втрачається і не додається; вони просто перевпорядковуються, роблячи повне коло на каруселі. Це дешевий і зворотний спосіб перерозподілити інформацію.
- **Логічна воронка** (XOR). Фішки проходять через воронку, яка порівнює їх по дві: якщо вони однакового кольору, виходить біла; якщо різні — чорна. Це найпростіша операція бінарної логіки, але у поєднанні з обертанням каруселі вона стає надзвичайно потужною для змішування інформації без її втрати.
- **Переповнення** (модульне додавання). Результат додається до *фішки постійного поштовху*, взятої із загальнодоступного списку шістдесяти чотирьох констант (кубічні корені з перших шістдесяти чотирьох простих чисел). Якщо при додаванні виникають зайві фішки, які не вміщуються у відведений простір із 32 фішок, ці зайві фішки відкидаються. На столі є місце лише для 32 фішок, ні фішкою більше.

Наприкінці шістдесят четвертого раунду кожна з фішок вашого тексту вплинула на положення восьми майстер-фішок. Енергія поштовху пройшла крізь усю схему.

Крок 5. Додавання наступного фрагмента (без скидання). Якщо ваш текст був довгим і залишився ще один фрагмент із 512 фішок для обробки, **схема не перезапускається**. Вісім майстер-фішок залишаються такими, якими їх залишила перша лавина, і другий фрагмент спрямовується на них, щоб активувати наступні шістдесят чотири раунди. Це ніби добудувати нову кімнату з доміно в кінці тієї, що щойно впала: безлад у першій повністю визначає, як впаде друга.

Крок 6. Фінальне фото. Коли фрагментів для обробки більше не залишається, лавина зупиняється. Ми дивимося на фінальне положення, у якому залишилися вісім майстер-фішок. Ми перекладаємо їхню конфігурацію у код із літер та цифр у шістнадцятковій системі. Результат — рядок рівно із шістдесяти чотирьох символів: це і є ваш відбиток SHA-256.

Чотири властивості випливають самі собою з того, як побудована схема:

1. **Детермінізм.** Один і той самий текст завжди дає однакове фінальне фото на будь-якому комп'ютері у світі. Нуль випадковості, нуль сюрпризів.
2. **Ефект лавини.** Додана кома, змінена велика літера, забутий апостроф: фото стає абсолютно невпізнанним. Це та сама екстремальна чутливість, яку ми описали на початку.
3. **Незворотність.** Маючи фінальне фото, ви не можете відновити оригінальний текст. Обертання, воронки та переповнення знищують усю інформацію про те, *звідки прийшов кожен біт*, і зберігають лише *загальну суму змін*.
4. **Стійкість до колізій.** За двадцять п'ять років публічного криптоаналізу нікому не вдалося знайти два різні тексти, чиї фінальні фото збіглися б. І складність цього завдання виходить за межі обчислювальних можливостей будь-якої цивілізації, яку можна собі уявити.

Додаток із кодом нижче реалізує саме ці шість кроків на мові Zig. Тепер ви можете прочитати його, розуміючи значення кожної бітової операції, а не просто сліпо приймаючи маніпуляції з даними.

Технічний глосарій

Для читача, який хоче зрозуміти, що робить кожна операція. Можете вільно пропустити цей розділ: статтю можна зрозуміти і без нього.

ASCII та Unicode — як літери стають числами. Комп'ютери не бачать літер; вони бачать числа. Стандарт під назвою **ASCII** (1963 рік) призначає кожному символу клавіатури певне число: *A* — це 65, *B* — це 66, *a* — це 97, *0* — це 48, пробіл — 32, кома — 44. Сучасні системи розширюють його за допомогою **Unicode**, який призначає число кожному символу кожного алфавіту світу: кирилиці, арабського письма, китайських та японських ієрогліфів і навіть емодзі. Коли ви вводите символ або відкриваєте текстовий файл, комп'ютер читає число в основі, а не форму на екрані. SHA-256 працює з цими числами, сприймаючи будь-який текст як довгу послідовність цифр. Тому він може однаково обробити статтю іспанською, вірш японською або бінарний файл.
XOR — порозрядне порівняння. XOR (вимовляється як *«екзор»*, від англ. *exclusive or*, «виключне або») — це одна з найпростіших операцій, які комп'ютер може виконувати з двома бінарними числами. Вона порівнює два біти попозиційно і повертає: **1**, якщо рівно один із двох дорівнює 1 (один, але не обидва), і **0**, якщо вони однакові (обидва 0 або обидва 1). Наприклад: XOR для `1010` та `1100` дає `0110`. Операція має чудову властивість: вона зворотна — якщо ви двічі застосуєте XOR з тим самим ключем, ви повернетеся до оригіналу. Тому це робоча конячка криптографії: вона змішує біти без втрати інформації, але результат нічого не говорить про вхідні дані, якщо ви не знаєте одного з них.
Шістнадцяткова система — лічба за основою 16. Майже всі числа у повсякденному житті використовують десять цифр (0–9). Шістнадцяткова система використовує шістнадцять: звичні 0–9 плюс шість літер, що представляють наступні значення: `A` = 10, `B` = 11, `C` = 12, `D` = 13, `E` = 14, `F` = 15. Чому шістнадцять? Тому що комп'ютери мислять групами по чотири біти, а чотири біти можуть представляти рівно шістнадцять різних значень — таким чином, один шістнадцятковий символ чітко відповідає чотирьом бітам. Відбиток SHA-256 має довжину 256 бітів, що становить рівно **64 шістнадцяткові символи**. Якби ми записували його звичайними десятковими числами, він би займав близько 78 цифр і був би менш зручним. Цей вибір естетичний і компактний; число в основі залишається тим самим.
Циклічний зсув бітів — бінарна карусель. Уявіть ряд із семи лампочок, деякі з яких світяться (1), а деякі — ні (0): `1 0 1 1 0 0 1`. Зсув праворуч на одну позицію полягає в тому, щоб взяти крайню праву лампочку, перенести її на крайній лівий край і зсунути решту на одну позицію праворуч: `1 1 0 1 1 0 0`. Жодна лампочка не втрачається і не додається: вони просто рухаються по колу. SHA-256 використовує зсув бітів сотні разів у кожному обчисленні; це дешевий і без втрат спосіб перерозподілу інформації всередині стану.
Константи «nothing-up-my-sleeve» — чому вони походять від простих чисел. Вісім майстер-фішок та шістдесят чотири константи раундів у SHA-256 не були обрані випадково. Вони походять від квадратних та кубічних коренів перших простих чисел. Чому? Тому що дизайнери хотіли константи *«без нічого в рукавах»* (nothing up my sleeve): значення, походження яких будь-хто може перевірити. Якби хтось сказав вам: *«довірся мені: використовуй це випадкове 32-бітне число»*, ви б обґрунтовано запідозрили приховану слабкість або бекдор. Але будь-хто з калькулятором може перевірити, що перші 32 біти квадратного кореня з 2 — це `0x6a09e667`. Значення є математичними, публічними та відтворюваними: жодна пастка не може потрапити в рецепт.

Додаток: SHA-256 у зрозумілому коді

Цей додаток призначений для читача, який хоче побачити алгоритм зсередини. Це навчальна реалізація на Zig, яка відповідає специфікації FIPS 180-4. Це не та версія, яку використовує Solo2 — реальна знаходиться в `std.crypto.hash.sha2.Sha256` стандартної бібліотеки Zig, вона оптимізована та аудійована. Але алгоритм той самий: те, що ви бачите тут, — це крок за кроком те, що відбувається, коли той виклик із п'яти символів виконує свою роботу.

```
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
}
```

Будь-яке переписування іншою мовою, що відповідає тій самій структурі — початкові константи, розширення розкладу, шістдесят чотири раунди, накопичення — дає той самий результат. Алгоритм не має секретів: його цінність полягає в тому, що перелічені вище властивості продовжують зберігатися після двох десятиліть публічного криптоаналізу тисячами очей.

Якщо ви повернетеся до кінця цієї статті, ви побачите шістнадцяткову печатку із шістдесяти чотирьох символів. Це SHA-256 тексту, який ви щойно прочитали, цією мовою. Якби ми переклали статтю, печатка була б іншою; якби змінилося хоча б одне слово в іспанській версії, іспанська печатка змінилася б. Печатка не захищає вміст — для цього є інші інструменти — а однозначно ідентифікує його. І цього, як би скромно це не звучало, достатньо, щоб жоден крок у видавничому ланцюгу не міг змінити сказане непомітно. Усе інше — шифрування, підпис, ідентифікація — будується на цій простій ідеї.

Джерела та додаткова література

- NIST — *FIPS PUB 180-4: Secure Hash Standard (SHS)*, серпень 2015 р. Офіційна специфікація сімейства SHA-2, включаючи SHA-256.
- RFC 6234 — *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, IETF, травень 2011 р. Нормативна версія для розробників.
- Ferguson, N.; Schneier, B.; Kohno, T. — *Cryptography Engineering: Design Principles and Practical Applications* (Wiley, 2010). Розділи 5 і 6 охоплюють хеш-функції та їх законне й незаконне використання.
- Nakamoto, S. — Bitcoin: A Peer-to-Peer Electronic Cash System (2008). Практичний приклад використання SHA-256 для зчеплення блоків у незмінну за конструкцією структуру.
- Регламент (ЄС) 910/2014 (eIDAS) — рамки для кваліфікованих позначок часу. SHA-256 є еталонною функцією для кваліфікованих електронних підписів та печаток, що видаються в ЄС.
- Еталонна реалізація в Zig: `std.crypto.hash.sha2.Sha256` в офіційному репозиторії мови (*github.com/ziglang/zig* → `lib/std/crypto/sha2.zig`). Це оптимізована та аудійована версія, яку фактично використовує Solo2. Корисно для порівняння з навчальною реалізацією в додатку.

← ПопереднійCUADERNOS_LIST_SCHREMS_TITLEНаступний →CUADERNOS_LIST_KILLSWITCH_TITLE

Останні матеріали

- CUADERNOS_LIST_PREGUNTAS_TITLE
- CUADERNOS_LIST_SELFHOST_TITLE
- CUADERNOS_LIST_IDENTIDAD_TITLE

Візьміть цю статтю з собою куди завгодно.

↓ Markdown ↓ Звичайний текст ↓ PDF

Файл буде завантажено на ваш пристрій. Звідти ви можете зберегти його, імпортувати в Solo2 або поділитися ним де завгодно. Cuadernos не вирішує місце призначення за вас.

Сургучна печатка · SHA-256 551bdaab59b1ce9f36a958de0905cd7398a5019ab5b883dc3e1f715e8830d3d4

Cuadernos Lacre · Видання Menzuri Gestión S.L. ·
автор R.Eugenio · під редакцією команди Solo2.